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

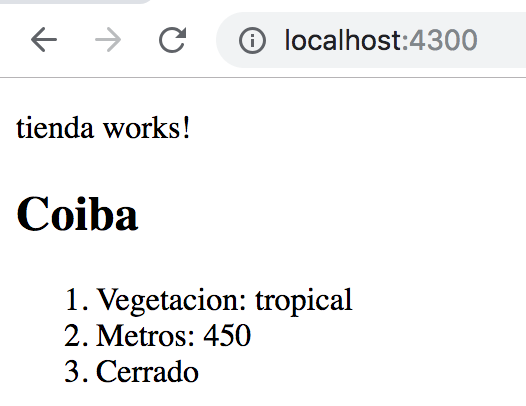


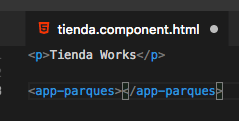
Creando componentes:

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

Ng g c components/tienda

Parques





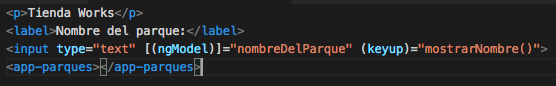


Comunicación entre componentes:

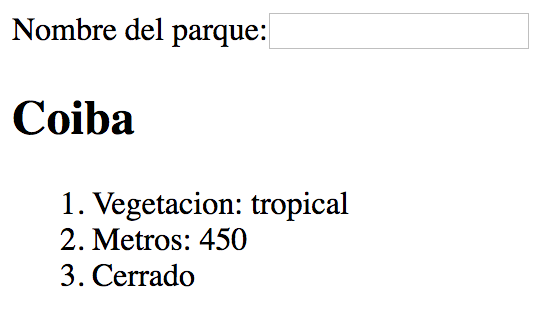
Input y Output

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



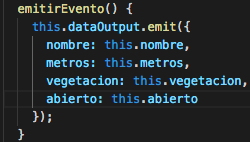


**Output**

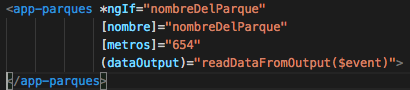
Para enviar datos al componente padre, hay que importar Output desde el componente hijo. Luego crear la variabe siendo una instancia del EventEmitter



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

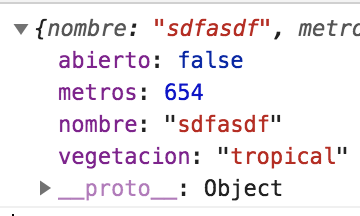


Desde el html del componente padre podemos utilizar el output con la variable declarada, luego llamando una función que recibe como parámetro un event, que realmente son los datos.



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.





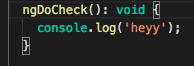
Hooks del ciclo de vida del componente:

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.



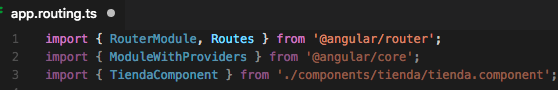
OnDestroy

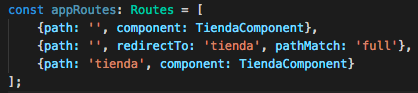
Se ejecuta cuando se destruye un componente. Por ejemplo cuando deja de mostrarse el componente hijo al no tener el nombre del parque en este ejemplo.

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:

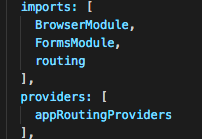


luego se crea una constante de rutas  


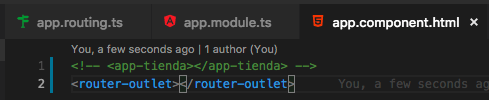
Luego exportando los provider de rutas y el modulo de rutas.:



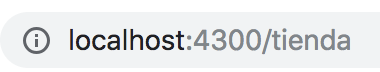
Luego desde el app.module importar estas variables en los imports y en providers:



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.

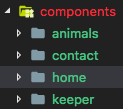


Routers y componentes:

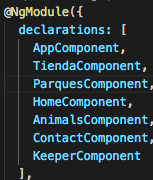
Creando componentes:

Home

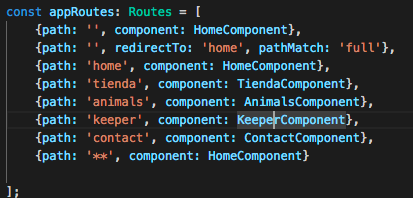




Recordar implementarlos en el app.module en declarations

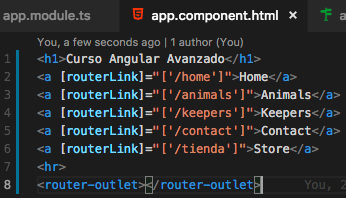


ahora como serán implementados en el nav es necesario incluirlos en el app.routing



Haciendo un menú de navegación

Desde el app componente creamos la navegación utilizando la propiedad routerLink

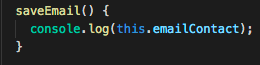




Utilizando localStorage y doCheck:

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





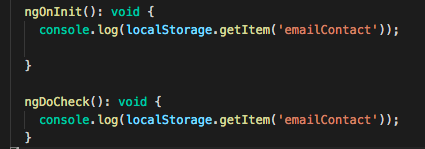
utilizando el localStorage



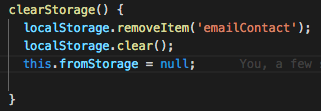
si luego de guardarlo se agrega una impresión del localStorage desde el app.Component

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

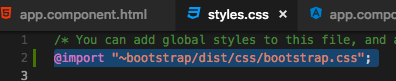
import \* as $ from 'jquery';

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

BOOTSTRAP

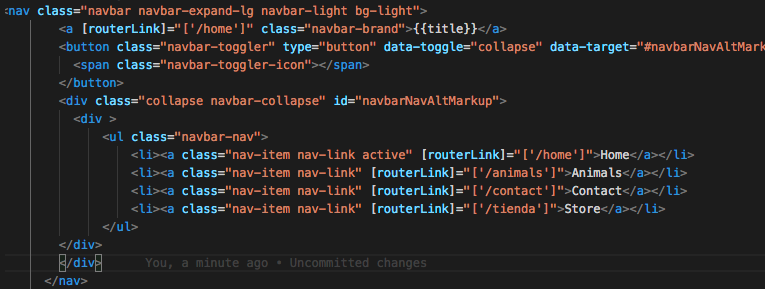
Npm install bootstrap

y desde el styles.css

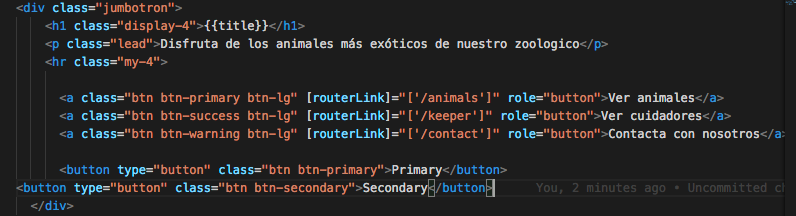


Luego como un plus podemos integrar una hoja de estilo global dentro del assets, styles.

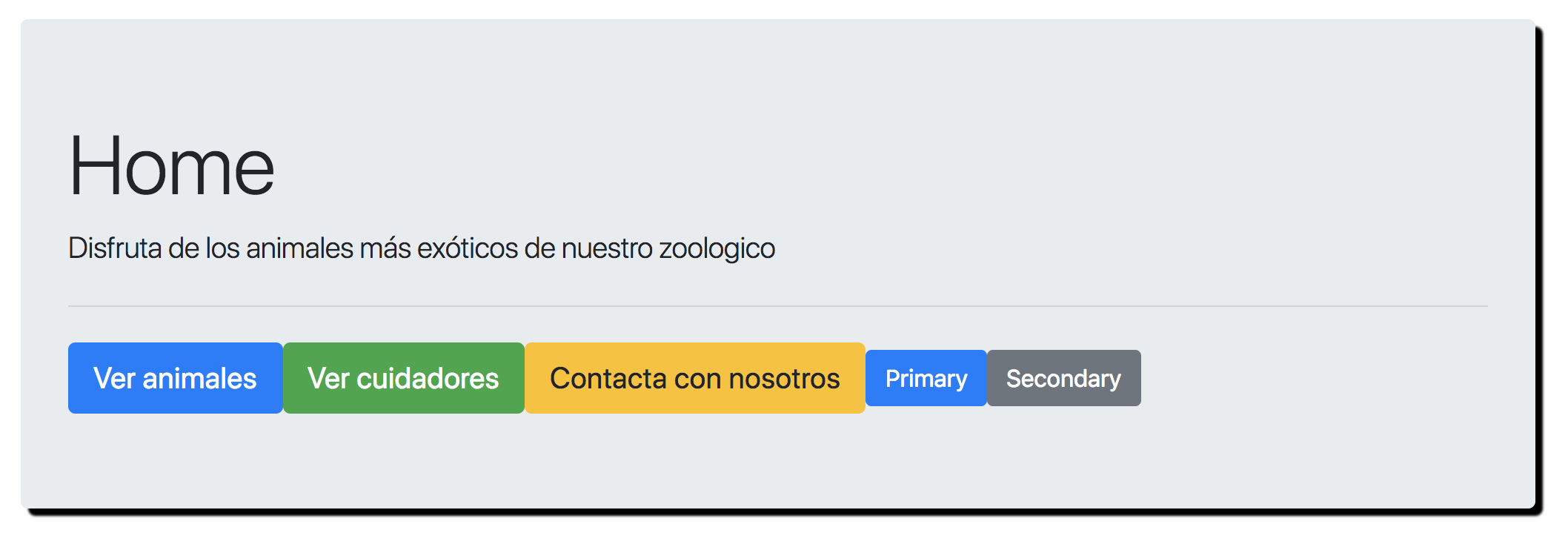
<link rel="stylesheet" type="text/css" href="assets/css/styles.css"

Maquetando el menu utilizando los routerLink 

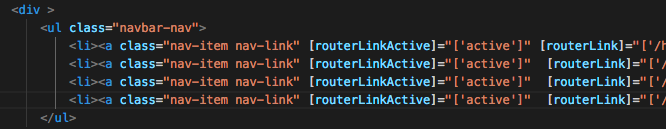
Bootstrap Jumbotron



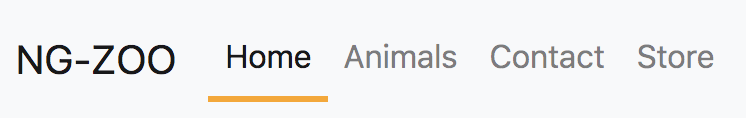




Marcar la ruta actual







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.

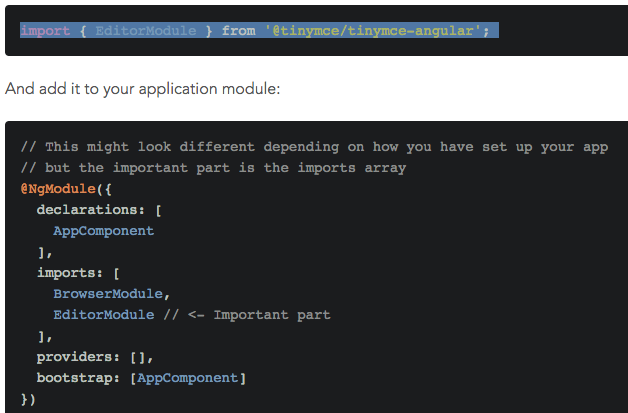
npm i jquery.dotdotdot

¿? No se pudo

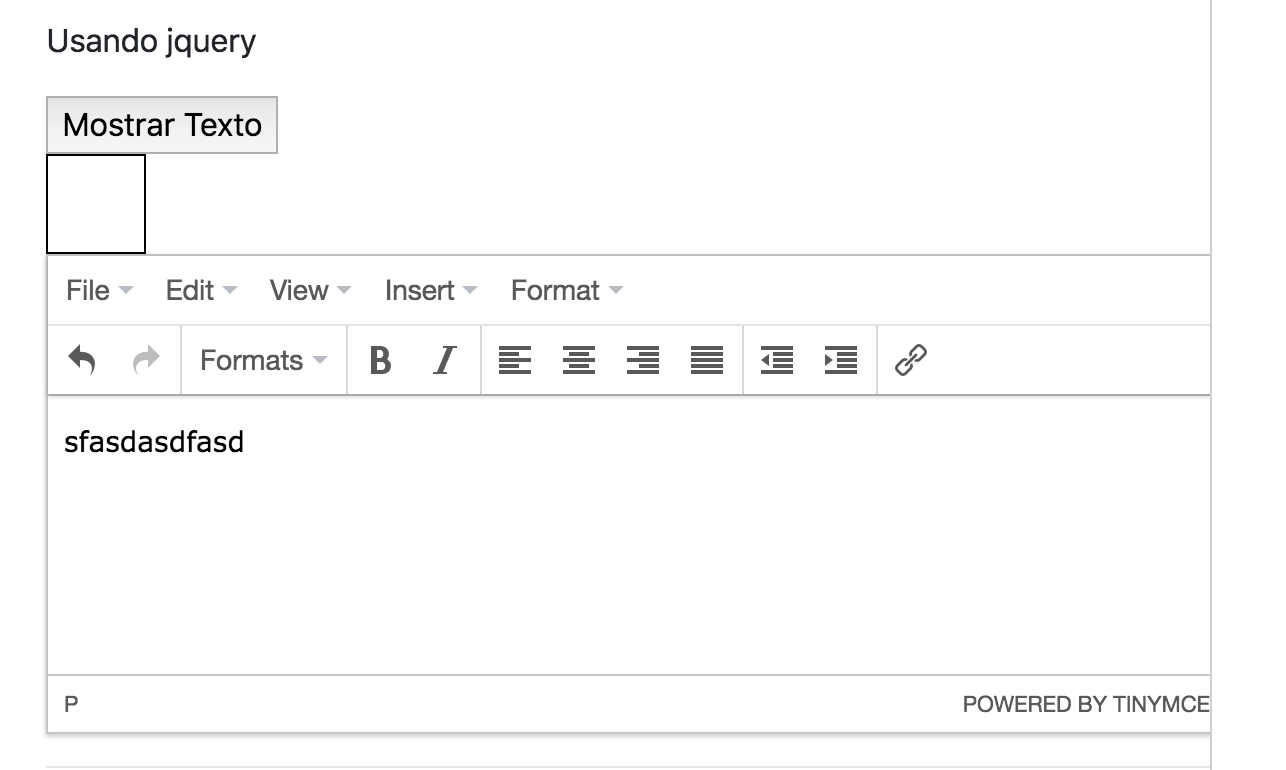
Instalando editor de texto enriquecido:

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



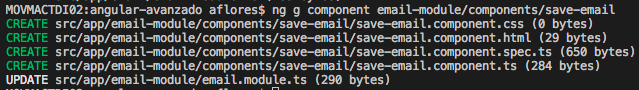
Explicación sobre scripts exteriores no sirve en el curso.

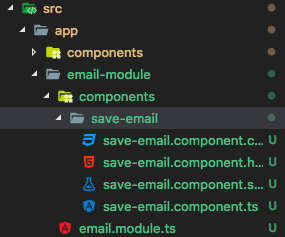
Modularizando webapps con Angular.

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.

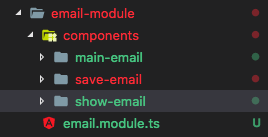




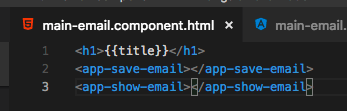
En el componente save-email se pasan las funciones existentes en contact



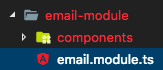
Igual para el show , luego se crea un componente principal en su propia carpeta.



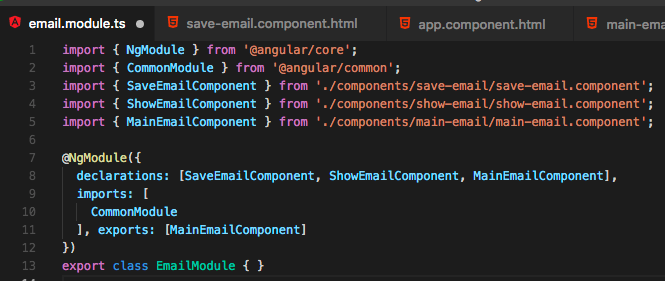
Donde el modulo principal tiene



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

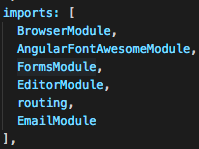


Importando el common module, declarando los componentes que contiene y declarando el modulo principal dentro del export.

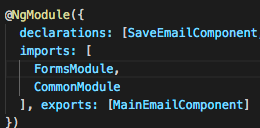


Para cargar el modulo en la aplicación principal hay que importarlo en el app.modules

Dentro de los imports



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



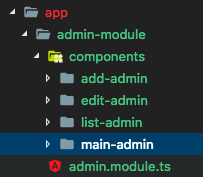
Al estar modularizado podemos utilizarla en varias partes de la aplicación utilizando el mismo tag de app-main-email

Creando un modulo avanzado con rutas:

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



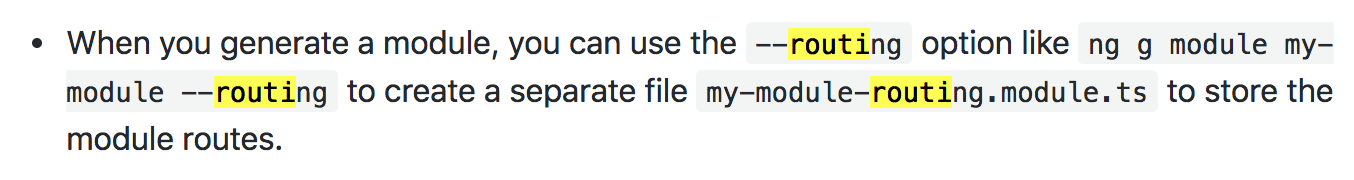
Creamos otros para: añadir, listar, editar:



Luego podemos añadir rutas hijas dentro del mismo modulo.

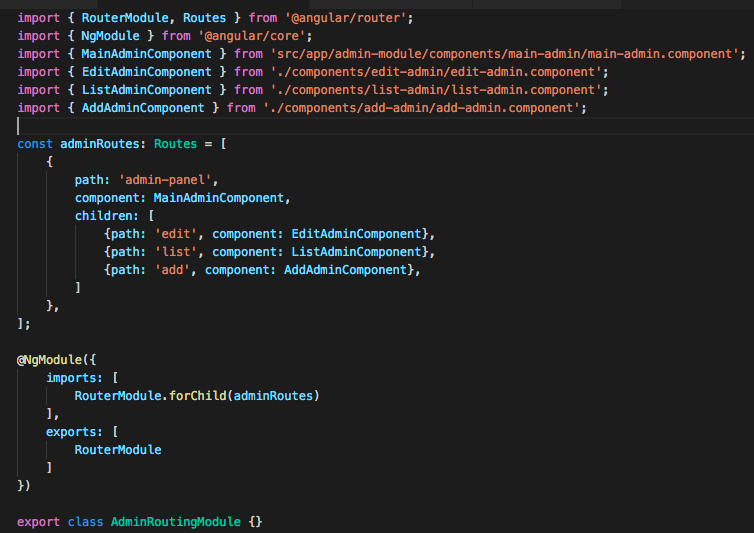
Dentro de la carpeta admin-module se crea el archivo admin-routing.module.ts

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

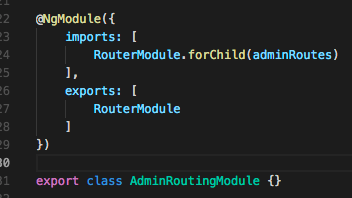


Siendo esta la configuración del archivo:

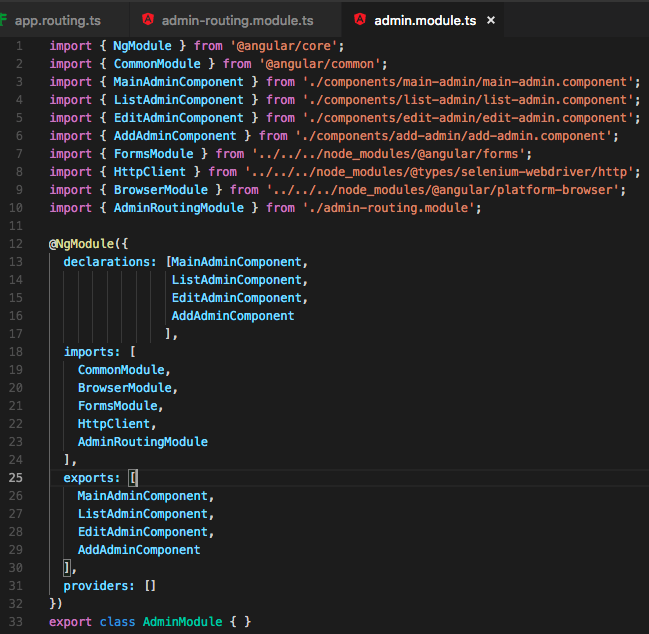
Un arreglo que contiene las rutas, donde hay una ruta principal la cual tiene hijos.  
Esto en el navegador se mostraría: /admin-panel/edit



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

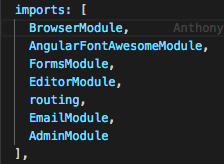


Utilizando el modulo de admin:



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

Y se importa en los imports del app.module:



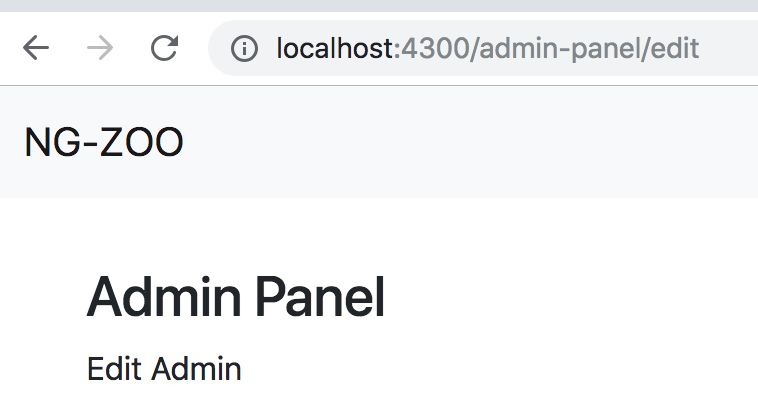
Modificando el routing para que el child route redirija al componente de list.



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

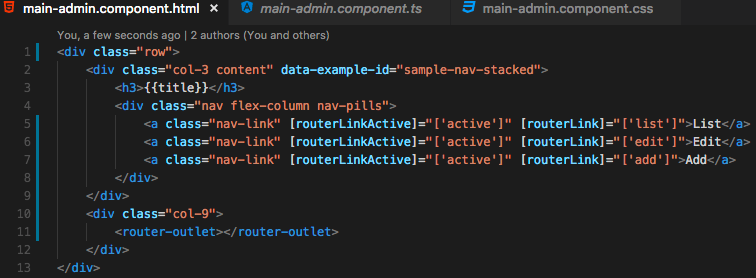
Hay que modificar el main de admin para utilizar el routerOutlet

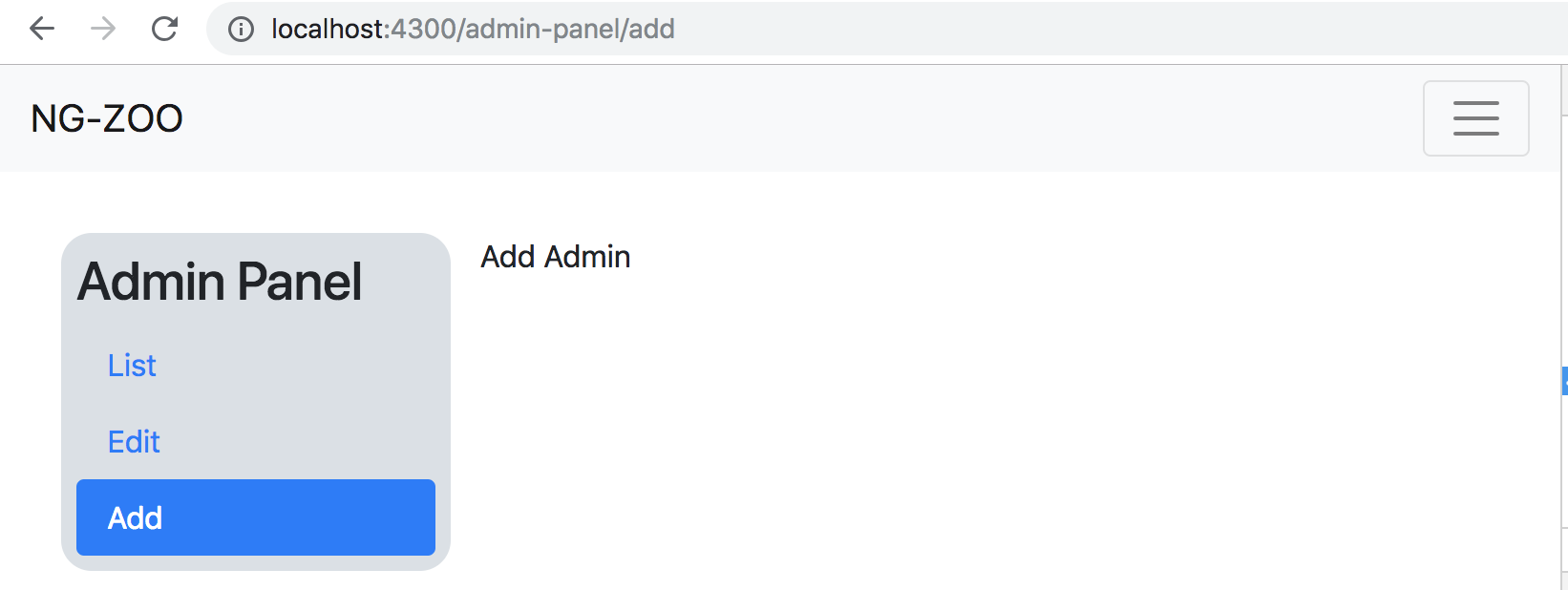




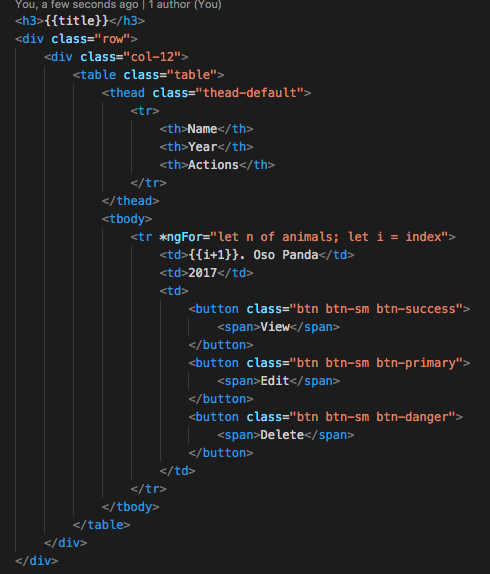
Recordar revisar el HTTP MODULE o lo que sirva en este momento

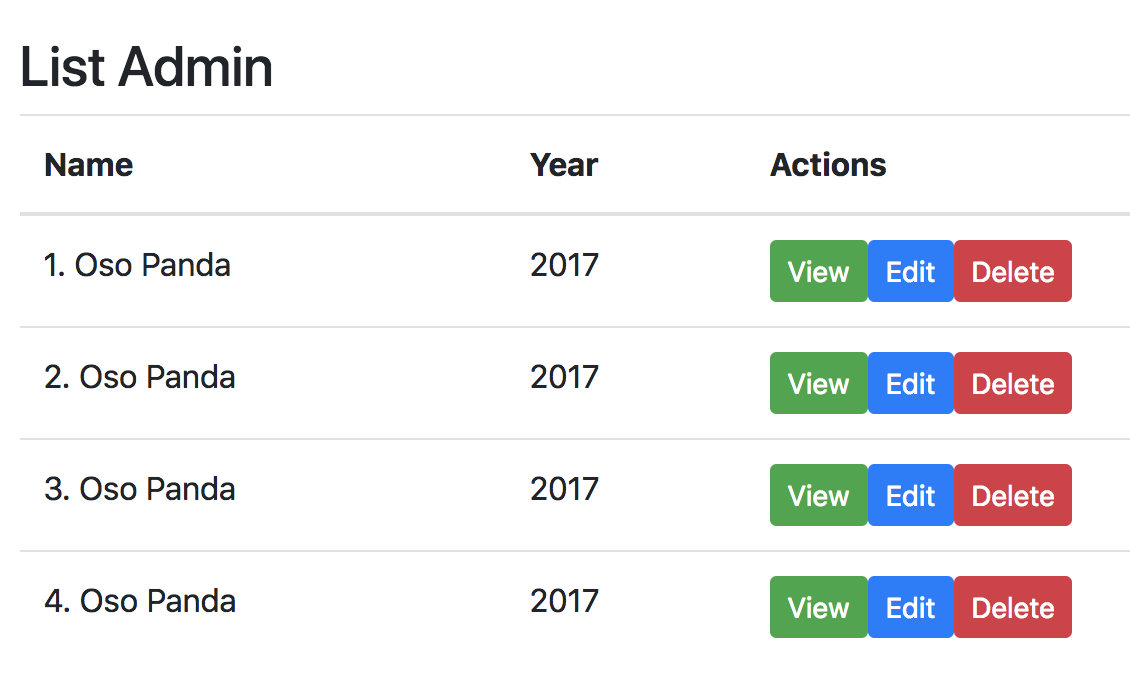
Maquetacion del modulo de panel de administración:



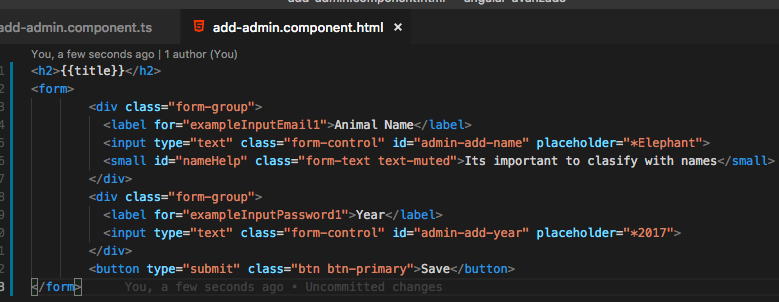


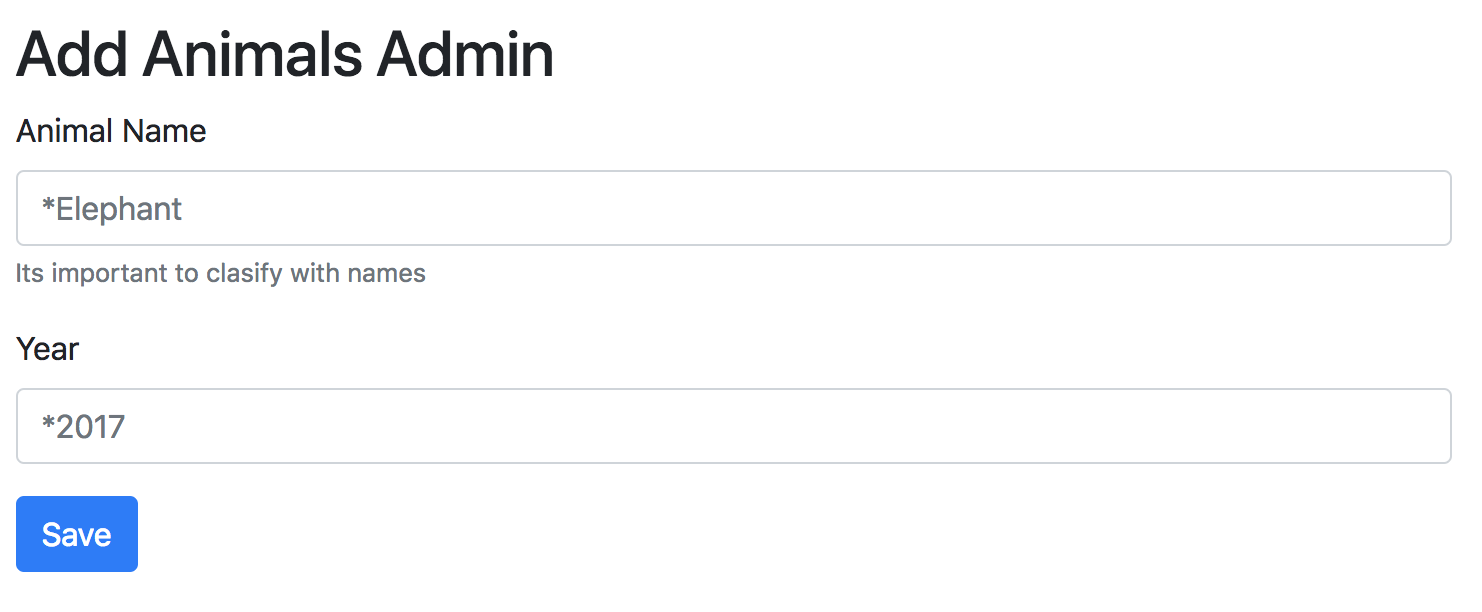
Ahora dentro de cada opción vamos a modificar el html.





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





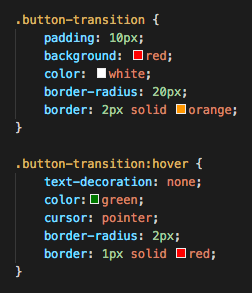
Animaciones CSS3

Transiciones:

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

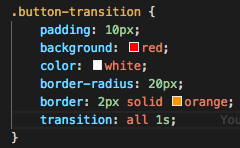


css



Añadiendo transiciones:

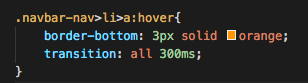
Se puede añadir la propiedad transition, y asignalo a todos los elementos o individual:



o

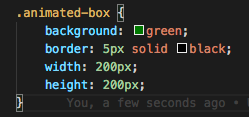


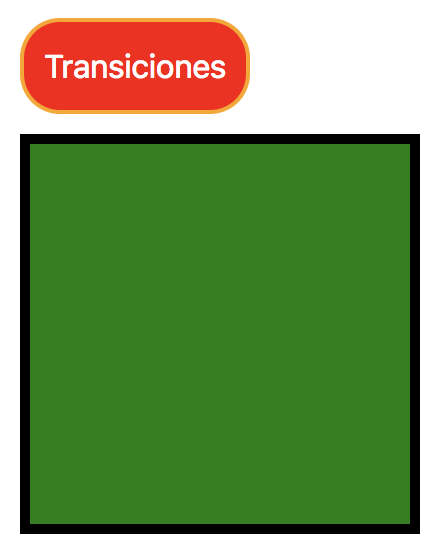
o específicamente a un elemento anidado.



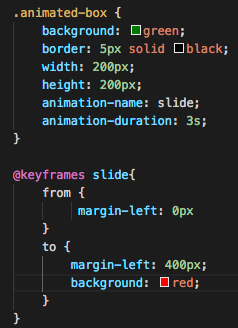
Animaciones:

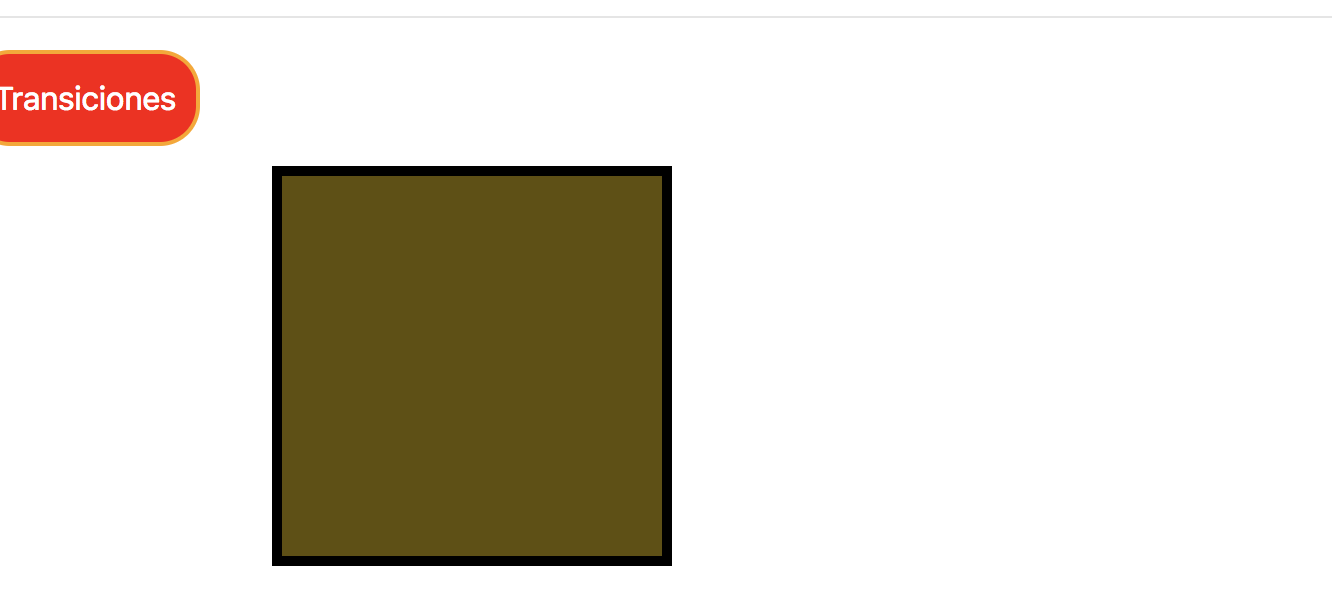
Con la propiedad animation se pueden utilizar keyframes





Añadiendo la clase animation:





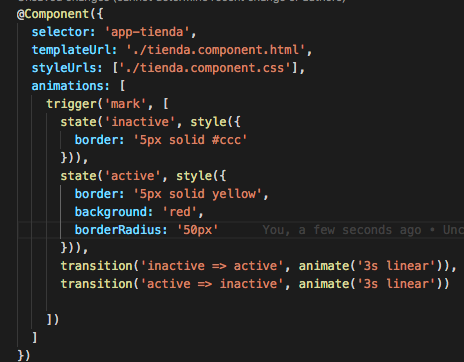
Existe otro tipo de keyframes por porcentajes:



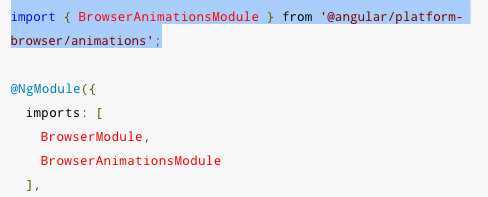
Animaciones con estados de angular:

Hay que importar





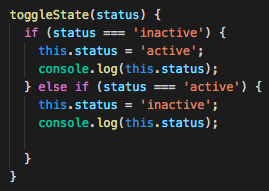
en el app.module



y para poder implementarlo en el html:



Haciendolo interactivo:



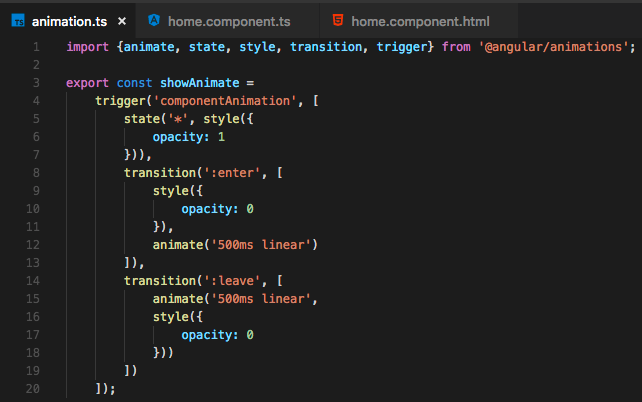


Animación entre componentes:

Se puede realizar animaciones entre rutas cuando entran o salen.

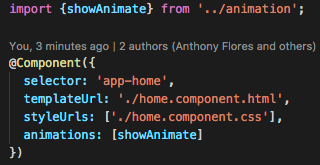
Para esto creamos un archivo especifico, dentro de components se crea el archivo animation.ts

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.

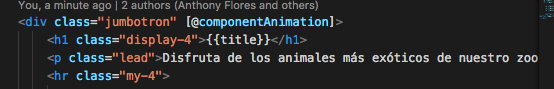


Para utilizar el animations.ts

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.

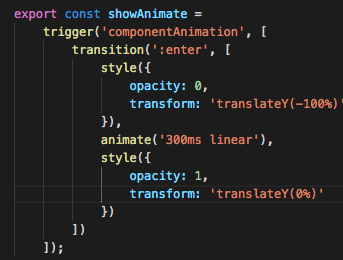


En el caso del leave no se presenta asi que se elimina el código:



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

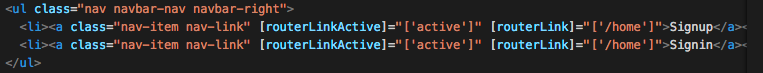
Se puede modificar la animación para que el componente aparezca desde arriba hacia abajo. Con el transform y translateY.



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

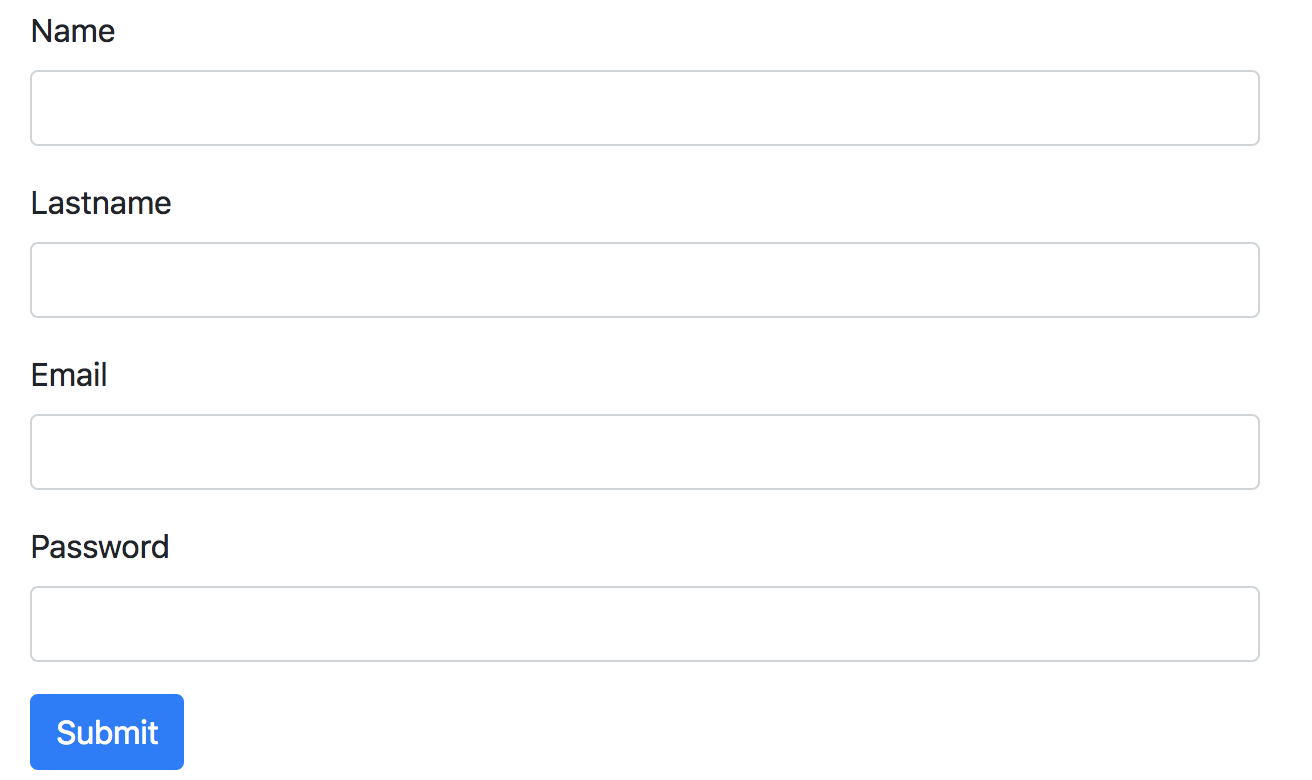


y en el routing



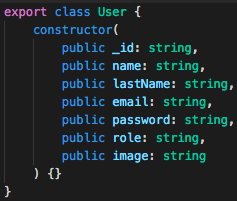
Creando el formulario de registro



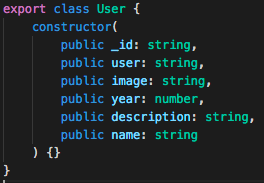


Creando los modelos

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

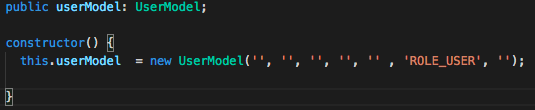


Verificar si tiene id o no



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

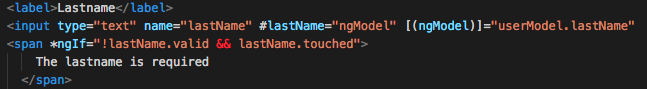


En cada input le vamos asignando un tag describiendo que es parte del ngmodel y luego asignándole directamente el valor con el databinding [()]

Y añadirle validaciones directamente con el modelo



Al igual con los otros campos

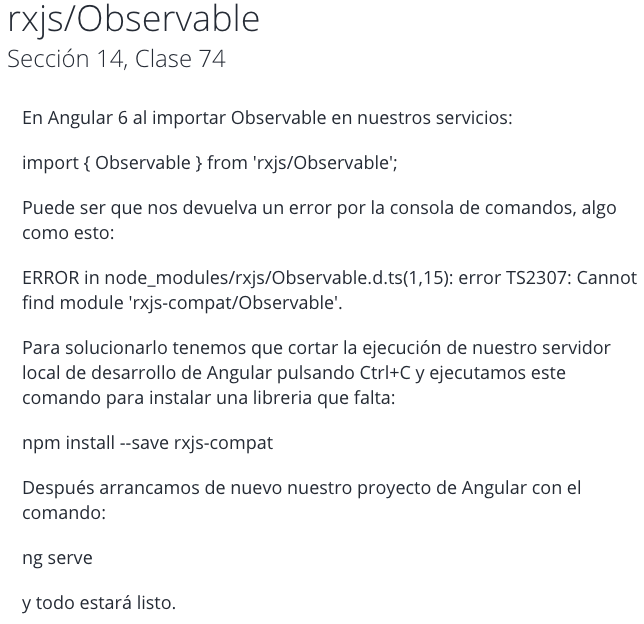


Haciendo el botón de submit disponible o noi dependiendo de que se avalido el formulario completo 

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

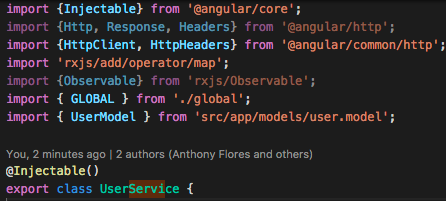
Y el httpClientModule



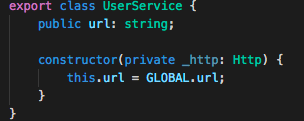


**Ahora se crea user.service.ts**

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.



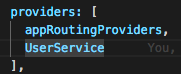
**Creando el método de registro en un componente para pegarle al servicio local**

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

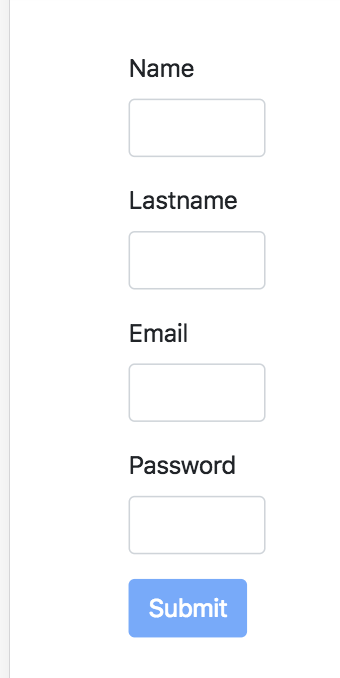




Recordar añadir el servicio desde el app.module providers



En el submit del formulario:

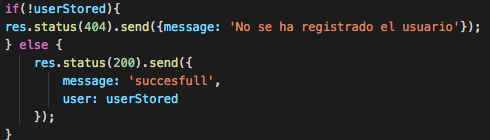


se implementa la funcionalidad del servicio haciendo un subscribe para obtener la respuesta o el error del servicio



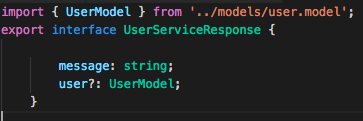
si imprimimos la respuesta del servicio



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


**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

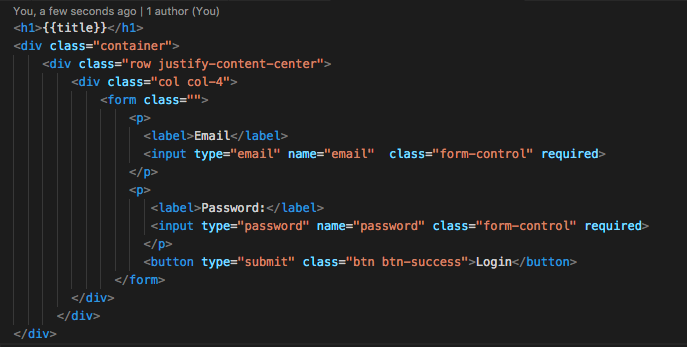


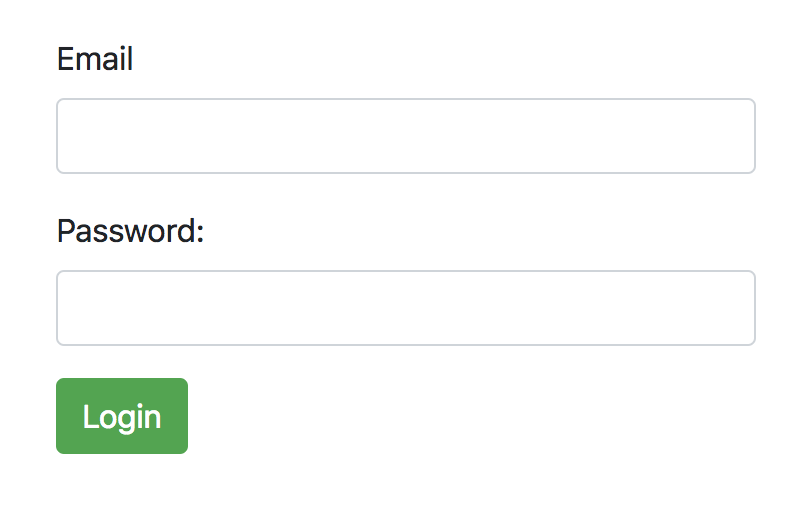
Se crea una variable local que contendrá el mensaje que tiene el servicio y solo se mostrara al recibirlo





**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.