**Historico de la creacion del Proyecto Sici aplicando el stack meam**

**Crear el esqueleto de la aplicación** = ng new Nombre del proyecto

**Crear componentes** = **ng g c nombre del componente** pero si queremos agregarlo a una carpeta simplemente agregaremos en nombre de la carpeta / y nombre del componente ejemplo: **ng g component componentes/home.**

**Ayuda de comandos de angular:** ng –help y me traerá todos los comandos

**Para que me cree el componente pero sin una carpeta que lo contenga uso** **- - flat:**

**ng g component pages/mainpage -is --spec=false –flat**

**ahora creemos servicios:** ng g s services/usuario - -spec=false: me creara una carpeta en la ruta app/con nombre de services y dentro el servicio usuario es decir app/services/usuario.

**Generar guard** = un guard no es mas que un servicio y lo generamos de la siguiente manera:

ng g g guards/auth –spec=false

**Iniciando con PreSici**

1: generando el component login en app/login

**ng g c login --spec=false -is**

2:generando el component nopagefound para todas las rutas no existentes:

**ng g c nopagefound --spec=false -is**

3:generemos el dashboard es decir el home en este caso pero quiero que este dentro de una carpeta llamada pages es decir app/pages/dashboard

**ng g c pages/dashboard --spec=false -is**

4: otros componentes personalizados en este caso se llamara proges y estará en la misma pages

**ng g c pages/progress --spec=false -is**

5:otro componente mas llamado graficas1

**ng g c pages/graficas1 --spec=false -is**

6:ahora en el nivel del app crearemos una carpeta shared y dentro todos los componentes como navbar, sidebar y otras cosas

**ng g c shared/header --spec=false –is:**sera como nuestro home

**ng g c shared/sidebar --spec=false –is:**sera nuestro menu

**ng g c shared/breadcrumbs --spec=false –is:**sera nuestro navegador interno of pages

7:movamos el componente pagenofound a shared acuérdate de cambiar la ruta en el app.module.ts.

8-los servicios los crearemos por el momento en el nivel 1 de app en una carpeta de servicios:

ng g s services/sidebar --spec=false

ng g s services/shared --spec=false

**Agregando librerías que necesita el témplate.**

1:Busquemos el template descargados en al carpeta main copiemos las carpetas css,js y scss y peguemosla en los assets.

2:ahora busquemos el template llamado pages-blank.html y arrastremoslo al visual studio code.

Esto para ver las importaciones por ejemplo copiaresmos el link que inponta el icono y lo pegamos en el index.html de nuestro proyecto.

*<*link rel="icon" type="image/png" sizes="16x16" href="../assets/images/favicon.png"*>*

3:hay que corregir porque las imagines no las tenemos en ..assets sino en assets enyonces

*<*link rel="icon" type="image/png" sizes="16x16" href="assets/images/favicon.png"*>*

4:nos copiamos también el boostrap que es en este caso el 4 y tambien el estilo default-dark.css

*<!-- Bootstrap Core CSS -->*

*<*link href="../assets/plugins/bootstrap/css/bootstrap.min.css" rel="stylesheet"*>*

*<!-- Custom CSS -->*

*<*link href="css/style.css" rel="stylesheet"*>*

*<!-- You can change the theme colors from here -->*

*<*link href="css/colors/default-dark.css" id="theme" rel="stylesheet"*>*

5: y arreglamos las rutas ya que todo esto apunta a nuestros assets, quedaría asi:

*<!-- Bootstrap Core CSS -->*

*<*link href="assets/plugins/bootstrap/css/bootstrap.min.css" rel="stylesheet"*>*

*<!-- Custom CSS -->*

*<*link href="css/style.css" rel="stylesheet"*>*

*<!-- You can change the theme colors from here -->*

*<*link href="css/colors/default-dark.css" id="theme" rel="stylesheet"*>*

6:Aun el bootstrap no esta pero lo arreglaremos copiando del material descargado el bootstrap que estara en assets/plugins/bootstrap, no copiemos todos los plugins porque son muchos solo copiemos el bootstrap entocnes deberemos crear la carpeta plugins en los assets

7: ahora sigamos con los js, que están abajo en el template, en este caso pugins de jquery que usa el template, el bootstrap, perfec-scrolbarr que es para que se vea mas bonito, el waves para que los botones tengan el efecto del material, el sidemenu, es decir todos los js.

*<*script src="../assets/plugins/jquery/jquery.min.js"*></*script*>*

*<!-- Bootstrap tether Core JavaScript -->*

*<*script src="../assets/plugins/bootstrap/js/popper.min.js"*></*script*>*

*<*script src="../assets/plugins/bootstrap/js/bootstrap.min.js"*></*script*>*

*<!-- slimscrollbar scrollbar JavaScript -->*

*<*script src="js/perfect-scrollbar.jquery.min.js"*></*script*>*

*<!--Wave Effects -->*

*<*script src="js/waves.js"*></*script*>*

*<!--Menu sidebar -->*

*<*script src="js/sidebarmenu.js"*></*script*>*

*<!--stickey kit -->*

*<*script src="../assets/plugins/sticky-kit-master/dist/sticky-kit.min.js"*></*script*>*

*<*script src="../assets/plugins/sparkline/jquery.sparkline.min.js"*></*script*>*

*<!--Custom JavaScript -->*

*<*script src="js/custom.min.js"*></*script*>*

*<!-- ============================================================== -->*

*<!-- Style switcher -->*

*<!-- ============================================================== -->*

*<*script src="../assets/plugins/styleswitcher/jQuery.style.switcher.js"*></*script*>*

8:hay muchos errores puesto que no encuentra los achivos entonces deberemos primero meter todo en la carpeta de assets y segundo corregir las rutas. Después nos seguirá marcando error diciéndonos que falta jquery entonces vayamos a los plugins del template descargado, copio jquery y lo pego en la carpeta plugins de nuestro proyecto, al lado de bootstrap y el resto como sticky-master,styleswitcher,sparline etc los podemos encotrar en los plugins del template descargado y pegarlos en nuestro proyecto.

Quedaría asi:

*<*app-root*></*app-root*>*

*<!-- All Jquery -->*

*<!-- ============================================================== -->*

*<*script src="assets/plugins/jquery/jquery.min.js"*></*script*>*

*<!-- Bootstrap tether Core JavaScript -->*

*<*script src="assets/plugins/bootstrap/js/popper.min.js"*></*script*>*

*<*script src="assets/plugins/bootstrap/js/bootstrap.min.js"*></*script*>*

*<!-- slimscrollbar scrollbar JavaScript -->*

*<*script src="assets/plugins/perfect-scrollbar/js/perfect-scrollbar.jquery.min.js"*></*script*>*

*<!--Wave Effects -->*

*<*script src="assets/js/waves.js"*></*script*>*

*<!--Menu sidebar -->*

*<*script src="assets/js/sidebarmenu.js"*></*script*>*

*<!--stickey kit -->*

*<*script src="assets/plugins/sticky-kit-master/dist/sticky-kit.min.js"*></*script*>*

*<*script src="assets/plugins/sparkline/jquery.sparkline.min.js"*></*script*>*

*<!--Custom JavaScript -->*

*<*script src="assets/js/custom.min.js"*></*script*>*

*<!-- ============================================================== -->*

*<!-- Style switcher -->*

*<!-- ============================================================== -->*

*<*script src="assets/plugins/styleswitcher/jQuery.style.switcher.js"*></*script*>*

*</*body*>*

*<*script src="assets/plugins/sticky-kit-master/dist/sticky-kit.min.js"*></*script*>*

*<*script src="assets/plugins/sparkline/jquery.sparkline.min.js"*></*script*>*

*<!--Custom JavaScript -->*

*<*script src="assets/js/custom.min.js"*></*script*>*

*<!-- ============================================================== -->*

*<!-- Style switcher -->*

*<!-- ============================================================== -->*

*<*script src="assets/plugins/styleswitcher/jQuery.style.switcher.js"*></*script*>*

**Creemos el contenido de Breadcrumbs, Header y Sidebar:**

Sigamos usando el pages-blanck.html del template descargado y vayamos hasta el body y copiemos toda la clase y peguemosla en el index.html de nuestro proyecto.

*<*body class="fix-header card-no-border fix-sidebar"*>*

Quedaria de esta manera

*<*body class="fix-header card-no-border fix-sidebar"*>*

*<*app-root*></*app-root*>*

Usemos el prelouder, este muestra un pequeño loading que se quita cuando dispra la funcon listo de jquery, usémoslo también.

Lo clocaremos debajo del body justo antes del app-root

*<*body class="fix-header card-no-border fix-sidebar"*>* *<!-- ============================================================== -->*

*<!-- Preloader - style you can find in spinners.css -->*

*<!-- ============================================================== -->*

*<*div class="preloader"*>*

*<*div class="loader"*>*

*<*div class="loader\_\_figure"*></*div*>*

*<*p class="loader\_\_label"*>*Admin Pro*</*p*>*

*</*div*>*

*</*div*>*

*<*app-root*></*app-root*>*

Ahora empecemos, podemos observer que viene un div con id main-wrapper, desde aquí inicia el template. Y esto lo pegaremos en el app.component.html que es la vista principal de los componentes y allí borraremos todo y pegaremos ese div allí.

Debería quedar asi.

*<*div id="main-wrapper"*>*

*</*div*>*

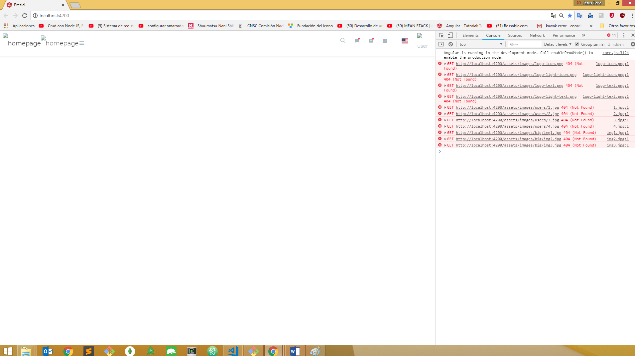
Luego viene lo que es el header copiaremos todo y lo pegaremos en la vista del componente creado header, luego abrimos el .ts del header y notemos que para usar el header utilizamos el selector app-header. Ese selector lo copiare me ire a app.component.html y allí dentro lo pegare para ir armando la estructura. Es decir:

*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

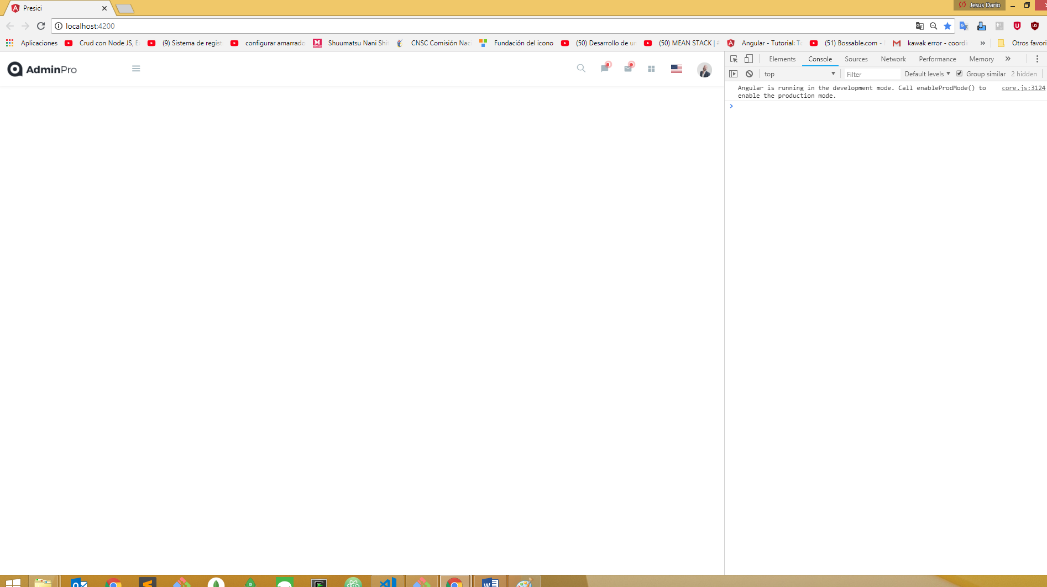
*</*div*>*

En cuanto recarguemos podremos ver que el header esta funcionando.

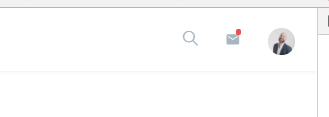


Los errores que salen es porque no hemos corregido en el header las rutas de los elementos asociados vayamos y corrijamos todo el bloque de assets que esta con ../ cambiémoslo por assets solo y corrijamos el resto. Cuando hagamos esto nos pedirá la carpeta de imágenes, copiémoslas del template descargado y agreguemosla a nuestra carpeta de assets.

Podemos observar que todo quedo ok sin ningún error.



Quitemos algunas cosas en nuestro header es decir los iconos de notificaciones y el megamenu también comentemos lo de los lenguajes es posible que lo usemos mas adelante. Quedando de la siguiente forma.



Listo ahora viene el sidebar asi que vayamos a nuestro template descargado y copiemos todo el left sidebar y lo pegaremos en la vista del componente sidebar, luego iremos al app.component.html y lo implementaremos con la respectiva etiqueta.

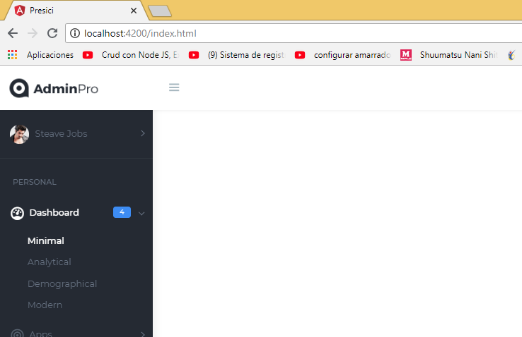
*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

*<*app-sidebar*></*app-sidebar*>*

*</*div*>*

Se vera de la siuient manera:



Podemos observar que hay muchos elementos quedémonos con lo que nos sirve y dejemos solo lo del profile y personal el resto borremos todo. Debería quedar asi

*<!-- Left Sidebar - style you can find in sidebar.scss -->*

*<!-- ============================================================== -->*

*<*aside class="left-sidebar"*>*

*<!-- Sidebar scroll-->*

*<*div class="scroll-sidebar"*>*

*<!-- Sidebar navigation-->*

*<*nav class="sidebar-nav"*>*

*<*ul id="sidebarnav"*>*

*<*li class="user-profile"*>*

*<*a class="has-arrow waves-effect waves-dark" href="#" aria-expanded="false"*><*img src="../assets/images/users/profile.png" alt="user" */><*span class="hide-menu"*>*Steave Jobs *</*span*></*a*>*

*<*ul aria-expanded="false" class="collapse"*>*

*<*li*><*a href="javascript:void()"*>*My Profile *</*a*></*li*>*

*<*li*><*a href="javascript:void()"*>*My Balance*</*a*></*li*>*

*<*li*><*a href="javascript:void()"*>*Inbox*</*a*></*li*>*

*<*li*><*a href="javascript:void()"*>*Account Setting*</*a*></*li*>*

*<*li*><*a href="javascript:void()"*>*Logout*</*a*></*li*>*

*</*ul*>*

*</*li*>*

*<*li class="nav-devider"*></*li*>*

*<*li class="nav-small-cap"*>*PERSONAL*</*li*>*

*<*li*>* *<*a class="has-arrow waves-effect waves-dark" href="#" aria-expanded="false"*><*i class="mdi mdi-gauge"*></*i*><*span class="hide-menu"*>*Dashboard *<*span class="label label-rouded label-themecolor pull-right"*>*4*</*span*></*span*></*a*>*

*<*ul aria-expanded="false" class="collapse"*>*

*<*li*><*a href="index.html"*>*Minimal *</*a*></*li*>*

*<*li*><*a href="index2.html"*>*Analytical*</*a*></*li*>*

*<*li*><*a href="index3.html"*>*Demographical*</*a*></*li*>*

*<*li*><*a href="index4.html"*>*Modern*</*a*></*li*>*

*</*ul*>*

*</*li*>*

*</*ul*>*

*</*nav*>*

*<!-- End Sidebar navigation -->*

*</*div*>*

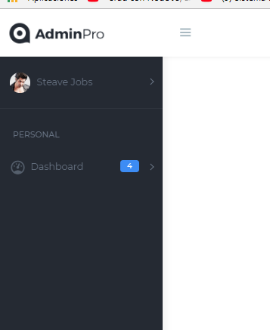
*<!-- End Sidebar scroll-->*

*</*aside*>*

*<!-- ============================================================== -->*

*<!-- End Left Sidebar - style you can find in sidebar.scss -->*

Y la vista renderizada se deberia vera si:



Ahora viene el **page-wrapper** esto es necesario para que el template funcione bien ese seria el envoltorio de lo que será el content principal y lo colocare en el app.component.html después del sidebar.

Tambein el **container-fluid** de adentro **page-wrapwer** del que es del bootstrap también

*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

*<*app-sidebar*></*app-sidebar*>*

*<!-- ==================page-wrapper==================================== -->*

*<*div class="page-wrapper"*>*

*<!-- ============================================================== -->*

*<!-- Container fluid -->*

*<!-- ============================================================== -->*

*<*div class="page-wrapper"*>*

*</*div*>*

*</*div*>*

*<!-- =================end-page-wrapper============================= -->*

*</*div*>*

Luego también dentro vendrá el div del bread crumb que será el directorio interno lo compiamos completo y pegamos en el componente breadcrumbs en su vista y quitemos el ultimo div que tiene un botón ya que no lo usaremos. Y al app.component.html traeremos la etiqueda de configuración del componente breadcrumbs.

Por ultimo debajo colocaremos el div donde ira todo el contenido que es el star-page-content esto completo debería quedar asi:

*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

*<*app-sidebar*></*app-sidebar*>*

*<!-- ==================page-wrapper==================================== -->*

*<*div class="page-wrapper"*>*

*<!-- ============================================================== -->*

*<!-- Container fluid -->*

*<!-- ============================================================== -->*

*<*div class="container-fluid"*>*

*<!-- ============================================================== -->*

*<!-- Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<*app-breadcrumbs*></*app-breadcrumbs*>*

*<!-- ============================================================== -->*

*<!-- end Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<!-- ============================================================== -->*

*<!-- Start Page Content -->*

*<!-- ============================================================== -->*

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

This is some text within a card block.

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<!-- ============================================================== -->*

*<!-- End PAge Content -->*

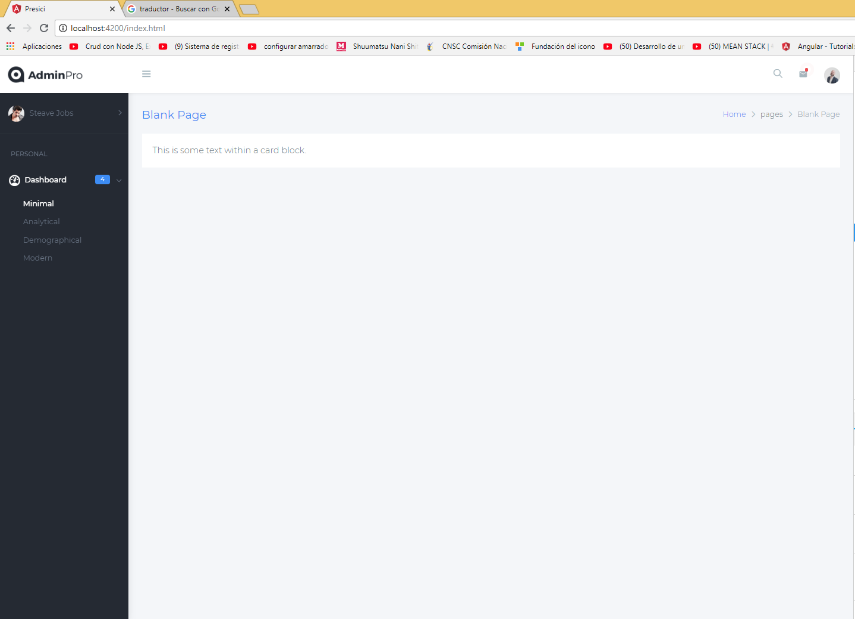
*<!-- ============================================================== -->*

*</*div*>*

*<!-- =================end-page-wrapper============================= -->*

*</*div*>*

Y el reder asi:



**Implementando las rutas principales:**

Aquí implantaremos las rutas que cambiaran el contenido que se encuentre en medio.

Entones creemos en app el archivo app.routes.ts, este será mi archivo de rutas principales.

1: importamos el modulo routes.

import { Routes } from '@angular/router';

2-Definiremos una constante appRoutes de tipo Routes para definir el arreglo de rutas. Cuando termino de agregar el componente y le doy enter el me importara automáticamente el componente. Quedando la importación de componentes y la constante de la siguiente forma.

import { Routes } from '@angular/router';

import { DashboardComponent } from './pages/dashboard/dashboard.component';

const appRoutes : Routes = [

{path: 'dasboard', component: DashboardComponent}

]

Y asi harrems ccon los componentes login,register que aun no hemos cread pero lo colocaremos, tambien el progress,grafica1, la redirecciona cualquier ruta distinta para que mande al dashboard y la de error.

import { Routes } from '@angular/router';

import { DashboardComponent } from './pages/dashboard/dashboard.component';

import { LoginComponent } from './login/login.component';

import { ProgressComponent } from './pages/progress/progress.component';

import { Graficas1Component } from './pages/graficas1/graficas1.component';

import { NopagefoundComponent } from './shared/nopagefound/nopagefound.component';

const appRoutes : Routes = [

{path: 'dasboard', component: DashboardComponent},

{path: 'login', component: LoginComponent},

{path: 'register', component: LoginComponent},

{path: 'progress', component: ProgressComponent},

{path: 'graficas1', component: Graficas1Component},

{path: '', redirectTo: '/dashboard' pathMath: ‘full’},

{path: '\*\*', component: NopagefoundComponent}

]

Ahora necesitamos tambien importar al lado del modulo Router el RouterModule

import { RouterModule, Routes } from '@angular/router';

y exportemos el modulo de la siguiente manera: tambien usemos el hash de esta manera podre enviar un objeto useHash.

export const APP\_ROUTES = RouterModule.forRoot(appRoutes, {useHash: true});

con esto basta para que funcione la aplicación

ahora para usar el app.routing.ts debere importarlo en el app.module.ts

import { APP\_ROUTES } from './app.routes';

y lo agregaremos l array de los imports.

imports: [

BrowserModule,

APP\_ROUTES

],

Ahora deberemos usar la etiqueta <router-outlet> para activar el ruteo y eso lo haremos en el app.component.html en el star Page

*<!-- Start Page Content -->*

*<!-- ============================================================== -->*

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*router-outlet*></*router-outlet*>*

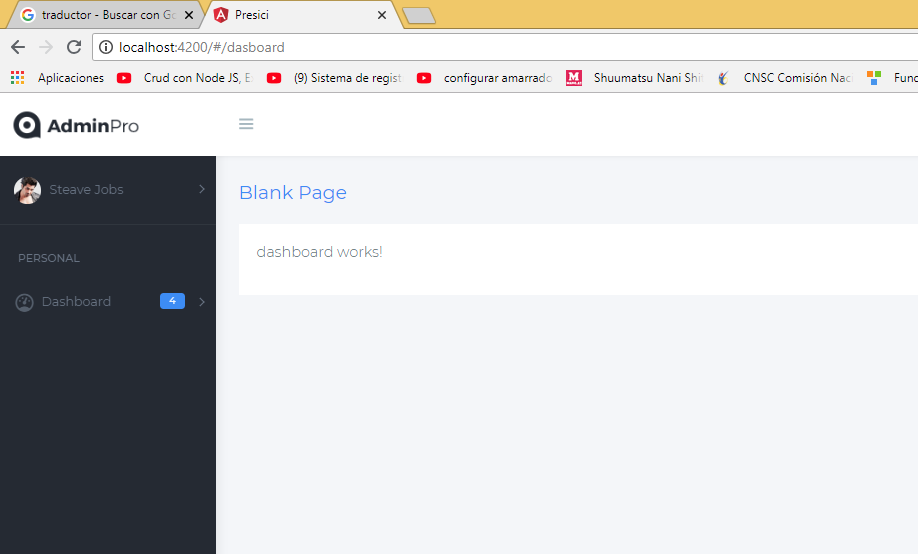
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Lo visualizaremos de la siguiente manera.



Y si modificamos las rutas tambien cambiaran.

**Implementemos la vista de login:**

Deberemos crear un nuevo componente en la carpeta pages con el siguiente comando.

ng g c pages/pages --flat -is --spect=false esto nos creara el .ts y .html dentro de la carpeta pero sin una subcarpeta que lo contenga, no creara tampoco el css y tampoco el spec.

Es decir tendre 2 nuevos achivos en la ruta

App/pages/pages.component.html

App/pages/pages.component.ts

Ahora todo el contenido del app.component.html cortemoslo y peguemoslo en el pages.component.html.

En app.component.html colocare la etiqueta de routeo <router-outlet></ router-outlet>

Ahora en el archive de rutas cambiare las rutas del dashboarsd para que apunte a pages.component para que me muestre nuevamente la vista inicial

{path: '', component: PagesComponent},

Ahora implementaremos unas rutas hijas o secundarias para que el contenido vaya cambiando mientras nuestra ruta padre sea path‘ ’, Entonces en el app.routter.ts declararemos las rutas hijas dentro del path que acabamos de modificar. Esto lo haremos de la siguiente manera.

const appRoutes : Routes = [

{

path: '',

component: PagesComponent,

children:[

{path: 'dashboard', component: DashboardComponent},

{path: 'progress', component: ProgressComponent},

{path: 'graficas1', component: Graficas1Component},

{path: '', redirectTo: '/dashboard', pathMatch:'full'}

]

},

{path: 'login', component: LoginComponent},

{path: 'register', component: LoginComponent},

{path: '\*\*', component: NopagefoundComponent}

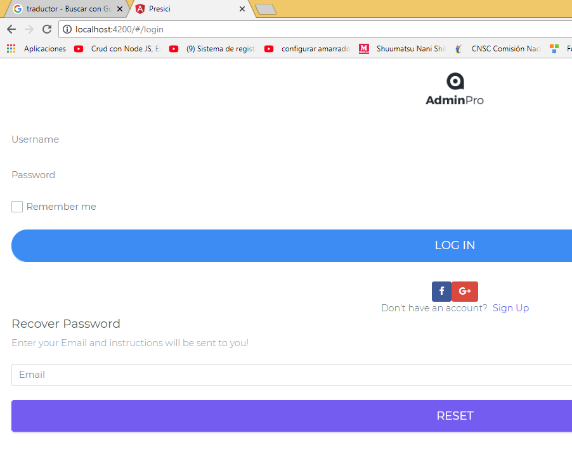
]

De este modo volverá a funcionar el dashboard y sus rutas hijas desde el page.component.ts, asi separamos todo de la vista login y register.

Ahora creemos la pagina de login, abramos en visual page-login2.htm del material descargado.

Podremos observar unos estilos existe uno en particular que tomaremos que es el page/login-register-lock.css, el preloader y luego la sección del login, toda esa sección la copiaremos.

Vamos al componente login y en el html pegamos todo lo anterior, esto es lo que aparece si escribimos en nuestra url la siguiente direccion. <http://localhost:4200/#/login>



Obvio se ve mal porque no importamos los estilos que tenia originalmente asi que lo que haremos será buscar en la documentación de descarga del proyecto se css que el page-login2.html tiene.

*<*link href="css/pages/login-register-lock.css" rel="stylesheet"*>*

Busquemos entonces en el template ese estilo y abrámoslo en visual copiemos todo. Ahora en el login creemos un estilo solo para ese componente se llamara **login.component.css** y allí pegaremos todo lo copiado.

Después en el login.component.ts le definiremos los estilos que acabamso de crear

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

constructor() { }

ngOnInit() {

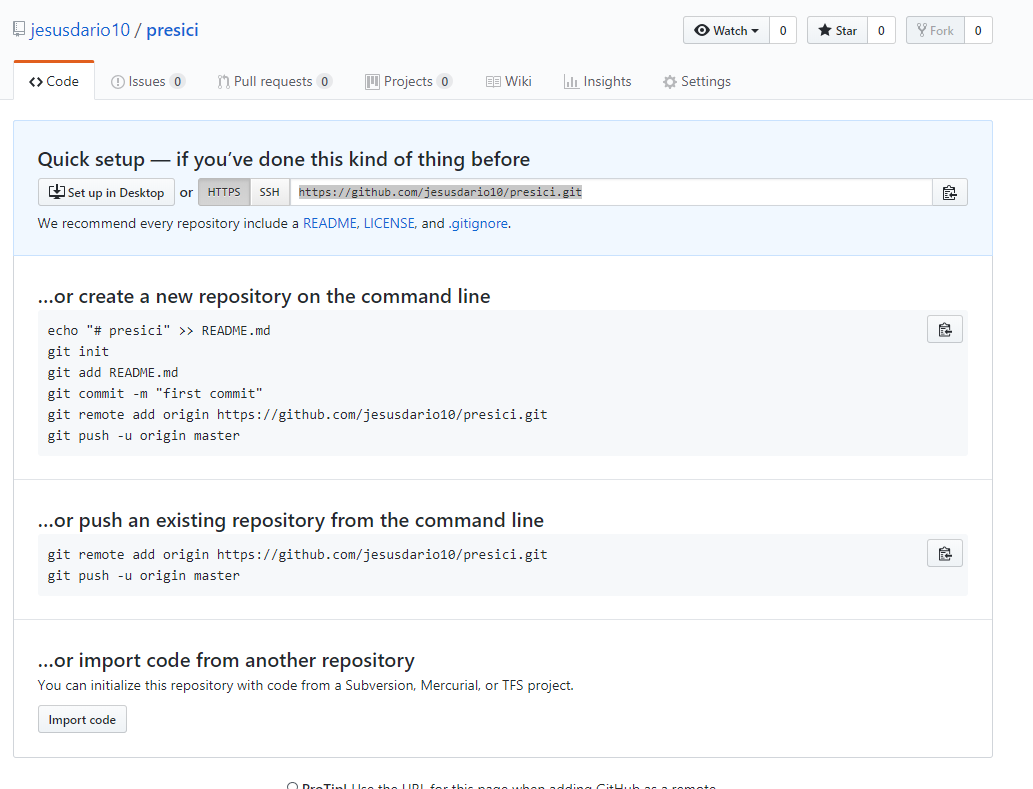
}

}

Lo mismo haremos pero con el register, pero el modulo de register deberá estar dentro de la carpeta login para usar los estilos del login.

**Respaldo del Proyecto en GitHub.**

Entramos a git creamos un nuevo proyecto y seguimos los pasos tal y como nos muestra la pagina, antes de hacer un commit recuerda añadir todos los archivos con git add –A



Después del comando git remote add origin <https://github.com/jesusdario10/presici.git> podemos ver el repositorio remoto con git remote –v

Por ultimo haremos como dice en la doc un **git push -u origin master**

Temas puntuales que aprenderemos en esta sección

Sección 4, Clase 28

Esta sección esta enfocada en módulos principalmente:

* Crear un módulo personalizado
* Crear rutas hijas
* Comenzaremos a crear módulos para agrupar tareas específicas
* Realizar cambios en GitHub
* Crear Release Tags que nos permitan descargar el código fácilmente, en caso de que necesitemos volver a comenzar donde nos quedemos.

El objetivo de esta unidad es simplificar el app.module.ts creando submodulos que soporten los componentes importados en este.

Es decir que en el @NgModule solo tengamos appComponent y que solo tengamos un solo archivo de rutas en los imports es decir solo APP\_ROUTES, los demás estarn enotros modulos.

**Entonces iniciemos:**

Creemos un modulo que nos administre todos los componentes que creamos y crearemos en los pages es decir en el app/pages.

Para eso en esa ruta crearemos el archivo **app/pages/pages.module.ts**, la estructura de un modulo se compone de la importación del NgModule y en @NgModule

Como **pages** consta de los componentes:

**Dashboard, gradicas1, progress y pageComponent** esos serán lo componentes que por ahora agregaremos a este modulo.

En el array de declarations definiremos los componentes anteriormente mencionados y quitemoslos de **app.modules.ts**.

import { NgModule } from '@angular/core';

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent

]

})

Cuando terminemos de definirlos importaremos sus componentes ira quedando asi

import { NgModule } from '@angular/core';

import { DashboardComponent } from './dashboard/dashboard.component';

import { ProgressComponent } from './progress/progress.component';

import { Graficas1Component } from './graficas1/graficas1.component';

import { PagesComponent } from './pages.component';

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent

]

})

Para exportarlas hacia otros componentes usamos la propiedad exports dentro del @NgModule. Y exportaremos el modulo completo normalmente

import { NgModule } from '@angular/core';

import { DashboardComponent } from './dashboard/dashboard.component';

import { ProgressComponent } from './progress/progress.component';

import { Graficas1Component } from './graficas1/graficas1.component';

import { PagesComponent } from './pages.component'

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent

],

exports:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent

]

})

export class PageModule{}

Ahora importaremos el modulo creado en el app.module.ts.

import {PageModule } from './pages/pages.module';

luego añadámoslos al array de imports ya que todos los modulos deben ir allí.

imports: [

BrowserModule,

APP\_ROUTES,

PageModule

],

También deberemos quitarlos del @NgModule en el apartado del array declaratios, ya que los estamos importando en el PageModule. ejecutemos

Si observamos en el navegador tendremos un error y es porque el principio de angular es que si yo tengo un modulo debo de importarlo en el archivo de app.modules.ts, como quite del Array **declarations** los modulos de DashboarComponent, ProgressComponent, PagesComponent, Graficas1Component me arrojara un error, aunque yo después haya importado en el app.module.ts el modulo de PagesModule.ts que es quien tienen estos componentes, pero porque se da el error en realidad, simplemente porque el componente PagesComponent requiere en su vista los componentes sidebar, breadcrums y header quienes no están importados en el modulo page.module.ts

Por lo tanto para que todo empiece a funcionar y sigamos resumiendo el app.module.ts, hagamos también lo mismo para los componentes que están en shared es decir breadbrumbs, header y sidebar y luego importemos ese modulo en el page.module.ts y en un array de imports en el declaratios coloquemoslo, deshabilitemos los componentes antes mencionados del app.module.ts y probemos debería funcionar pero aun no del todo.

**Shared.module.ts:**

import {NgModule } from '@angular/core';

import { SidebarComponent } from './sidebar/sidebar.component';

import { HeaderComponent } from './header/header.component';

import {BreadcrumbsComponent } from './breadcrumbs/breadcrumbs.component'

import { NopagefoundComponent } from './nopagefound/nopagefound.component';

@NgModule({

declarations:[

HeaderComponent,

SidebarComponent,

BreadcrumbsComponent,

NopagefoundComponent

],

exports:[

HeaderComponent,

SidebarComponent,

BreadcrumbsComponent,

NopagefoundComponent

]

})

export class SharedModule{ }

**pages.module.ts:**

import { NgModule } from '@angular/core';

import { DashboardComponent } from './dashboard/dashboard.component';

import { ProgressComponent } from './progress/progress.component';

import { Graficas1Component } from './graficas1/graficas1.component';

import { PagesComponent } from './pages.component';

import { SharedModule } from '../shared/shared.module';

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent

],

exports:[

DashboardComponent,

ProgressComponent,

Graficas1Component

],

imports:[

SharedModule

]

})

export class PageModule{}

**app.module.ts:**

@NgModule({

declarations: [

AppComponent,

LoginComponent,

*//NopagefoundComponent,*

*//DashboardComponent,*

*//ProgressComponent,*

*//Graficas1Component,*

*//PagesComponent,*

*//HeaderComponent,*

*//SidebarComponent,*

*//BreadcrumbsComponent,*

RegisterComponent

],

imports: [

BrowserModule,

APP\_ROUTES,

PageModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

y por ultimo en el pages.component.html deberemos comentar el router-outlet por el momento al ejecutar y mirar el navegador deberemos tener este resultado hasta ahora. Observaremos que el ruteo no se ejecuta por la falta del router-outlet pero si lo ponemos nos arrojara un error.

**Page.component.html**

*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

*<*app-sidebar*></*app-sidebar*>*

*<!-- ==================page-wrapper==================================== -->*

*<*div class="page-wrapper"*>*

*<!-- ============================================================== -->*

*<!-- Container fluid -->*

*<!-- ============================================================== -->*

*<*div class="container-fluid"*>*

*<!-- ============================================================== -->*

*<!-- Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<*app-breadcrumbs*></*app-breadcrumbs*>*

*<!-- ============================================================== -->*

*<!-- end Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<!-- ============================================================== -->*

*<!-- Start Page Content -->*

*<!-- ============================================================== -->*

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

***<!--<router-outlet></router-outlet>-->***

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<!-- ============================================================== -->*

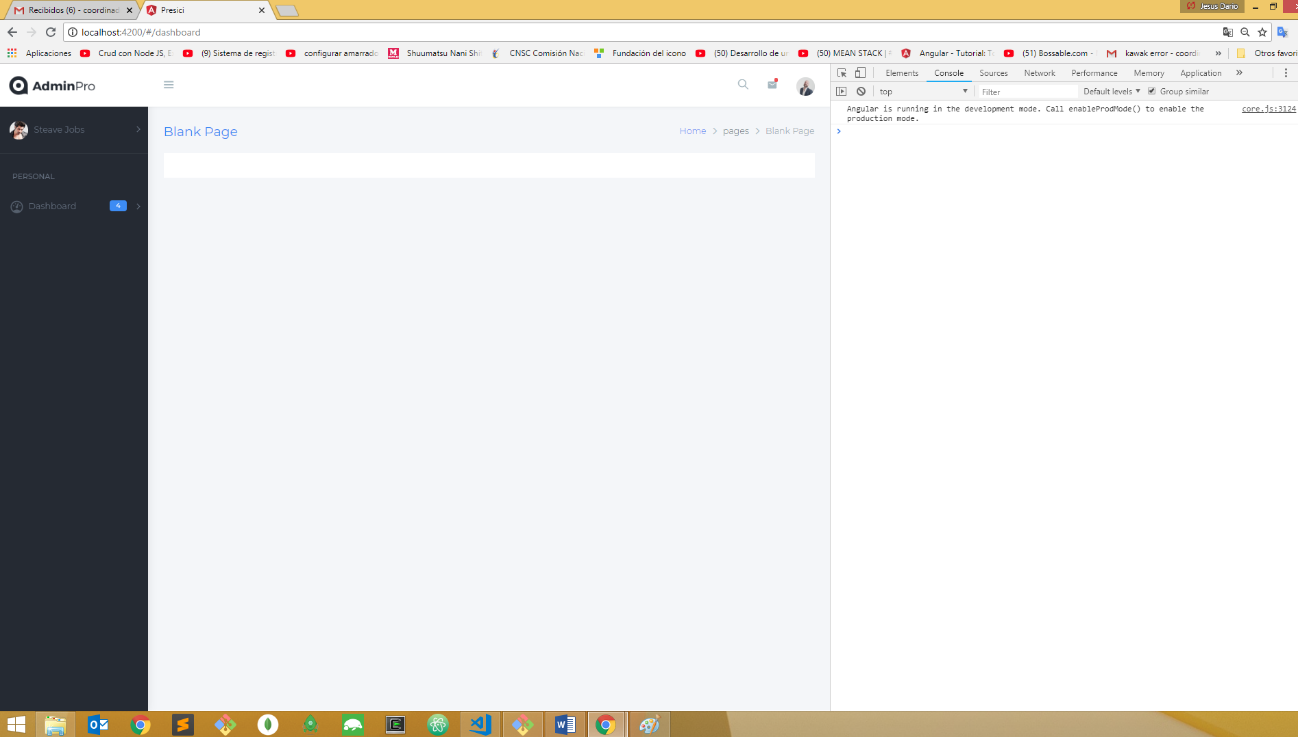
*<!-- End Page Content -->*

*<!-- ============================================================== -->*

*</*div*>*

*<!-- =================end-page-wrapper============================= -->*

*</*div*>*



**Rutas hijas:**

Arreglemos el problema de las rutas que se desactivaron, para mejor organización de las rutas crearemos rutas por carpeta contenedora en este caso crearemos un routing para **pages**.

1:En la carpeta **pages** creare un nuevo archivo llamado pages.routes.ts, este archivo es casi igual al archivo de rutas padre.

2: en pages.routes.ts cortaremos todas las rutas que le conciernen del app.routing.ts es decir de los componentes que le corresponden:

App.routing.ts quedara de la siguiente manera:

import { RouterModule, Routes } from '@angular/router';

import { DashboardComponent } from './pages/dashboard/dashboard.component';

import { LoginComponent } from './login/login.component';

import { ProgressComponent } from './pages/progress/progress.component';

import { Graficas1Component } from './pages/graficas1/graficas1.component';

import { NopagefoundComponent } from './shared/nopagefound/nopagefound.component';

import { PagesComponent } from './pages/pages.component';

import { RegisterComponent } from './login/register.component';

const appRoutes : Routes = [

{path: 'login', component: LoginComponent},

{path: 'register', component: RegisterComponent},

{path: '\*\*', component: NopagefoundComponent}

]

export var APP\_ROUTES = RouterModule.forRoot(appRoutes, { useHash: true });

el pages.routes.ts quedara de la siguiente manera, como son rutas hijas en la exportación al usar el RouterModule en ves de forRoot usaremos forChild.

import { RouterModule, Routes } from '@angular/router';

import { PagesComponent } from './pages.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { ProgressComponent } from './progress/progress.component';

import { Graficas1Component } from './graficas1/graficas1.component';

const pageRoutes : Routes = [

{

path: '',

component: PagesComponent,

children:[

{path: 'dashboard', component: DashboardComponent},

{path: 'progress', component: ProgressComponent},

{path: 'graficas1', component: Graficas1Component},

{path: '', redirectTo: '/dashboard', pathMatch:'full'}

]

}

];

export const PAGES\_ROUTES = RouterModule.forChild(pageRoutes)

ahora debemos importar el PAGES\_ROUTES en el modulo de pages es decir en pages.module.ts ya que harán parte del pages.module.ts y las rutas deben ser añadidas al array de imports.

Importamos

import { PAGES\_ROUTES } from './pages.routes';

añadimos a los imports

imports:[

SharedModule,

PAGES\_ROUTES

]

Ahora volveré a activar el router-outlet. Del pages.component.html.

*<*div id="main-wrapper"*>*

*<*app-header*></*app-header*>*

*<*app-sidebar*></*app-sidebar*>*

*<!-- ==================page-wrapper==================================== -->*

*<*div class="page-wrapper"*>*

*<!-- ============================================================== -->*

*<!-- Container fluid -->*

*<!-- ============================================================== -->*

*<*div class="container-fluid"*>*

*<!-- ============================================================== -->*

*<!-- Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<*app-breadcrumbs*></*app-breadcrumbs*>*

*<!-- ============================================================== -->*

*<!-- end Bread crumb and right sidebar toggle -->*

*<!-- ============================================================== -->*

*<!-- ============================================================== -->*

*<!-- Start Page Content -->*

*<!-- ============================================================== -->*

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*router-outlet*></*router-outlet*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<!-- ============================================================== -->*

*<!-- End PAge Content -->*

*<!-- ============================================================== -->*

*</*div*>*

*<!-- =================end-page-wrapper============================= -->*

*</*div*>*

**Guardemos todo nuevamente en git con los siguientes comandos.**

En este momento he alcanzado gracias a Dios la versión 1 para producción.

1:revisamos los cambios con **git status**.

2:agregamos esos cambios con **git add .** para incluirlos en el stagin area

3:creamos el commit con **git commit –m “modificaciones en los modulos”**

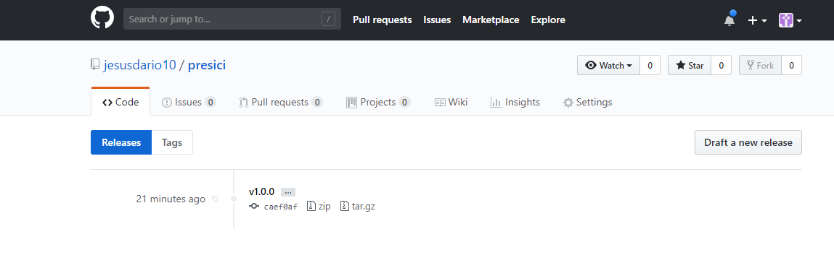
4:hagamos un **git push** para subirlo al repositorio.

5:Creemos un tag de git para algunas anotaciones como versiónes como a continuación.

**git tag –a v1.0.0 –m “Versión 1 – lista para produccion”**

6:para ver el tag usamos **git tag**.

7:para subirlo a github : **git push - -tags**

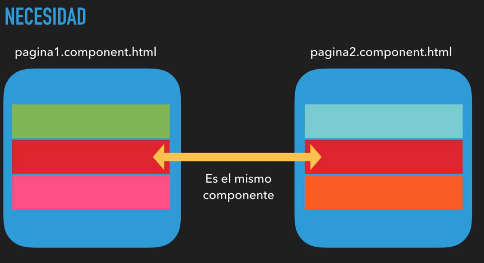


**Trabajaremos con Outputs, Inputs y ViewChild**

* Aprenderemos como utilizar atributos personalizados
* Crearemos un componente re utilizable con una funcionalidad en especifico
* Aprenderemos a tener referencias a elementos HTML
* Tips de JavaScript puro: colocar el foco en elementos
* Uso de gráficas como componentes personalizados

**Para que puede servir la comunicación entre componentes?**

Supongamos que tenemos 2 componentes, ambos hacen y trabajan de formas diferentes, pero ambos tienen una sección interna que funciona para ambos lados, lo ideal es que compartieran el mismo componente, es decir lo ideal es que todos nuestros componentes los podamso reutilizar en cualquier otro componente.



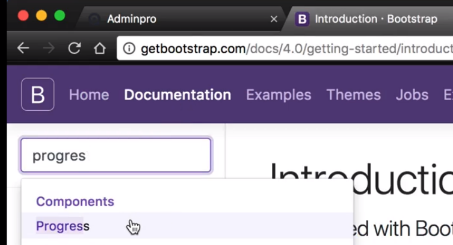
Entonces el componente de color **rojo** es el componente en común, tiene un comportamiento especifico que puede servirle a diferentes componentes padres o simplemente componentes que locontengan, este componente a su ves podrá recibir información del padre o no podrá hacerlo, puede emitir información hacia el padre o simplemente es algo solo para mostrar información, en angular existe algo que nos permitirá que los componentes tengan este comportamiento.

**@Input()=** es cuando un componente recibe información del componente padre.

**@Outpu()t**= es cuando un componente hijo envía información hacia su padre.

Trabajemos con el componente que creamos llamado **Progress**, aquí manipularemos los progressbar que tiene el bootstrap.

Para eso iremos al <https://getbootstrap.com/> documentación> y el el buscador coloquemos progress



Usemos el ultimo que tiene animaciones



Copiamos el código, vamnos al progress.component.html y lo pegamos allí.

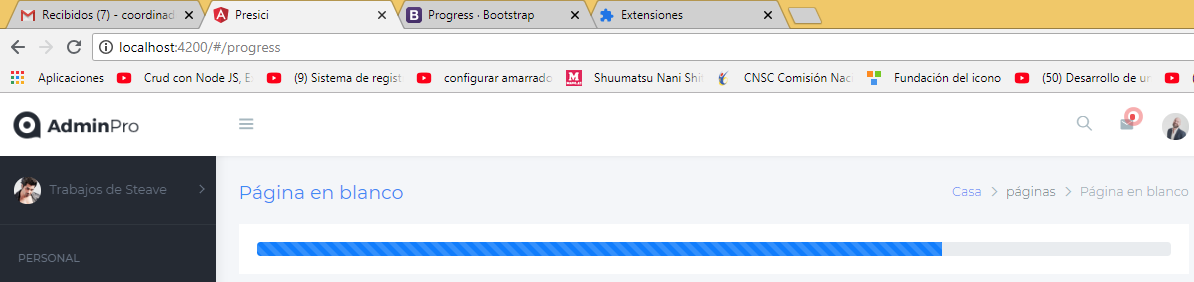
*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100" style="width: 75%"*></*div*>*

*</*div*>*

Si probamos en la ruta del progress podremos observar el cambio.

<http://localhost:4200/#/progress>



Copiemos la tarjeta estándar del page.component.html para agregar allí dentro el código del progress y que se vea mejor

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*router-outlet*></*router-outlet*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Y donde esta el router-outlet coloquemos el progress bar que trajimos de bootstrap. Es decir el progress.component.html quedaría asi: y agreguemso un titulo

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Card Title*</*h4*>*

*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100" style="width: 75%"*></*div*>*

*</*div*>*

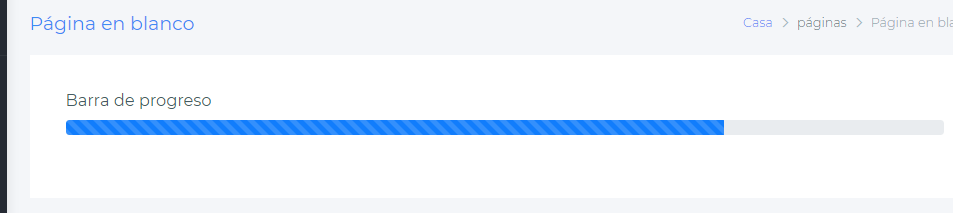
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

a



Añadamos un par de tarjetas mas

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

Texto 1

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

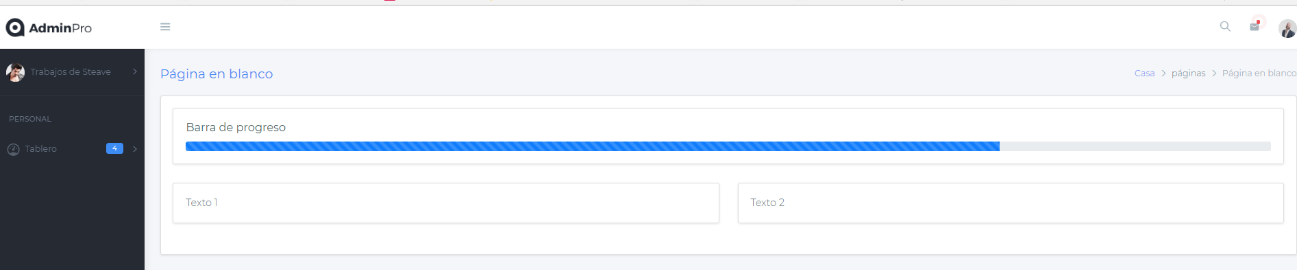
Texto 2

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*



Ahora coloquemos botones de mas y de menos para que la barra cresca y decrezca, ahora vamos a bootstrap y busquemos input group.

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div class="input-group"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control"*>*Hate id*</*button*>*

*</*span*>*

*<*input type="text" class="form-control" placeholder="Product name"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control"*>*Hate id*</*button*>*

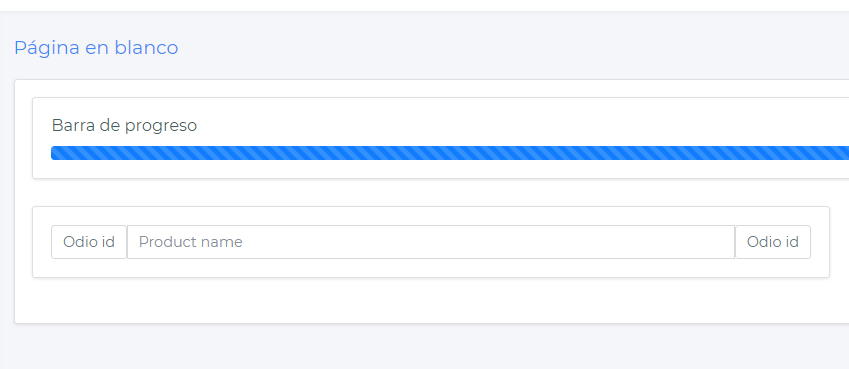
*</*span*>*

*</*div*>*

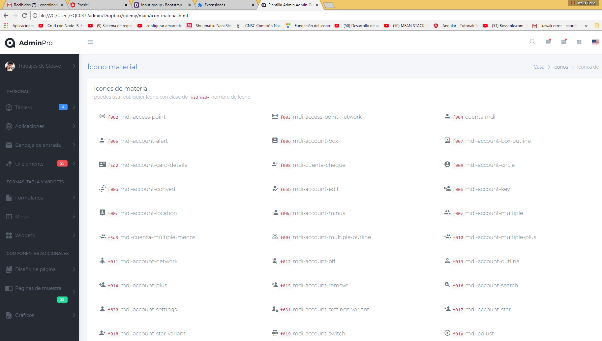
*</*div*>*

*</*div*>*

*</*div*>*



Pongamos iconos. Busquemos en el material descargado del template. Ya que viene pregargado con fontawesome etc. Entonces busquemos en template la carpeta main y el html que tiene los iconos de material que se llama icon-material.html, la abrimos y podemos observar que tenemos un monto de iconos.



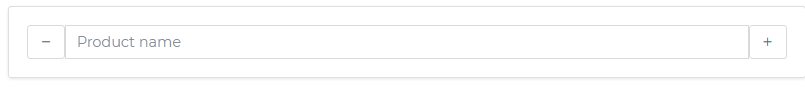
Hay un monton entonces escojamos el que se llama mdi-plus, y lo usaremos añadiendo una etiqueta <i> y añadimos la clase mdi seguida del nombre del icono.

*<*i class="mdi mdi-plus"*></*i*>*

Ahora coloquemos el menos

*<*i class="mdi mdi-minus"*></*i*>*

Se vera asi



**Uso de atributos personalizados con angular.**

Ya tenemos la barra ahora hagamos que funcione, el botón + ingrementara en 5% la barra de progreso y el – decrementara en 5%, validemos que no se pase del 100% y que no se pueda poner menos del cero.

Abramos el componente progress que es donde vamos a poner la lógica.

1-declararemos una propiedad llamada progreso de tipo number igual a 50

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-progress',

templateUrl: './progress.component.html',

styles: []

})

export class ProgressComponent implements OnInit {

public progreso: number = 50;

constructor() {

ngOnInit() {

}

}

Ahora vamos a la vista enlacemos el **atrinbuto aria-valuenow para que angular lo reconozca**? Con el atributo attr y las los corchetes podre enlazarlo como asi.

**Esto es lo que esta antes de enlazarlo**

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100" style="width: 75%"*></*div*>*

Esto es lo que esta al enlazarlo

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="50" [attr.aria-valuenow]="progreso" aria-valuemin="0" aria-valuemax="100" style="width: 50%"*></*div*>*

Lo mismo podriamos hacer con los demas atributos propios de bootstrap, ahora también hay que cambiar el style también lo manejarsmos con las [] y concatenaremos la variable progreso de la siguiente manera.

[style.width]="progreso+'%'"

El % va porque es lo que me pide la etiqueta style originalmente de modo que quedara asi. Terminaremos removiendo el style original

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="50" [attr.aria-valuenow]="progreso" aria-valuemin="0" aria-valuemax="100" [style.width]="progreso+'%'"*></*div*>*

Ahora programemos los botones el + incrementa en 5 y el menos lo decrementa

Agreguemos los eventos a los botones iniciemos con el cambiarValor(-5) le pasamos- 5 para que decremente en 5 este seria el -

*<*button class="form-control" (click)="cambiarValor(-5)"*>*

Y el otro incrementara

*<*button class="form-control" (click)="cambiarValor(+5)"*>*

Esta function la definiremos en el component progressComponent.ts

cambiarValor(valor){

this.progreso = this.progreso + valor;

}

De ese modo incrementara o decrementara dependiendo del botón que cliquee, ahora valodemos que no permita que sea menor que 0 y meyor a 100.

cambiarValor(valor){

if(this.progreso >= 100){

alert("llegaste al 100%");

return;

}

if(this.progreso <=0){

alert("llegaste al 0%");

return;

}

this.progreso = this.progreso + valor;

}

Ahora hagamos que el valor introducido en el input sea el progreso para esto usaremos en ngModel, para que funcione hay que ponerle un name al input o al campo en cuestión. E importar en el modulo el formsModule, en este caso el modulo que contiene progress es el page.module.ts.

import { FormsModule } from '@angular/forms'

y en el array de imports

imports:[

SharedModule,

PAGES\_ROUTES,

FormsModule

]

Ahora si afectemos el input como deberíamos coloquémosle un name y el [(ngModel)]

*<*input type="text" class="form-control" name="progreso" [(ngModel)]="progreso" placeholder="Product name"*>*

Ahora validemos mejor el tipo de dato.

cambiarValor(valor : number){

if(this.progreso >= 100 && valor > 0){

alert("llegaste al 100%");

this.progrso = 100;

return;

}

if(this.progreso <=0 && valor < 0){

alert("llegaste al 0%");

this.progreso = 0;

return;

}

this.progreso = this.progreso + valor;

}

Ahora haremos de este un componente reutilizable ya que si se da la necesidad de crear otro progressbar, entonces vamos a la vista y agreguemos ese otro progressbar démosle un color verde con la clase bg-success de la siguiente manera.

*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar"

aria-valuenow="50" [attr.aria-valuenow]="progreso" aria-valuemin="0" aria-valuemax="100" [style.width]="progreso +'%'"*></*div*>*

*</*div*>*

*<*hr*>*

*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated bg-success" role="progressbar"

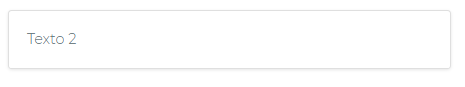
aria-valuenow="50" [attr.aria-valuenow]="progreso" aria-valuemin="0" aria-valuemax="100" [style.width]="progreso +'%'"*></*div*>*

*</*div*>*

El problema es que si yo quisiera en tomar toda la lógica de esta caja



Y ponerla en esta otra caja para independizar los progressbar, tendría que crear nuevas variables, un nuevo metodo en fin tendría que hacerlo por separado.



Aquí es donde podríamos pensar en crear un componente personalizado que nos permita controlar este elemento y que reciba los parámetros y que saque parámetros para poder usarlos en cualquier lado. HAGAMOS ESO.

Creemos en el app una nueva carpeta que me permita trabajar con todos mis componetnes personalizados esa carpeta se llamara app/components y el primer componente se llamara incrementador lo queremos sin el archivo spec y sin los estilos. Para esto podemos usar el comando **ng g c components/incrementador –is - -spec=false**.

Toda la lógica y le html que yo quiero reutilizar realimente es el input group es decir:

*<*div class="input-group"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control" (click)="cambiarValor(-5)"*>*

*<*i class="mdi mdi-minus"*></*i*>*

*</*button*>*

*</*span*>*

*<*input type="text" class="form-control" name="progreso" [(ngModel)]="progreso" placeholder="Product name"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control" (click)="cambiarValor(5)"*>*

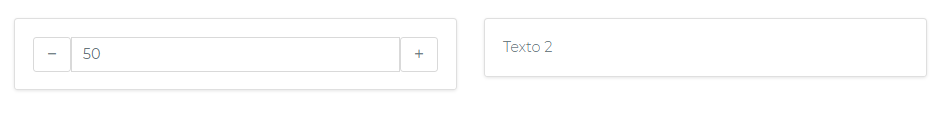
*<*i class="mdi mdi-plus"*></*i*>*

*</*button*>*

*</*span*>*

*</*div*>*

Que en render es



Nos vamos al html del componente incrementador y lo pegamos alla, ahora iremos al incrementador.component.ts y crearemos allí la función de cambiarValor y la propiedad progreso.

export class IncrementadorComponent implements OnInit {

public progreso: string="50";

public val :number;

constructor() { }

ngOnInit() {

}

cambiarValor(valor : number){

console.log(this.progreso);

this.val = parseFloat(this.progreso);

console.log(this.val);

console.log(typeof(this.val));

if(this.val >= 100 && valor > 0){

alert("llegaste al 100%");

this.val = 100;

return;

}

if(this.val <=0 && valor < 0){

alert("llegaste al 0%");

this.val = 0;

return;

}

this.val +=valor;

this.progreso = String(this.val);

}

Al hacer esto la podemos borrar de **progress.component.ts,** quedando de la siguiente manera este componente

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-progress',

templateUrl: './progress.component.html',

styles: []

})

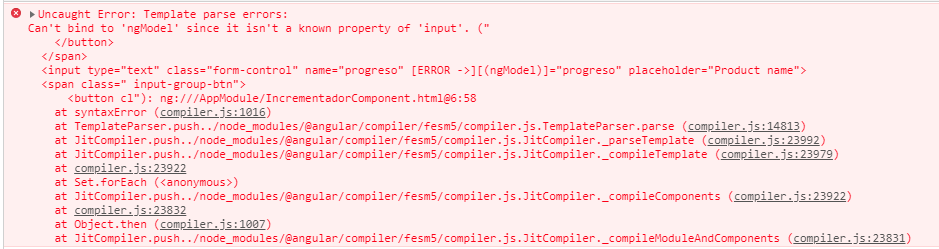
export class ProgressComponent implements OnInit {

ngOnInit() {

}

}

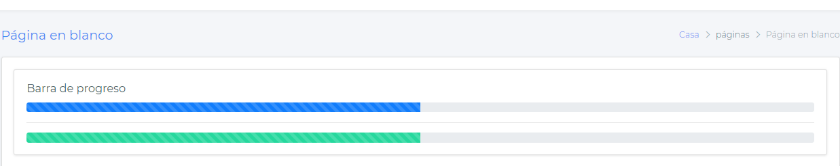
En cuando baje y suba el servicio me aparecerá este error



Dice que no puede encontrar el ngModel y esto es comprensible porque el componente incrementador usa en su vista el [(ngModel)]=”progreso” y en ninguna parte recibe el modulo de

import { FormsModule } from '@angular/forms'

que es el que active esta funcionalidad, según la lógica este componente es decir incrementador, esta en el modulo principal es decir deberíamos de importar el FormsModule en el app.module.ts y agregarlo al array de imports. Al hacer esto el error desaparece como observamos a continuacion



Ahora que haremos con el incrementador, primero que todo debere inportarlo debajo del progressbar porque ya no existe allí es decir que aparezcan debajo de las barras de progreso los botones de + y -, de entrada recibire 2 parametros, el primero es el valor inicial del progress y una leyenda para saber que va a actualizar.

Por lo tanto en el componente incrementador creare dos propiedades:

La primera publica llamada leyenda de tipo string con valor leyenda.

public leyenda : string = "Leyenda";

esa leyenda la colocare como titulo de los botones de la vista del componente incrementador.

*<*h3*>*{{leyenda}}*</*h3*>*

*<*div class="input-group"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control" (click)="cambiarValor(-5)"*>*

*<*i class="mdi mdi-minus"*></*i*>*

*</*button*>*

*</*span*>*

*<*input type="text" class="form-control" name="progreso" [(ngModel)]="progreso" placeholder="Product name"*>*

*<*span class=" input-group-btn"*>*

*<*button class="form-control" (click)="cambiarValor(5)"*>*

*<*i class="mdi mdi-plus"*></*i*>*

*</*button*>*

*</*span*>*

*</*div*>*

Ahora importemos el componente incrementador en el componente progreso para poderlo ver, como hacermos eso, simple, como son componentes que dependen de modulos distintos en este caso el componente incrementador depende del app.module.ts y el componente progress.component.ts depende del modulo de pages.module.ts, y como quiero utilizar el incrementador en el progress, por lógica debo importar el incrementador en el page.module.ts para poder usarlo en el progress

Entonces: el page.module.ts quedaría asi:

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms'

import { DashboardComponent } from './dashboard/dashboard.component';

import { ProgressComponent } from './progress/progress.component';

import { Graficas1Component } from './graficas1/graficas1.component';

import { PagesComponent } from './pages.component';

import { SharedModule } from '../shared/shared.module';

import { PAGES\_ROUTES } from './pages.routes';

import { IncrementadorComponent } from '../components/incrementador/incrementador.component';

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent,

IncrementadorComponent

],

exports:[

DashboardComponent,

ProgressComponent,

Graficas1Component

],

imports:[

SharedModule,

PAGES\_ROUTES,

FormsModule

]

})

export class PageModule{}

luego podremos usar ya el selector del componente importado es decir incrementador , <app-incrementador>, en el componente progress en este caso en la vista como se muestra a continuacion.

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador*></*app-incrementador*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador*></*app-incrementador*>*

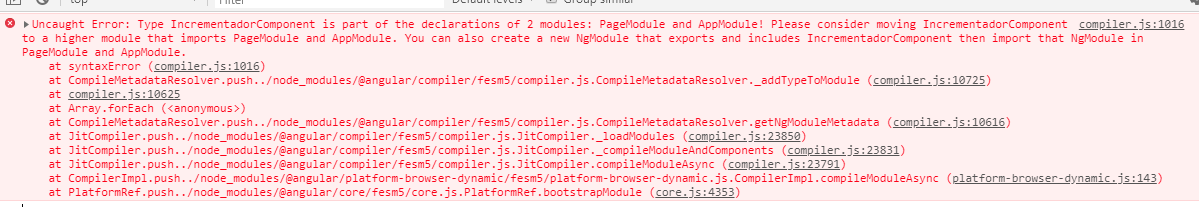
*</*div*>*

*</*div*>*

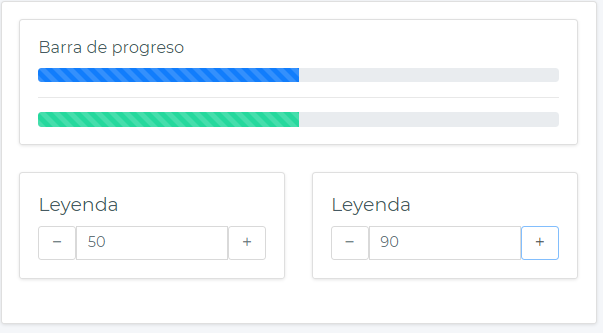
*</*div*>*

*</*div*>*

Al grabar esto me va a arrojar errores, y es porque el componente incrementador esta importado tanto en el app.module.ts(cuando lo creamos automáticamente se agrego alli) como en el pages.module.ts(cuando lo importamos para poder usarlo)



Para corregir este error de momento deberemos de quitarlo del app.module.ts tanto la importación como quitarlo del @ngModule. Cuando hagamos eso ya podremos ver el resultado.



Ambas cards de botones que dentro tienen el componente incrementador funcionan de menera independiente podemos observar que una tiene 50 y la otra 90, pero los progressbar o las barras de progreso no se mueven. Yo necesito ahora que del componente padre que en este caso es el **progress.component** le envie 2 parametros al componente increment que usa una leyenda y un numero que seria el valor inicial del progress,

Es decir vayamos al componente incrementador, podemos observar que se declaran 2 variables(atributos) una llamada leyenda y otra llamada progreso. Esas 2 variables yo podría recibirlas desde otro componente utilizando el decorador **@Input()** . como observamos a continuación:

export class IncrementadorComponent implements OnInit {

@Input() progreso: number=50;

@Input() leyenda : string = "Leyenda";

Al hacer esto debere por obligcion importar el modulo de input es decir agregarlo a lo que ya tenia de la siguiente manera.

import { Component, OnInit, **Input** } from '@angular/core';

ahora como hacemos que termine de funcionar, cuando yo hago lo anterior inmediatamente puedo en usar en el html de mi componente padre en este caso progress.component.html, atributos con los nombres de las variables a las que se les hizo el @input por ejemplo.

Vamos al progress.component.html y podre usar lo siguiente.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador **leyenda ="barra Azul**"*></*app-incrementador*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador **leyenda="barra Verde**"*></*app-incrementador*>*

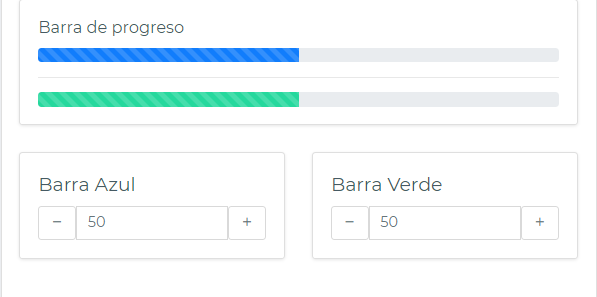
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

De ese modo estos valores que yo coloque aquí serán los que se pasen al componente hijo en este caso increment.component.ts osea que el atributo leyenda declarado en la etiqueta -----<app-increment> del componente padre progress, le dara su valor a la variable @Input leyenda : string del componente hijo increment. Al guardar podremos observar los resultados que dice en uno barra azul y el otro barra verde.



Una anotación si yo a un atributo html de angular lo declaro encerrado en [] quera decir que esa etiqueta la podre relacionar con una variable declarada en mi componente que tenga el mismo nombre de aquí lo que en adelante será muy importante porque la siguiente etiqueta será el progreso del componente padre lo declararemos dentro de [] de la siguiente manera.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador leyenda ="barra Azul" **[progreso]="progreso"*></***app-incrementador*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-incrementador leyenda="barra Verde" **[progreso]="progreso"*></***app-incrementador*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

De este modo esa etiqueta del componente **progress.component.html** obtendrá su valor de la variable o atributo declarado en el componente progress.component.ts

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-progress',

templateUrl: './progress.component.html',

styles: []

})

export class ProgressComponent implements OnInit {

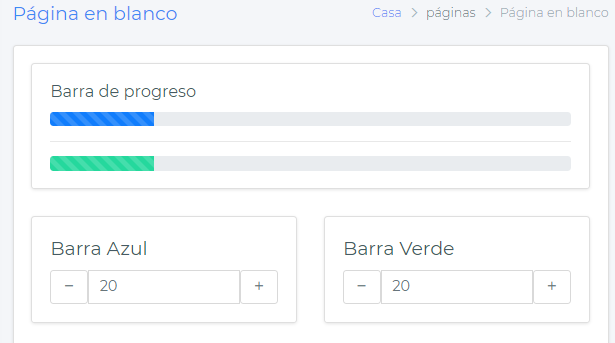
public progreso : number = 20;

ngOnInit() {

}

}

Vamos bien ya podemos observar que si yo cambio a 20 el valor de progreso en el render también cambiara aunque aun las barras no tendrán funcionalidad aun.



Un truco adicional si yo quiero darle un aliar a una variable que desee importar por ejemplo quiero que leyenda del componente incrementador sea rconocida en el componente progress como leye lo que hace será: en el componente hijo pasarle el parámetro al input con el alias que quiero que se relacione ejemplo:

@Input('**leye**') leyenda : string = "Leyenda";

Y en el componente padre es decir progress en la vista html debere usar el atributo leye para poder enlazar.

*<*app-incrementador **leye** ="barra Azul" [progreso]="progreso"*></*app-incrementador*>*

Es un truco para renombrar atributos personalizados y que no se repitan sus nombres.

Ya enviamos datos desde el componente padre progress hacia el componente hijo incrementador, ahora como hacemos para enviar datos desde el hijo hacia el padre? Es decir para que al yo utilizar los botones del componente hijo increment pueda con esos datos enviárselos al padre y mover los progressbar del padre. Eso se hace de la siguiente manera.

Lo primero será independizar el atributo progreso del componente padre es decir crearemos uno mas quedando de la siguiente manera.

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-progress',

templateUrl: './progress.component.html',

styles: []

})

export class ProgressComponent implements OnInit {

public progreso1 : number = 20;

public progreso2 : number = 30;

ngOnInit() {

}

}

Luego arreglaremos los nombres definidos en el html es decir desde el style de las barras hasta los botones.

*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar"

aria-valuenow="50" [attr.aria-valuenow]="progreso1" aria-valuemin="0" aria-valuemax="100" [style.width]="progreso1 +'%'"*></*div*>*

*</*div*>*

*<*div class="progress"*>*

*<*div class="progress-bar progress-bar-striped progress-bar-animated bg-success" role="progressbar"

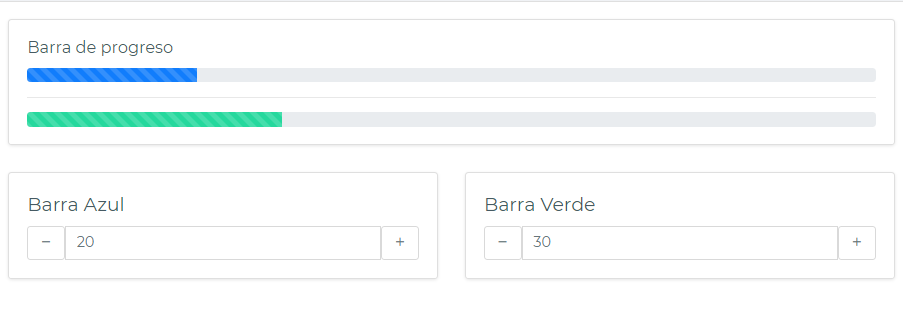
aria-valuenow="50" [attr.aria-valuenow]="progreso2" aria-valuemin="0" aria-valuemax="100" [style.width]="progreso2 +'%'"*></*div*>*

*</*div*>*

*<*app-incrementador leye ="barra Azul" [progreso]="progreso1"*></*app-incrementador*>*

*<*app-incrementador leye="barra Verde" [progreso]="progreso2"*></*app-incrementador*>*

La vista quería por el momento asi.



Ahora vayamos al componente incrementador. Aquí notificaremos al padre que en el hijo ocurrirá un evento que cambiara el atributo de progreso en las barras graficas.

Para eso usaremos el decorador @Output(), deberemos importarlo

import { Component, OnInit, Input, Output } from '@angular/core';

le pondremos un nombre a un evento que le pondremos en este caso **cambioValor** y será de tipo **EventEmitter**  y su tipo de dato sera, **EventEmitter**  también tiene su modulo asi que hay que importarlo.

import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

y le debemos añadir el tipo de dato que ese evento emitirá en este caso será un numero. Y eso lo igualaremos a una instancia del modulo EventEmiter, la siguiente será la sintaxis.

export class IncrementadorComponent implements OnInit {

@Input() progreso: number;

@Input('leye') leyenda : string = "Leyenda";

@Output() cambioValor:EventEmitter<number> = new EventEmitter();

Y en que momento lo emitiremos? Cuando ya tengamos el resultado del numero en este caso cuanto tengamos el valor numérico de progreso osea:

this.progreso = this.progreso + valor;

this.cambioValor.emit(this.progreso)

ahora como activamos esto en el component padre progress? Bueno escuchamos el evento es decir: vamos al html del componente progress y declaramos un atributo en el <app-incrementador> que se llamara igual al alias o al nombre que declaramos en el @Output del hijo osea en este caso **cambioValor**. luego Como se muestra a continuación pero entre () como si fuera un evento click.

*<*app-incrementador leye ="barra Azul" (cambioValor)=""

Y lo igualaremos al nombre de la propiedad declarada en el componente padre en este caso progreso1 y progreso2 y también lo igualaremos a un evento osea $event, ya que es lo que envía. Esta es la sintaxis para usar los @Output.

(cambioValor)="progreso2 = $event"

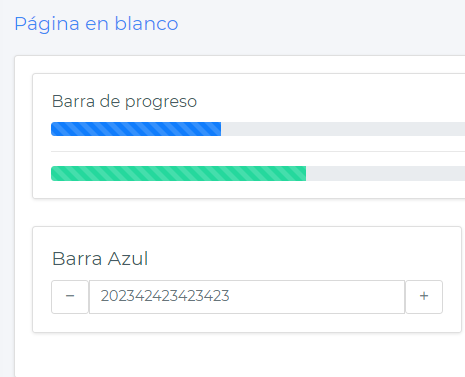
Completa seria:

*<*app-incrementador leye ="barra Azul" (cambioValor)="progreso1 = $event" [progreso]="progreso1"*></*app-incrementador*>*

Ahora validemos que los input solo reciban números, vamos al incrementador.component.html y en el campo input vamos a cambiarlo de text a number, y al final del input le agregaremos la etiqueta min que sea igual a 1 para que el valor minimo que permita sea 1 y la etiqueta max en 10 para que el valor máximo que reciba sea 100.

*<*input type="number" class="form-control" name="progreso" [(ngModel)]="progreso" placeholder="Product name" min="0" max="100"*>*

Esto me permitirá solo recibir números pero si me doy cuenta la caja me sigue aceptando todos los números que yo le coloque.



Lo primero que debemos hacer es darnos cuenta cuando los valores cambian en el input, para eso usamos un evento llamado ngModelChange y este disparara un metodo al que llamaremos onChanges que recibirá el evento qu al usar el NgModel=progreso será un numero.

*<*input type="number" class="form-control" **(ngModelChange)="onChange($event)"** name="progreso" [(ngModel)]="progreso" placeholder="Product name" min="0" max="100"*>*

Ahora definamos ese metodo en el component incrementador.component.ts, como recibe un evento numero se declarara de la siguiente forma.

onChanges(newValue:number){

}

Y para que esto afecte las barras deberé pasarle el cambiarValor con e emit.

onChanges(newValue:number){

console.log(newValue);

this.cambioValor.emit(this.progreso);

}

Ahora validemos que n se pase d los rangos 0 y 200

Lo primero será setear los campos con la codiciones

onChanges(newValue:number){

console.log(newValue);

if(newValue >=100){

this.progreso = 100;

}else if(newValue <=0){

this.progreso = 0;

}else{

this.progreso = newValue;

}

this.cambioValor.emit(this.progreso);

}

Lo segundo era por medio de javasript no permitir asignar el valor de progreso al input y esto lo haremos con javascript puro. En dos pasos.

1. Encontrar el elemento html en este caso el input por el nombre aunque lo puedo hacer por el id u otras cosas.

onChanges(newValue:number){

let elementHTML : any = document.getElementsByName('progreso')[0];

1. Asignar a ese elemento el valor de this.progreso de esta forma por mas que quiera ese campo jamas me permitira otro valor.

onChanges(newValue:number){

let elementHTML : any = document.getElementsByName('progreso');

if(newValue >=100){

this.progreso = 100;

}else if(newValue <=0){

this.progreso = 0;

}else{

this.progreso = newValue;

}

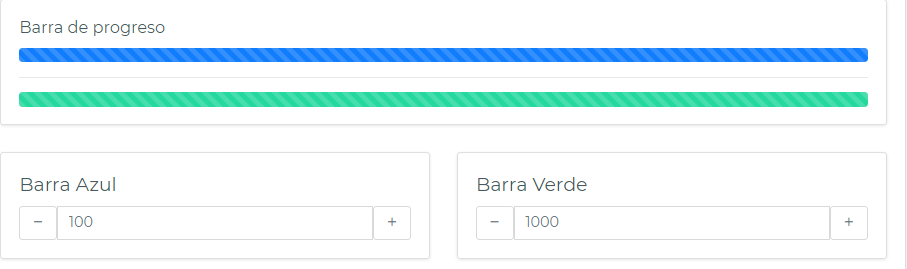
**elementHTML.value = this.progreso;**

this.cambioValor.emit(this.progreso);

}

Probemos, en el primer input tratare de colocar 1000 y no me dejara pero en el segundo input si , y esto como lo soluciono? Con el decorador @ViewChild ojo hay que inportarlo

import { Component, OnInit, Input, Output, EventEmitter, ViewChild } from '@angular/core';



este decorador recibe como parámetro una referencia a un elemento html y le dare también un nombre del lado de typescript dándole también un tipo, en este caso ElementRef(debería ser importado) osea referencia de elemento.

import { Component, OnInit, Input, Output, EventEmitter, ViewChild, ElementRef } from '@angular/core';

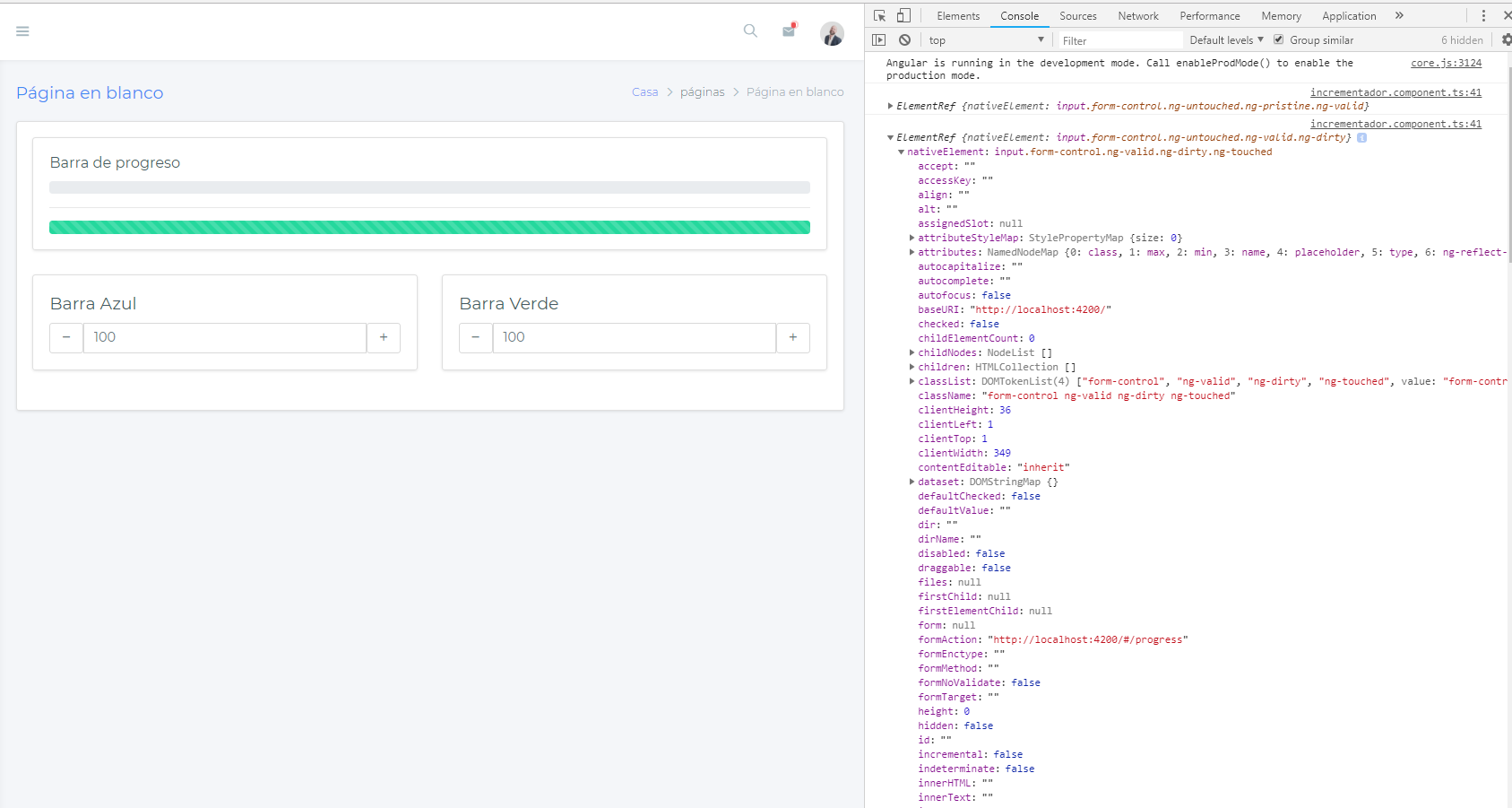
export class IncrementadorComponent implements OnInit {

@ViewChild('txtProgress') txtProgress : ElementRef;

y como hago esa referencia al elemento? Pues simple le doy un nombre en angular y eso lo hago con el #nombrequequieroponerle tal como se muestra a continuación.

*<*input type="number" **#txtProgress** class="form-control" (ngModelChange)="onChanges($event)" name="progreso" [(ngModel)]="progreso" placeholder="Product name" min="0" max="100"*>*

Si yo hago un clg a txtProgress en el metodo onchanges me dare cuenta que tengo un objeto ElementRef y dentro de el tengo un nativeElement donde tendre todo el input completo



Ahora para que funcione ya no hare la referencia con javascript puro es decir que

let elementHTML : any = document.getElementsByName('progreso')[0];

y

elementHTML.value = this.progreso;

los borraremos para referenciar con el Nuevo nombre angular de la siguiente manera:

this.txtProgress.nativeElement.value = this.progreso;

el metodo en general quedaria de la siguiente forma.

onChanges(newValue:number){

*//let elementHTML : any = document.getElementsByName('progreso')[0];*

*//console.log(this.txtProgress);*

if(newValue >=100){

this.progreso = 100;

}else if(newValue <=0){

this.progreso = 0;

}else{

this.progreso = newValue;

}

*//elementHTML.value = this.progreso;*

this.txtProgress.nativeElement.value = this.progreso;

this.cambioValor.emit(this.progreso);

}

Con esto todo esta arreglado y referenciado.



ng2-charts

implementación de gráficos de donas. <https://valor-software.com/ng2-charts/>

Lo primero es instalarlo:

npm install ng2-charts --save

luego se necesitara incuir un charts.js de la biblioteca asi que también la instalaremos

npm install chart.js --save

como vamos a trabajar en las graficas de torta usaremos el componente de graficas1 pero recuerda que este componente esta en el modulo de pages asi que la siguiente importación para que los ng2-chats funcionen la deberemos hacer en el modulo de pages.

### **Uso**

import { ChartsModule } from 'ng2-charts';

// In your App's module:

imports: [

ChartsModule

]

Osea en pages.module.ts inportare respectivamente como lo dice la documentación.

*//ng2-charts*

import { ChartsModule } from 'ng2-charts';

imports:[

SharedModule,

PAGES\_ROUTES,

FormsModule,

**ChartsModule**

]

En la pagina podremos observar la documentación de como funcionan estas graficas pero trabajaremos con las donas

#### **Doughnut Chart**

Encontraremos el html necesario

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

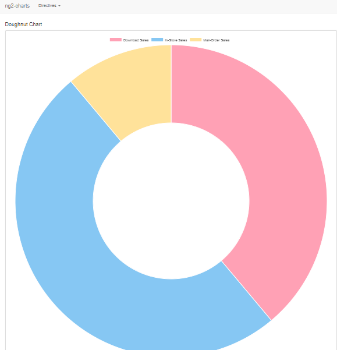
[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"

(chartHover)="chartHovered($event)"

(chartClick)="chartClicked($event)"*></*canvas*>*

*</*div*>*



Y el typescript también

import { Component } from '@angular/core';

@Component({

selector: 'doughnut-chart-demo',

templateUrl: './doughnut-chart-demo.html'

})

export class DoughnutChartDemoComponent {

*// Doughnut*

public doughnutChartLabels:string[] = ['Download Sales', 'In-Store Sales', 'Mail-Order Sales'];

public doughnutChartData:number[] = [350, 450, 100];

public doughnutChartType:string = 'doughnut';

*// events*

public chartClicked(e:any):void {

console.log(e);

}

public chartHovered(e:any):void {

console.log(e);

}

}

Vamos al pages.component.html copiamos el marco que es

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*Aqui ira la grafica*

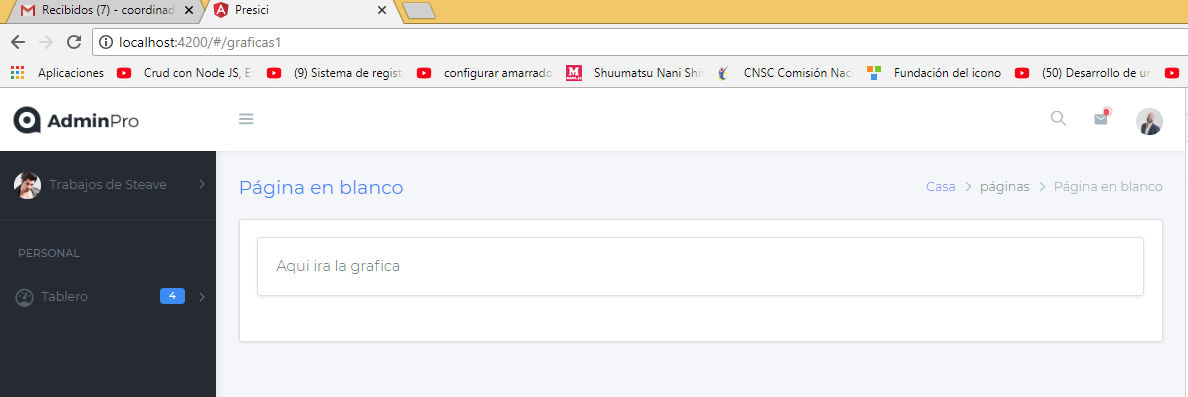
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Y lo pegamos en graficas1.component.html



Ahora tomaremos el codio html de la documentación es decir el de las donas y lo pegaremos dentro del div con clase card-body

De las iguiente manera.

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"

(chartHover)="chartHovered($event)"

(chartClick)="chartClicked($event)"*></*canvas*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

No utilizaremos el evento chartClickm y chatHover, dejare la data de ladona, los labels y el chartType

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora vayamos al graficas1.component.ts y copiemos por ahora las propiedades que la documentación de donas nos ofrece debería quedr de las siguiente manera.

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-graficas1',

templateUrl: './graficas1.component.html',

styles: []

})

export class Graficas1Component implements OnInit {

*// Doughnut*

public doughnutChartLabels:string[] = ['Download Sales', 'In-Store Sales', 'Mail-Order Sales'];

public doughnutChartData:number[] = [350, 450, 100];

public doughnutChartType:string = 'doughnut';

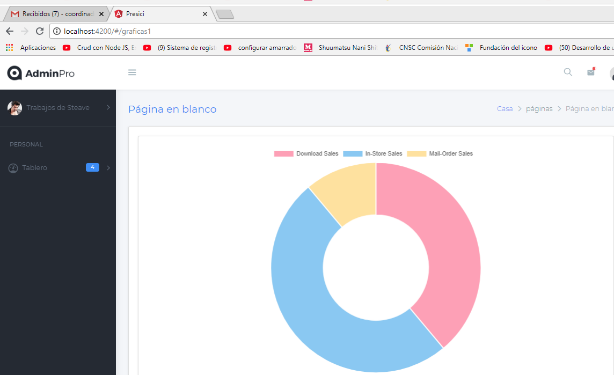
constructor() { }

ngOnInit() {

}

}

Al yo guardar todos etos cambios y observar el render ya podemos visualizar la grafica.



Hagamos un col 6 para hacer 2 tortas.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"*></*canvas*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"*></*canvas*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora hagamos 2 mas peguemos en el ts el siguiente objeto que tendrá la data para 4 donas como se muestra a continuación.

export class Graficas1Component implements OnInit {

*// Doughnut*

public doughnutChartLabels:string[] = ['Download Sales', 'In-Store Sales', 'Mail-Order Sales'];

public doughnutChartData:number[] = [350, 450, 100];

public doughnutChartType:string = 'doughnut';

graficos: any = {

'grafico1': {

'labels': ['Con Frijoles', 'Con Natilla', 'Con tocino'],

'data': [24, 30, 46],

'type': 'doughnut',

'leyenda': 'El pan se come con'

},

'grafico2': {

'labels': ['Hombres', 'Mujeres'],

'data': [4500, 6000],

'type': 'doughnut',

'leyenda': 'Entrevistados'

},

'grafico3': {

'labels': ['Si', 'No'],

'data': [95, 5],

'type': 'doughnut',

'leyenda': '¿Le dan gases los frijoles?'

},

'grafico4': {

'labels': ['No', 'Si'],

'data': [85, 15],

'type': 'doughnut',

'leyenda': '¿Le importa que le den gases?'

},

};

Lo primero es creemos el componente personalizado llamado grafico-dona.

Ng g c components/grafico-dona - -spect=false –is

Como voy a usar ese componente es en el modulo de pages, debo eliminar la importación automática de app.module.ts y también en el array de @NgModel, esto será para poder importarlo en el modulo Pages es decir pages.module.ts. si hacemos esto no ocurrirá ningún error ya que el componente esta declarado en un solo modulo.

Ahora el html de la dona que se encuentra en el graficos1.component.html me la llevare para el grafico-dona.html y la pegare alla.

Grafico-dona.component.html

*<*div style="display: block"*>*

*<*canvas baseChart

[data]="doughnutChartData"

[labels]="doughnutChartLabels"

[chartType]="doughnutChartType"*></*canvas*>*

*</*div*>*

Y en graficos1.component.html solo pegare la etiqueta o selector del componente grafico-dona que es <app-grafico-dona> quedando de la siguiente manera.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona*></*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora debere definir los propiedades de la dona en al componente hijo grafico-dona.component.ts.

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-grafico-dona',

templateUrl: './grafico-dona.component.html'

})

export class GraficoDonaComponent implements OnInit {

*// Doughnut*

public doughnutChartLabels:string[] = ['Download Sales', 'In-Store Sales', 'Mail-Order Sales'];

public doughnutChartData:number[] = [350, 450, 100];

public doughnutChartType:string = 'doughnut';

constructor() { }

ngOnInit() {

}

}

Al hacer esto ya podre observar nuevamente la dona en la vista de graficas1, limpiemos graficas1.component.ts la data de las propiedades de la dona

Agreguemos 3 donas mas. En el html de graficos1.component.html.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona*></*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona*></*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona*></*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona*></*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora declararemos las propiedades de grafico-dona.component.ts como @Input() es decir que serán variables que recibirán la información desde afuera osea de modo que quitare sus respectivos valores del componente padre graficos1.component.ts, tendras que importar el Input.

import { Component, OnInit, Input } from '@angular/core';

@Component({

selector: 'app-grafico-dona',

templateUrl: './grafico-dona.component.html'

})

export class GraficoDonaComponent implements OnInit {

*// Doughnut*

@Input() doughnutChartLabels:string[] = [];

@Input() doughnutChartData:number[] = [];

@Input() doughnutChartType:string = 'doughnut';

constructor() { }

ngOnInit() {

}

}

Lo que sigue es enviar la data del padre al hijo. Es decir necesito enviar desde graficas1.component.ts lo siguiente a grafico-dona.component.ts.

graficos: any = {

'grafico1': {

'labels': ['Con Frijoles', 'Con Natilla', 'Con tocino'],

'data': [24, 30, 46],

'type': 'doughnut',

'leyenda': 'El pan se come con'

},

'grafico2': {

'labels': ['Hombres', 'Mujeres'],

'data': [4500, 6000],

'type': 'doughnut',

'leyenda': 'Entrevistados'

},

'grafico3': {

'labels': ['Si', 'No'],

'data': [95, 5],

'type': 'doughnut',

'leyenda': '¿Le dan gases los frijoles?'

},

'grafico4': {

'labels': ['No', 'Si'],

'data': [85, 15],

'type': 'doughnut',

'leyenda': '¿Le importa que le den gases?'

},

};

Entoces lo que siguees colcoar esar propiedades en el html de graficas1 para enlazarlas.

Como se las voy a enviar desde una variable de angular las podre [] que serán igualadas con el objeto de graficos.

*<*div class="card-body"*>*

*<*app-grafico-dona

[doughnutChartLabels]="graficos.grafico1.labels"

[doughnutChartData]="graficos.grafico1.data"

[doughnutChartType]="graficos.grafico1.type"*>*

*</*app-grafico-dona*>*

*</*div*>*

Cuando termine de hacer esto y guarde ya estare enviando la data del padre al hijo y podre ver la dona nuevamente.



Si quiero ver los datos de las demás donas simplemente usare el objeto graficos haciendo referencia a grafico2,grafico3 y grafico4 y replicare en el html del padre de la siguiente manera.

*<*div class="row"*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona

[doughnutChartLabels]="graficos.grafico1.labels"

[doughnutChartData]="graficos.grafico1.data"

[doughnutChartType]="graficos.grafico1.type"*>*

*</*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona

[doughnutChartLabels]="graficos.grafico2.labels"

[doughnutChartData]="graficos.grafico2.data"

[doughnutChartType]="graficos.grafico2.type"*>*

*</*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona

[doughnutChartLabels]="graficos.grafico3.labels"

[doughnutChartData]="graficos.grafico3.data"

[doughnutChartType]="graficos.grafico3.type"*>*

*</*app-grafico-dona*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*app-grafico-dona

[doughnutChartLabels]="graficos.grafico4.labels"

[doughnutChartData]="graficos.grafico4.data"

[doughnutChartType]="graficos.grafico4.type"*>*

*</*app-grafico-dona*>*

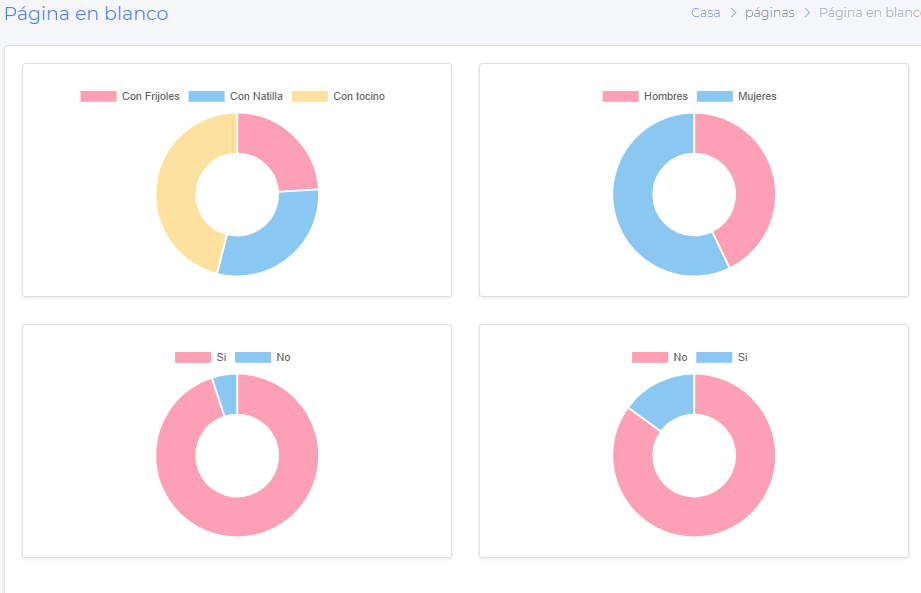
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Podremos observar los resultados.



Ahora guardemos los cambios realizados hasta ahora en github

1- **git status** para ver que vamos a subir

2- **git add .** para agregar todo al stagin área

3- **git commit –m “input output y viewchild en graficas1 y 2”** para agregar el commit

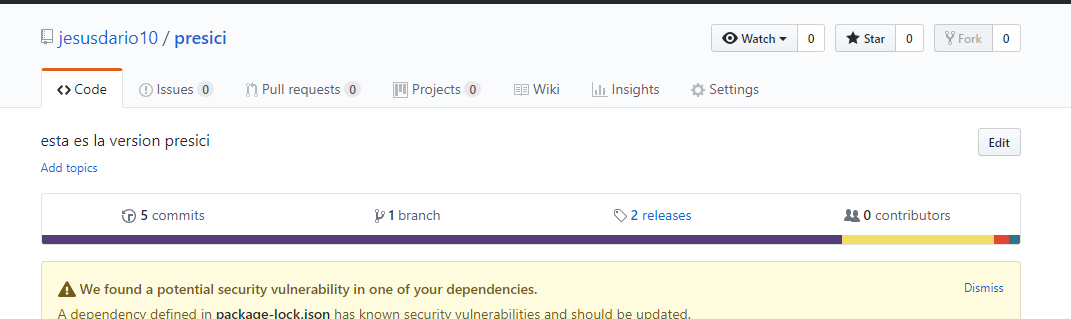
4- **git push origin master** para subirlo a github

5- **git tag –a v1.1.0 –m “Version input,output, viewchild, graficas1 y2” creando la version2** es decir un nuevo tag

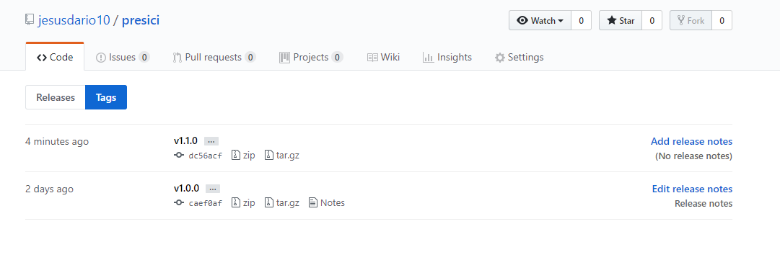
6- **git tag** para verificar el versionamiento deben salir todas las versiones a las que les hemos hecho tag

7- **git push - - tags** para subir los tag a github

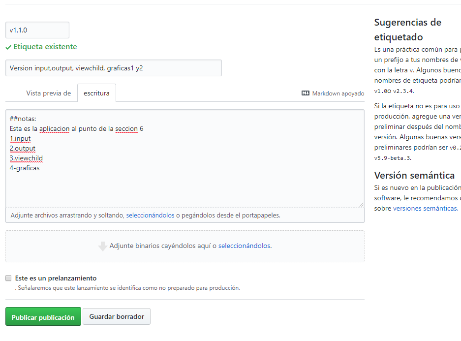
8- la verificamos en github en el proyecto la opción reléase



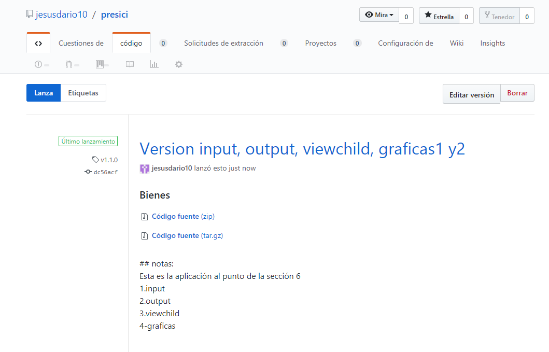
9-luego en la pestaña tags podre observar mis versiones



10-agregaremos un reléase notes para que no se me olvide que es



Publicamos y aquí tenemos la nueva versión registrada



**Diseño Inicial de la pagina account-setting.**

Creemos un nuevo componente dentro de pages para el accountsetting.

ng g c pages/accountSettings -m="pages/pages.module.ts" --spec=false –is

Nota1: cuando uso cammelcase para crear modulos o servicios, angular cli los crea siempre con un –(gion).

Nota2: cuando quiero crear un componente y que este se cargue en un modulo en especifico usare el –m que quiere decir module y le especificare el modulo.

Vamos al account-setting.component.html y crearemos los temas. Coloquemos esto dentro del card principal que podemos copiar del pages.component.html

*<*div class="row"*>*

*<*div class="col-5"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Temas*</*h4*>*

*<*div class="r-panel-body"*>*

*<*ul id="themecolors" class="m-t-20"*>*

*<*li*><*b*>*Con el sidebar claro*</*b*></*li*>*

*<*li*><*a data-theme="default" class="selector default-theme"*>*1*</*a*></*li*>*

*<*li*><*a data-theme="green" class="selector green-theme"*>*2*</*a*></*li*>*

*<*li*><*a data-theme="red" class="selector red-theme"*>*3*</*a*></*li*>*

*<*li*><*a data-theme="blue" class="selector blue-theme"*>*4*</*a*></*li*>*

*<*li*><*a data-theme="purple" class="selector purple-theme"*>*5*</*a*></*li*>*

*<*li*><*a data-theme="megna" class="selector megna-theme"*>*6*</*a*></*li*>*

*<*li class="d-block m-t-30"*><*b*>*Con el sidebar oscuro*</*b*></*li*>*

*<*li*><*a data-theme="default-dark" class="selector default-dark-theme"*>*7*</*a*></*li*>*

*<*li*><*a data-theme="green-dark" class="selector green-dark-theme"*>*8*</*a*></*li*>*

*<*li*><*a data-theme="red-dark" class="selector red-dark-theme"*>*9*</*a*></*li*>*

*<*li*><*a data-theme="blue-dark" class="selector blue-dark-theme working"*>*10*</*a*></*li*>*

*<*li*><*a data-theme="purple-dark" class="selector purple-dark-theme"*>*11*</*a*></*li*>*

*<*li*><*a data-theme="megna-dark" class="selector megna-dark-theme"*>*12*</*a*></*li*>*

*</*ul*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

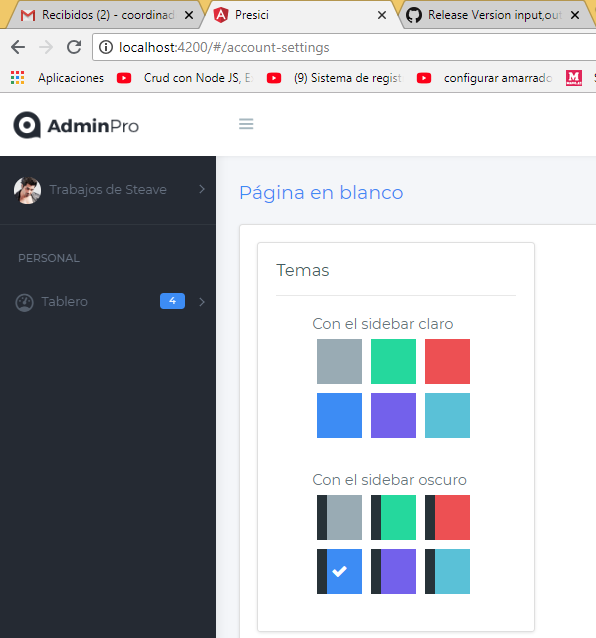
Ahora creemos la ruta para llegar a esa pagina:

En pages.routes.ts crearemos la nueva ruta por lo tanto debemos importar el componente antes de crearla.

import { AccountSettingsComponent } from './account-settings/account-settings.component'

{path: 'account-settings', component: AccountSettingsComponent},

Cambiemos el tema principal de nuestra aplicacion al hacer click en uno de los cuadros de temas

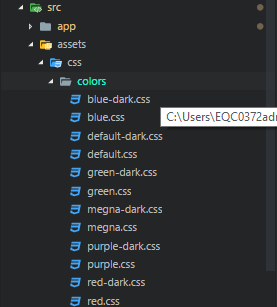


Vayamos un momento al **index.html** y observemos lo siguiente, con este link es que se define el tema asi que sera a ese elemento al que haremos referencia para cambiar el tema.

*<!-- You can change the theme colors from here -->*

*<*link href="assets/css/colors/default-dark.css" id="theme" rel="stylesheet"*>*

Si nosotros quisieramos cambiar el tema los podemos encontrar en src/assets/css/colors/**yaquidentroestantodoslostemas**.



Entonces como hacemos ese cambio de formadinamica?

Si observamos el account.setting cada li tiene un atributo con el nombre del tema

*<*li*><*a data-theme="red-dark" class="selector red-dark-theme"*>*9*</*a*></*li*>*

Hay que leer ese el **data-theme** y enviárselo al link del index.html

Primero que nada index.html esta fuera del scope del account.setting. asi que lo primero que haremos sera ponerle un id a ese elemento link del index.html.

*<*link id="tema" href="assets/css/colors/default-dark.css" id="theme" rel="stylesheet"*>*

Luego crearnos un escuchador con el evento click para que al darle click yo pueda saber que color queiren usar por medio de una función que recibirá el color.

*<*div class="row"*>*

*<*div class="col-4"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Temas*</*h4*>*

*<*hr*>*

*<*div class="r-panel-body"*>*

*<*ul id="themecolors" class="m-t-20"*>*

*<*li*><*b*>*Con el sidebar claro*</*b*></*li*>*

*<*li*><*a (click)="cambiarColor('default')" data-theme="default" class="selector default-theme"*>*1*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('green')" data-theme="green" class="selector green-theme"*>*2*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('red')" data-theme="red" class="selector red-theme"*>*3*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('blue')" data-theme="blue" class="selector blue-theme"*>*4*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('purple')" data-theme="purple" class="selector purple-theme"*>*5*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor(megna)" data-theme="megna" class="selector megna-theme"*>*6*</*a*></*li*>*

*<*li class="d-block m-t-30"*><*b*>*Con el sidebar oscuro*</*b*></*li*>*

*<*li*><*a (click)="cambiarColor('default-dark')" data-theme="default-dark" class="selector default-dark-theme"*>*7*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('green-dark')" data-theme="green-dark" class="selector green-dark-theme"*>*8*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('red-dark')" data-theme="red-dark" class="selector red-dark-theme"*>*9*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('blue-dark')" data-theme="blue-dark" class="selector blue-dark-theme working"*>*10*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('purple-dark')" data-theme="purple-dark" class="selector purple-dark-theme"*>*11*</*a*></*li*>*

*<*li*><*a (click)="cambiarColor('purple-dark')" data-theme="megna-dark" class="selector megna-dark-theme"*>*12*</*a*></*li*>*

*</*ul*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora deberemos en la clase del acount-setting crear el metodo cambiarColor.

cambiarColor(tema:string){

console.log(tema);

}

Ahora como tendre que crear referencia a todo el DOM del proyecto para poder acceder al index.html lo hare de la siguiente manera. Primero importar los siguientes modulos:

El Inject y el DOCUMENT

import { Component, OnInit, Inject } from '@angular/core';

import { DOCUMENT } from '@angular/platform-browser';

luego inyectarlo como un servicio en el constructor

constructor(@Inject(DOCUMENT) private \_document) { }

con el objeto declarado **\_document** ya tenemos acceso a todo el DOM

ahora para poder cambiar de la etiqueta link con id tema del index haremos lo siguiente

cambiarColor(tema:string){

console.log(tema);

this.\_document.getElementById('tema').setAttribute('href',)

}

Pero como obtenemos la ruta Para cambiar el contenido del href? La solución es simple.

Vamos al index.html tomamos el path del link que queremos modificar

*<*link id="tema" href="**assets/css/colors/default-dark.css**"

Declaramos una variable local en el componente account-settings y lo igualamos hasta el color porque el resto lo concatenaremos. Todo esto po puedo hacer con las comillas especiales que permiten la interpolación con el signo $. Luego el segundo parámetro del href sera la url y listo.

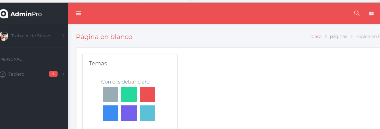
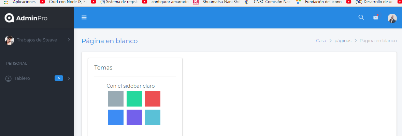
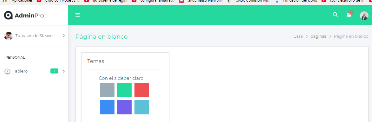
cambiarColor(tema:string){

let url = `assets/css/colors/${tema}.css`;

this.\_document.getElementById('tema').setAttribute('href', url);

}

Y perfecto observemos los cambios:

 -----

**Como agrupamos todos los ervicios en un solo modulo?**

1: creemos un modulo de servicios

2: **ng g m services/service - -spec=false - -flat**

3: ahora crare 2 carpetas dentro de services para clasificar mis servicios, una settings y allí meteré el servicio de seting, la otra carpeta se llamara shared y meteré el servicio de sidebar.servicesy shared.services

4: ahora creare en la raiz de services un archivo llamado services.index.ts donde importaremos los 3 archivos de los servicios que tenemos.

import { SettingsService } from './settings/settings.service';

import { SharedService } from './shared/shared.service';

import { SidebarService } from './shared/sidebar.service';

luego cambiaremos el import por exports para poder exportarlos.

export { SettingsService } from './settings/settings.service';

export { SharedService } from './shared/shared.service';

export { SidebarService } from './shared/sidebar.service';

ahora tanto en el modulo principal como en los otros al usar los servicios debere hacer referencia al archivo service.index.ts. por ejemplo en el app.module.ts la importación sera.

*//servicios*

import { SettingsService } from './services/service.index';

si necesitara usar otro servicio aparte del settings services simplemente lo separaría por comas y lo añadiría a los providers.

Ahora para que me sirvió el services.module que cree bueno para dejarlos mejor organizados. Entonces que haremos importaremos todos los serviciso allí uevamente pero ya apuntando al index. Luego los agregamos a los providers ya que son servicios.

Ahora en el app.module.ts podremos importar el modulo service.module.ts de la siguiente manera. De modo que deberemos quitar el servicio importado en los providers e importar el modulo de servicios enlos imports.

*//modulo de servicios*

import { ServiceModule } from './services/service.module';

imports: [

BrowserModule,

APP\_ROUTES,

PageModule,

FormsModule,

ServiceModule

],

**Creeos servicios para administrar el sidebar, vayamos al sicebarservices.ts**

Por el momento vayamos al servicio sidebar.service.ts y creemos un objeto menú de tipo any que va a ser igual a un arreglo de objetos, con las siguientes propiedades.

El titulo sera la cabecera del menú, el icono sera el icono que tenga la cabecera del menú, el submenú serán las opciones a clickear que me llevaran a las rutas.

menu : any = [

{

titulo: "Principal",

icono: "mdi mdi-gauge",

submenu:[

{ titulo:"Dashboard", url:"/dashboard" },

{ titulo:"ProgressBar", url:"/progress" },

{ titulo:"Grafica1", url:"/graficas1" },

{ titulo:"Grafica2", url:"/graficas2" }

]

}

];

Ahora vamos al componente del sidebar para implementar el objeto menú. Es decir abramos el archivo sidebar.component.ts. en este archivo deberemos importar el servicio sidebar.service.ts e inicializarla en el constructor.

import { Component, OnInit } from '@angular/core';

import { SidebarService } from '../../services/service.index';

@Component({

selector: 'app-sidebar',

templateUrl: './sidebar.component.html',

styleUrls: ['./sidebar.component.css']

})

export class SidebarComponent implements OnInit {

constructor( public \_sidebar:SidebarService) { }

ngOnInit() {

}

}

Ahora para poder hacer ruteo necesitare importar en el shared.module.ts el modulo de Router que se llama RouterModule.

import { RouterModule } from '@angular/router';

y agreguémoslo a los imports del @NgModule

@NgModule({

imports:[

RouterModule

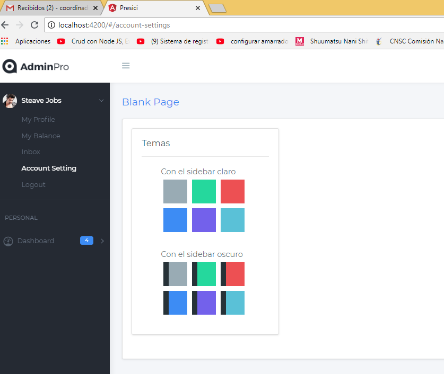
],

Ahoraen la vista del sidebar.component.html usaremos el router.link para la opción account-setting y revisar el ruteo después.

El routerLink envía la ruta y el routerLinkActive activa la opción.

*<*li*><*a routerLinkActive="active" routerLink="/account-settings"*>*Account Setting*</*a*></*li*>*

Al revisar el ruteo todo anda bien al dar click en la opción account-setting se redirecciona a esa ruta y me muestra su contenido.



Ahora si consentremonos en el menú, utilicemos la directiva \*ngFor para tomar el objeto del menú que me trae el servicio.

*<*li class="nav-small-cap"*>*PERSONAL*</*li*>*

*<*li \*ngFor="let menu of \_sidebar.menu"*>* *<*a class="has-arrow waves-effect waves-dark" href="#" aria-expanded="false"*><*i class="mdi mdi-gauge"*></*i*><*span class="hide-menu"*>*Dashboard *<*span class="label label-rouded label-themecolor pull-right"*>*4*</*span*></*span*></*a*>*

*<*ul aria-expanded="false" class="collapse"*>*

*<*li*><*a href="index.html"*>*Minimal *</*a*></*li*>*

*<*li*><*a href="index2.html"*>*Analytical*</*a*></*li*>*

*<*li*><*a href="index3.html"*>*Demographical*</*a*></*li*>*

*<*li*><*a href="index4.html"*>*Modern*</*a*></*li*>*

*</*ul*>*

*</*li*>*

Esto nos dara un error sino importamos el CommonModule en el shared.module.ts, dicho modulo nos activa las directivas angulares.

import { CommonModule } from '@angular/common';

ahora tambien agreguemos a los imports

@NgModule({

imports:[

RouterModule,

CommonModule

],

Ahora si de regreso al html es decor el sidebar.component.html

Quitemos el href del menú principal para que no redireccione, cambiemos su icono usando la implementación de las clases de angular. Y el titulo también cambiémoslo utilizando la interpolación de las {{}}, el 4 estatico que tiene cambiémoslo por el numero de sub elementos del objeto menú {{menú.submenu.length}}

*<*li class="nav-small-cap"*>*PERSONAL*</*li*>*

*<*li \*ngFor="let menu of \_sidebar.menu" *>*

*<*a class="has-arrow waves-effect waves-dark" href="#" aria-expanded="false"*>*

*<*i [class]="menu.icono"*></*i*>*

*<*span class="hide-menu"*>*{{menu.titulo}}

*<*span class="label label-rouded label-themecolor pull-right"*>*{{menu.submenu.length}}*</*span*>*

*</*span*>*

*</*a*>*

*<*ul aria-expanded="false" class="collapse"*>*

*<*li*><*a href="index.html"*>*Minimal *</*a*></*li*>*

*<*li*><*a href="index2.html"*>*Analytical*</*a*></*li*>*

*<*li*><*a href="index3.html"*>*Demographical*</*a*></*li*>*

*<*li*><*a href="index4.html"*>*Modern*</*a*></*li*>*

*</*ul*>*

*</*li*>*

Ahora el hay que traer las opciones del submenú y ese también lo haremos desde un ngForm.

Los RouterLinks como son de manera dinámica y como puede que lleven algún parámetro serán declarados de la siguiente forma [routerLink]=”[submenú.url]”

*<*ul aria-expanded="false" class="collapse"*>*

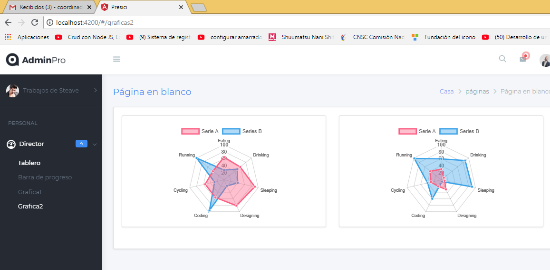
*<*li \*ngFor="let submenu of menu.submenu"*>*

*<*a [routerLink]="[submenu.url]" *>*{{ submenu.titulo }}*</*a*>*

*</*li*>*

*</*ul*>*

Después de esto nuestro menú es completamente funcional.



**Uso de scripts d Archivos importados en el index.html en typescript:**

Hagamos que desde la vista login al dar en el botón ingresar nos lleve al dashboard. Debemos ir al login.component.html.

En el form borraremos el id y el action y declaremos un enevto submit que ejecute un metodo en el componente llamado ingresar.

*<*form class="form-horizontal form-material" (ngSubmit)="ingresar()"*>*

Ahora creemos ese metodo en el login.component.ts, este metodo hara uso del routerlink para que me lleve al dahsboard.

Debemos importar el servicio Router e inicializarlo en el constructor.

import { Router } from '@angular/router';

constructor(public router: Router) { }

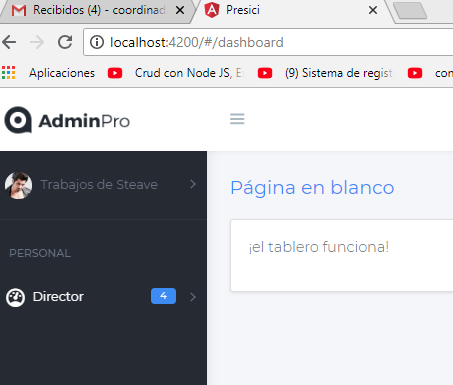
y ahora utilicemos el metodo navigate en el metodo ingresar para que se de la redirección al dashboard:

ingresar(){

this.router.navigate(['/dashboard']);

}

Aquí esto nos redireccionara al dashboard pero existe un problema que las opciones del menú no funcionan, ya no se abren.



Y esto se debe a que el template esta utilizando un plugin para el menú, este plugin comprime ítems y las opciones y demás. Pero este plugin ya no funciona.

Y no funciona porque cuando el plugin es llamado en el index.html el plugin se llama

*<!--Menu sidebar -->*

*<*script src="assets/js/sidebarmenu.js"*></*script*>*

Pero en el assets/js/custom.min.js es donde se llama la inicialización de los plugins

*<!--Custom JavaScript -->*

*<*script src="assets/js/custom.min.js"*></*script*>*

*<!-- ============================================================== -->*

Enconces cuando pasaaos del login al dashboard el menú en el login no existe por consecuencia no se dispara la inicialización y cuando llegamso al dashboard los menus no funcionan. Avbramos este archivo pero como esta minimizado va a ser difícil de leer, asi que como tenemos el normal enlacemoslo a ese es decir de .min pasemos solo a .js como se muestra a continuación.

*<!--Custom JavaScript -->*

*<*script src="assets/js/custom.js"*></*script*>*

*<!-- ============================================================== -->*

abramos ese archivo custon.js, ojo este solo es un segmento para ilustrar, en este caso mostremos la inicialziacion del sidebar

var set = function() {

var width = (window.innerWidth > 0) ? window.innerWidth : this.screen.width;

var topOffset = 0;

if (width < 1170) {

$("body").addClass("mini-sidebar");

$('.navbar-brand span').hide();

$(".sidebartoggler i").addClass("ti-menu");

Como podemos llamar a todos los plugins estos de jquery fuera de angular desde algún componente de angular que me la active?

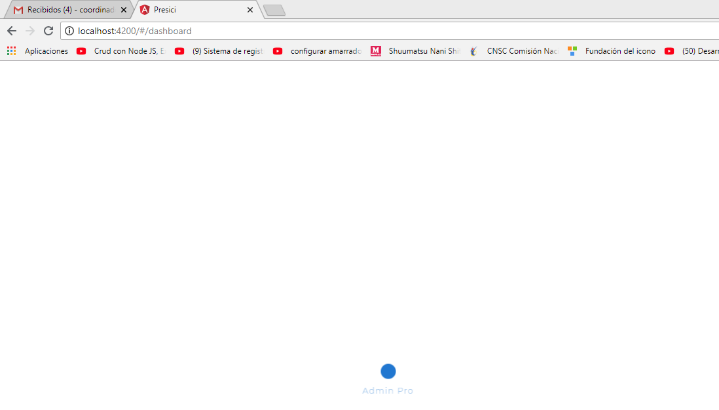
Lo primero es que en el plugin custon.js declararemos una función al inicio llamada **init\_plugins()** que me enciere toda la función principal de jquery es decir que encapsule todo el código del archivo. Para ser mas claro la función **init\_plugins()** tendrá dentro todo el código del archivo custon.js algo como esto.

**function init\_plugins(){**

**todo el código del archivo**

**}**

Cuando hagamos esto y guardemos nos daremos cuenta que se quedara cargando siempre es decir aparecerá el icono de cargue y no se quitara.



Como resolvemos esto para que cargue todo y por ende se quite el prelouder y carguen los componentes para los menus?

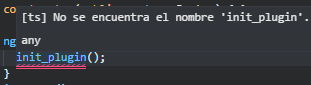
Para eso iremos al login.component.ts y lalmaremos a la función que creamos en el ngOnInit

ngOnInit() {

init\_plugin();

}

Podemos observar que nos marca error porque no la conoce.



Como yo se que existe lo que hare sera declararla al antes de la clase del componente de la siguiente manera:

declare function init\_plugin();

si yo hago esto cuando le de al botón ingresar del login me va a redirigir al dashboar, pero si yo recargo en el dashboard nuevamente se quedara cargando. Entonces resumanos

hasta ahora este es el componente login:

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

declare function init\_plugin();

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

constructor(public router: Router) { }

ngOnInit() {

init\_plugin();

}

ingresar(){

this.router.navigate(['/dashboard']);

}

}

Para resolver el problema actual deberemos ir componente que embuelve todos las componentes y paginas? Pages.component.ts

Deberemos ir alla y hacer lo mismo que hicimos con el loginComponent.ts.

import { Component, OnInit } from '@angular/core';

declare function init\_plugin();

@Component({

selector: 'app-pages',

templateUrl: './pages.component.html',

styles: []

})

export class PagesComponent implements OnInit {

constructor() { }

ngOnInit() {

init\_plugin();

}

}

Con esto resolvemos el problema. Los plugins se cargaran tanto en el login como al inicializar el pages.component.ts. De esta manera podremos llamar cualquier archivo que se encuentre fuera de angular en un archivo de javascript, esto funciona mucho con carruseles, tooltips y otras cosas, que están externas que están hechas en jquey y que no cargan directamente en angular o typescrip.

Guardemos en github nuestros cambios y versionemos nuevamente.

Ahora guardemos los cambios realizados hasta ahora en github

1- **git status** para ver que vamos a subir

2- **git add .** para agregar todo al stagin área

3- **git commit –m “input output y viewchild en graficas1 y 2”** para agregar el commit

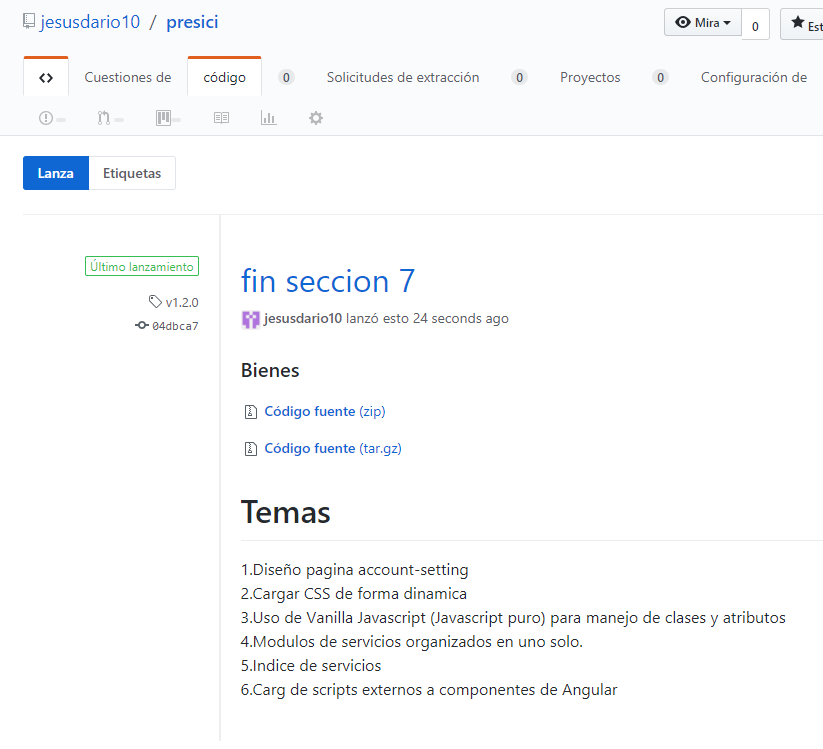
4- **git push origin master** para subirlo a github

5- **git tag –a v1.1.0 –m “Version input,output, viewchild, graficas1 y2” creando la version2** es decir un nuevo tag

6- **git tag** para verificar el versionamiento deben salir todas las versiones a las que les hemos hecho tag

7- **git push - - tags** para subir los tag a github

8- la verificamos en github en el proyecto la opción reléase



Esta sección tiene por objetivo principal, trabajar con observables y promesas:

* Tendremos una introducción ilustrativa para explicar estos dos temas
* Trabajaremos con promesas y funciones que retornan promesas
* Aprenderemos a crear un observable manualmente
* Trabajaremos con operadores de los observables como:
  + Retry
  + Map
  + Filter
  + Next
* Funciones que retornan observables
* Usaremos el conocimiento aprendido para crear un componente de seguimiento de la página actual
* Usaremos observables para leer parámetros de configuración de las rutas que son diferentes a los parámetros de las rutas por url
* Cambiaremos los metatags dependiendo de la página donde nos encontremos
* Cambiar el titulo de la página actual

Al finalizar, realizamos el respectivo backup en GitHub, junto a su release TAG.

**Observables y promesas:**

**Similitudes:** ambas son para trabajar con procesos asíncronos, no necesariamente son peticiones http, consumo de apis etc, en realidad nos sirven para trabajar con procesos que esperan una respuesta que no viene de forma instantánea.

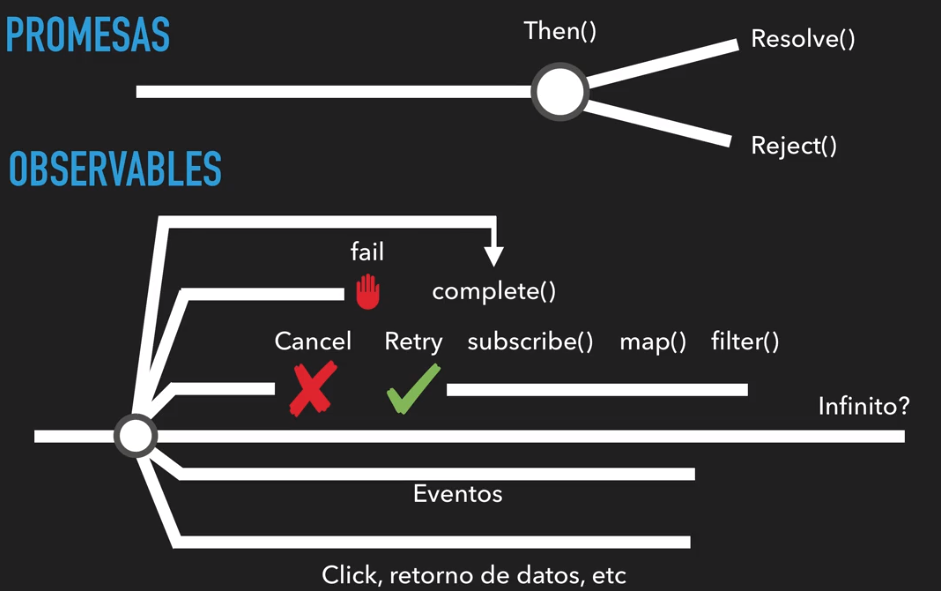
Características de las promesas:

1. Trabajan con un único flujo de datos.
2. Se usan con una única data asíncrona de repuesta
3. No es muy simple de cancelar
4. Sin simples de implementar para tareas sencillas

Caracteristicas de los observables:

1. Son mas robustos que las promesas
2. Trabajan con un único flujo continuo de datos es decir pueden estar escuchando una gran cantidad de respuestas.
3. Al fallar pueden ejeutar comandos y reintentar continuar con el observer.
4. Se pueden encadenar operadores adicionales como el map, forEach, reduce, filter y mas.
5. Existen otros operadores muy potentes como el retry() o el replace()
6. Pueden ser creados desde otras fuentes como eventos
7. Son funciones a las cuales podemos suscribirnos en varios lugares.

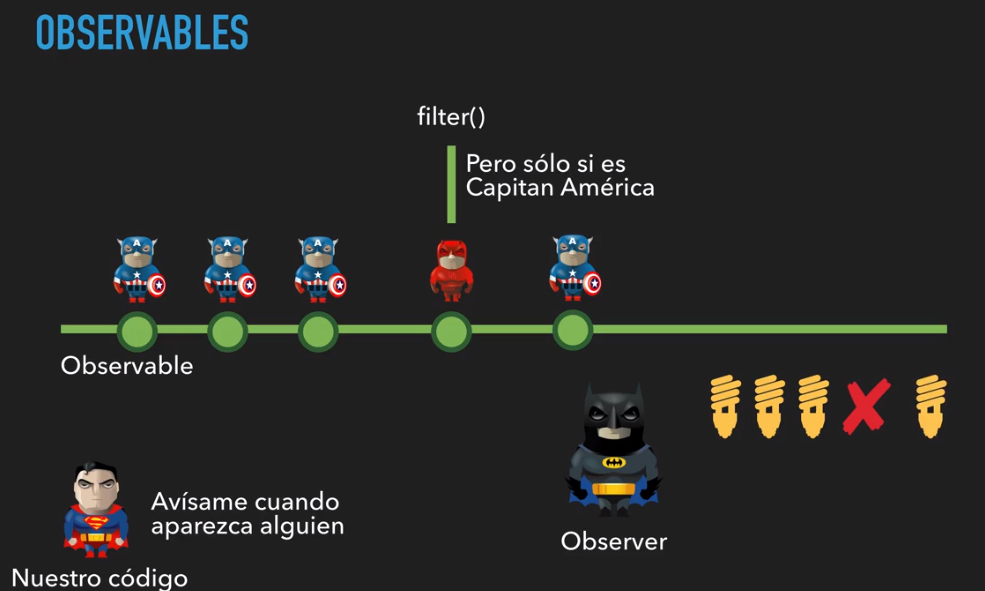
Comparacion



Promesas: cuando encuentres a capitán america avisame



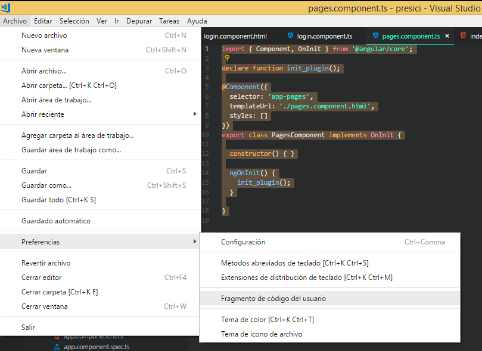
Observables:

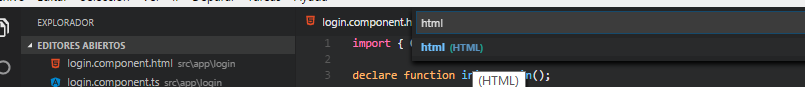


**Snippet** es un término del idioma inglés utilizado en **programación** para referirse a pequeñas **partes reusables de código fuente**, código máquina o texto. Comúnmente son definidas como unidades o métodos funcionales que se pueden integrar fácilmente en módulos mucho más grandes, aportando funcionalidad.

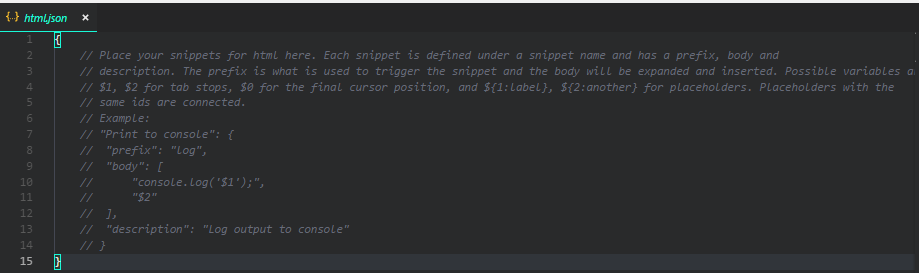
**Como decharamos un snippet**?

Para crear un snipet con visual studio code debemos ir a archivo ->preferencias->fragmentos de código de usuario y nos pedirá para que lenguaje creare el snipet, en este caso tomaremos el card del pages para poder reutilizarlo de mejor manera.





Se creara el iguiente archivo



Y aquí empesaermos a crear el snipet, notemos que parece un objeto de javascript pero realmente en un json. Y allí mismo esta el ejemplo de como usarlo.

*// Example:*

*// "Print to console": {*

*//  "prefix": "log",*

*//  "body": [*

*//      "console.log('$1');",*

*//      "$2"*

*//  ],*

*//  "description": "Log output to console"*

*// }*

Entonces guiemosnos con el y creémoslo de la siguiente manera,

1. **El primer campo** sera el nombre del snippet
2. **El campo prefix**: sera el comando con el que llamare al snippet
3. **El campo body**:es un array donde agregaremos el json es decir el html
   1. **Regla1:** cada línea del html debe empezar por comillas y terminar tambien.
   2. **Regla2**: para las comillas intermedias siempre deberemos antemoner el símbolo \ baqueshlas
   3. **Regla3**: cada línea debe terminar con una coma eceptuando la ultima.
4. **El campo description**: tendrá la description del snippet.

    "Tarjeta con Row": {

        "prefix": "cardrow",

        "body": [

        "   <div class=\"row\">",

        "   <div class=\"col-12\">",

        "    <div class=\"card\">",

        "       <div class=\"card-body\">",

        "        this some text ",

        "       </div>",

        "    </div>",

        "   </div>",

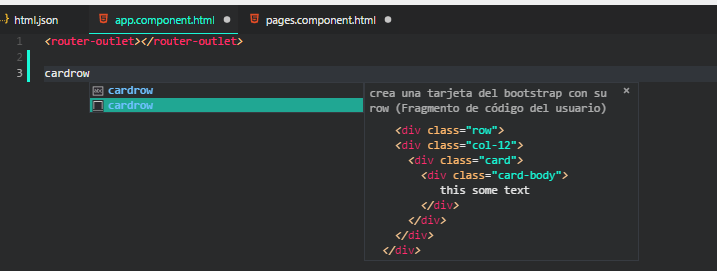
        " </div>"

        ],

        "description": "crea una tarjeta del bootstrap con su row"

     }

1. Cuando terminemos guardamos y ya podemos usar el snippet. Por ejemplo vayamos a app.component.html y escribamos el prefix definido es decir cardrow



Al darle tab o enter me creara todo el código que defini.

*<*router-outlet*></*router-outlet*>*

*<*div class="row"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

this some text

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Iniciemos creando promesas para el proyecto. Entonces creemos un modulo dentro de pages que las controle y digamos que lo inicialice en el modulo de pages.

**ng g c pages/promesas --spec=false –is**

Crearemos una ruta para las promesas para poder ver que es lo que colcoare allí

{path: 'promesas', component: PromesasComponent},

Y moduficaremos el menú para que sea una opción seleccionable entonces deberemos ir al sidebar.services.ts

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class SidebarService {

menu : any = [

{

titulo: "Principal",

icono: "mdi mdi-gauge",

submenu:[

{ titulo:"Dashboard", url:"/dashboard" },

{ titulo:"ProgressBar", url:"/progress" },

{ titulo:"Grafica1", url:"/graficas1" },

{ titulo:"Grafica2", url:"/graficas2" },

{ titulo:"Promesas", url:"/promesas" }

]

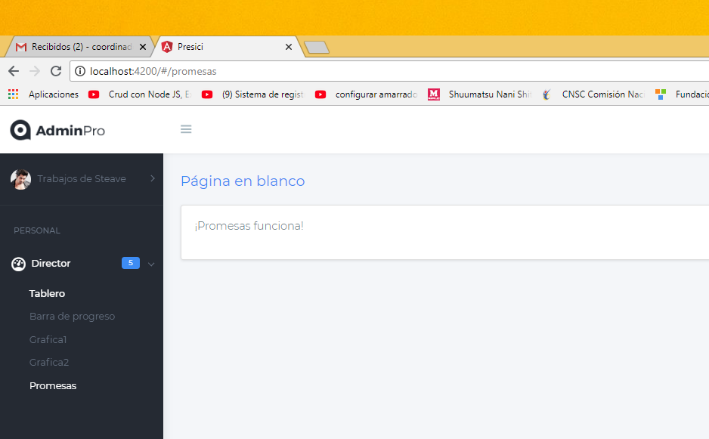
}

];

constructor() { }

}

Podemos observar que ya tenemos todo



Yo quisiera realizar una tarea que me notifique un intervalo de tiempo cuando este ejecute 3 segundos.

Entonces vayamos al componente promesas y definamos un metodo llamado contar3 en la clase para ejecutarla, tambien definamos una variable promesa que instance una promesa, toda promesa recibe una función anónima con 2 parametros el resolve que es la repuesta y el reject que es el fallo.

contar3(): Promise<boolean>{

let contador = 0;

return new Promise((resolve, reject)=>{

let intervalo = setInterval(()=>{

contador+=1;

console.log(contador);

if(contador==3){

resolve(true);

clearInterval(intervalo);

}

},1000);

});

}

Para solucionar coloquemos un set interval que recibe una función anomina y un intervalo de tiempo.

Para esto crearemos una variable llamada contador inicializada en 0 y en el ser lo incrementaremos en 1 cada que se ejecute.

Si el contador se iguala a 3 llamaremos al resolve() y un clearInterval. El resolve se enlaza con el metodo then que ejecutara la promesa y esta a su ves podrá capturar el error con el metodo cath.

constructor() {

this.contar3().then(

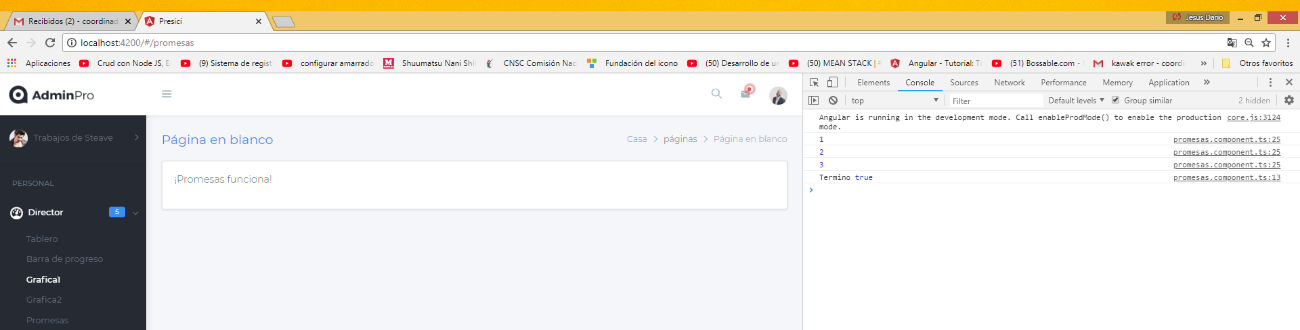
mensaje => console.log("Termino", mensaje)

).

catch(error => console.error('Error en la promesa', error))

}

La siguiente es una imagen del resultado.



Para empezar a trabajar con los observables creemos un nuevo componente en pages llamado rxjs que este inicializado en el pages.module.ts y que no tenga estilos ni es spec

ng g c pages/rxjs --spec=false –is

agreguemos la ruta rxjs y agreguemosla al menu.

Pages.routes.js

import { RxjsComponent } from './rxjs/rxjs.component';

{path: 'rxjs', component: RxjsComponent},

Agreguemosla al menú

})

export class SidebarService {

menu : any = [

{

titulo: "Principal",

icono: "mdi mdi-gauge",

submenu:[

{ titulo:"Dashboard", url:"/dashboard" },

{ titulo:"ProgressBar", url:"/progress" },

{ titulo:"Grafica1", url:"/graficas1" },

{ titulo:"Grafica2", url:"/graficas2" },

{ titulo:"Promesas", url:"/promesas" },

{ titulo:"RXJS", url:"/rxjs" }

]

}

];

Creemos un observable manual en el rxjs.component.ts

Para usar los observables debemos importar su modulo.

import { Observable} from 'rxjs';

import { map } from 'rxjs/operators';

primero creare el observable infinito en el contructor.

constructor() {

let contador = 1;

let obs = new Observable( observer =>{

let intervalo = setInterval(()=>{

observer.next(contador);

},1000)

});

}

Para poder escuchar lo que el obs hace debo suscribirme

constructor() {

let contador = 1;

let obs = new Observable( observer =>{

let intervalo = setInterval(()=>{

observer.next(contador);

},1000);

});

obs.subscribe(numero=>{

console.log("Subscripcion", numero);

})

}

Aun si me cambio de pagina el seguirá escuchando siempre.

Ahora hagamos que si contador es 3 detengamos el intervalo y aunque lo detenemos podemos darnos cuenta que el observable no se detiene. Para notificar que el observable termino hacemos lo siguiente ejecutamos el observer.complete y en el subscribe pasamos 3 funciones de callback la primera con la respuesta la segunda con el error y la tercera vacia que significara que si se da la condición de complete() detendrá el observable.

constructor() {

let contador = 1;

let obs = new Observable( observer =>{

let intervalo = setInterval(()=>{

observer.next(contador);

contador++

if(contador===3){

clearInterval(intervalo);

observer.complete();

}

},1000);

});

obs.subscribe(

numero=> console.log("Subscripcion", numero),*//este es cuando se recibe algo del next*

error => console.error("error en el obs", error),*//esto nes cuando pasa un error*

()=> console.log("el observador termino")*//cuando el observable termina*

);

}

Ahora hagamos que ocurra un error si contador llega a 2:

if(contador>3){

clearInterval(intervalo);

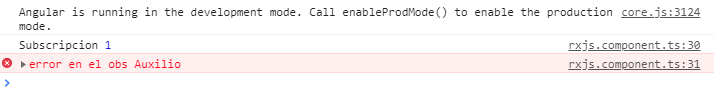
observer.complete();

}

if(contador==2){

observer.error(‘auxilio’);

}



El operador **Retry** nos ayudara a volver a enviar la solicitud de ejecución del metodo. Todos los observables tienen un **pipe**, un **pipe** es una tubería que permite transformar la información o hacer un proceso de transformación de datos. La función **pipe** nos permite definir varios operadores como parámetros, entre esos esta el **retry**, que solicitara importar el

import { map } from 'rxjs/operators';

entonces para usarlo sera de la siguiente forma:

obs.pipe(

retry()

)

Añadiéndolo al subscribe, si le paso 2 como parámetro quiere decir que hara dos intentos si ocurre un error.

constructor() {

let contador = 1;

let obs = new Observable( observer =>{

let intervalo = setInterval(()=>{

observer.next(contador);

contador++

if(contador===3){

clearInterval(intervalo);

observer.complete();

}

if(contador===2){

clearInterval(intervalo);

observer.error('Auxilio');

contador = 0;

}

},1000);

});

obs.pipe(

retry(2)

)

.subscribe(

numero=> console.log("Subscripcion", numero),*//este es cuando se recibe algo del next*

error => console.error("error en el obs", error),*//esto nes cuando pasa un error*

()=> console.log("el observador termino")*//cuando el observable termina*

);

}

**Funcionen que retornen observables, por lo general esto no se hace en el constructor asi que creemos una función que lo haga y se hace de la siguiente manera.**

constructor() {

this.regresaObservable().pipe(

retry(2)

)

.subscribe(

numero=> console.log("Subscripcion", numero),*//este es cuando se recibe algo del next*

error => console.error("error en el obs", error),*//esto nes cuando pasa un error*

()=> console.log("el observador termino")*//cuando el observable termina*

);

}

regresaObservable(): Observable<number>{

let contador = 1;

return new Observable( observer =>{

let intervalo = setInterval(()=>{

observer.next(contador);

contador++

if(contador===3){

clearInterval(intervalo);

observer.complete();

}

if(contador===2){

clearInterval(intervalo);

observer.error('Auxilio');

contador = 0;

}

},1000);

});

}

**Operador map: para esto debemos hacer un par de modificaciones lo primero es que el tipo de observable que se regrese sea de tipo any. Y también el subscriber**

regresaObservable(): Observable<any>{

let contador = 1;

return new Observable( (observer: Subscriber<any>) =>{

quitemos o comentemos la condicion contador = 2 no la necesitaremos y por ende el retry también lo quitamos.

constructor() {

this.regresaObservable()

.subscribe(

numero=> console.log("Subscripcion", numero),*//este es cuando se recibe algo del next*

error => console.error("error en el obs", error),*//esto nes cuando pasa un error*

()=> console.log("el observador termino")*//cuando el observable termina*

);

}

Vamos a imaginarnos que este setInterval es una llamada a una API que nosotros no hicimos y que solo estamos consumiendo.

let intervalo = setInterval(()=>{

observer.next(contador);

contador++

if(contador>3){

clearInterval(intervalo);

observer.complete();

}

},1000);

No sigo con el operador map no me parece funcional hasta ahora

Operador filter. Es ul filtro es útil pero no aquí.

**Creemos el Breadcrums usando observables y promesas**

Primeramente al page.routing.ts para darles un alias a las rutas y eso lo haremos usando el metodo data de la siguiente manera.

{path: 'dashboard', component: DashboardComponent, data:{titulo:'Dasboard'}},

{path: 'progress', component: ProgressComponent, data:{titulo:'Progress'}},

{path: 'graficas1', component: Graficas1Component, data:{titulo:'GradicasTorta'}},

{path: 'graficas2', component: Graficas2Component, data:{titulo:'GraficasAraña'}},

{path: 'promesas', component: PromesasComponent, data:{titulo:'Promesas'}},

{path: 'rxjs', component: RxjsComponent, data:{titulo:'Rxjs'}},

{path: 'account-settings', component: AccountSettingsComponent, data:{titulo:'Ajustes del Tema'} },

{path: '', redirectTo: '/dashboard', pathMatch:'full'},

{path: '\*\*', component: DashboardComponent}

Ahora vayamos al **breadcrumns.component.ts**: como podemos obtener esos parámetros que acabamos de definir en el pages.routes?:

Deberemos importar el modulo de Router

import { Router } from '@angular/router';

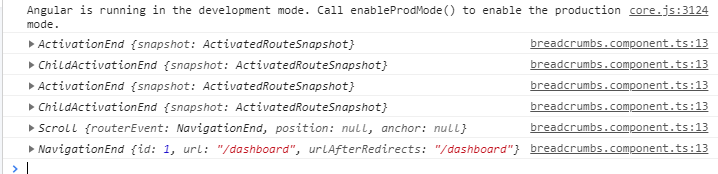
declaremos una variable publica router de tipo Router en los parámetros del constructor.

constructor( private router: Router) { }

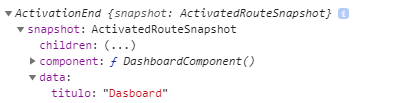
el Router tiene algo llamado **events** o eventos, esto es un Observable que puede regresar varias cosas.

1. Un RouterEvent, o un routerConfigurationLoadStart o RouterConfigLoadEnd etc.

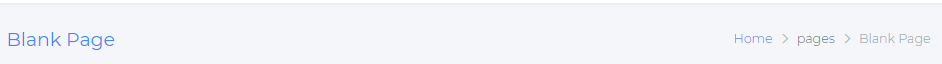
Suscribamosnos e imprimamosla en consola y veamos que optenemos:



Podemos observar todos los eventos que se disparan el que nos intereza es el **ActivationEnd** ya que en su elemento **snapshop** entontraremos el elemento **data** definido en el pages.routes.ts para esa pagina.



Lo que queremos es usar ese **titulo: “Dashboard”** y que nos reemplace el titulo de las paginas según sus rutas en el breadcrumbs es decir. Reemplazar el **BlanckPage** por el **titulo** de la **data**.



Lo primero es obtener la referencia a los **activationEnd**, ese sera el único que me interezara. Y eso lo podemos hacer de la siguiente manera

1. Importando operadores para trabajar con el metodo subscribe usando pipes
2. El primer operador sera el filter pilas que hay que importarlo

import { filter } from 'rxjs/operators';

con el definire una function que me regrese lo que a mi me intereza en este caso el **activationEnd** .

si el evento es una instancia de **activationEnd** eso lo hago con instanceof **activationEnd, eso solamente es lo que me va a dejar grabar,** también deberemos importar

import { Router, ActivationEnd } from '@angular/router';

export class BreadcrumbsComponent implements OnInit {

constructor( private router: Router) {

this.router.events.pipe(

filter(evento=> evento instanceof ActivationEnd)

)

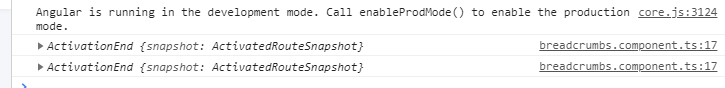
.subscribe(event=>{

console.log(event);

});

}

Esto me devolvera solo los eventos de tipo **activationEnd** donde encontraremos la data que necesitamos.



1. Ahora también filtremos por el tipo para ser mas específicos ya que el que tiene la propiedad firsChild null es el que nos sirve.

constructor( private router: Router) {

this.router.events.pipe(

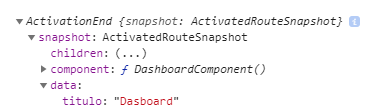
filter(evento=> evento instanceof ActivationEnd),

filter( (evento: ActivationEnd) => evento.snapshot.firstChild === null)

revisemos ahora



Y miremos si tenemos la data:



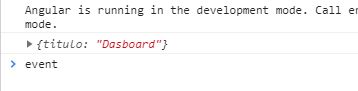
1. Ahora solo extraigamos la data debere pasarlo por el operador map que nos transforma los datos o simplemente nos ayuda a extraer de todo el elemento solo la data.
   1. Importemoslo primero:

import { filter, map } from 'rxjs/operators';

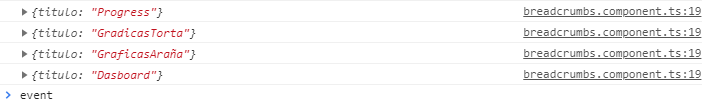
* 1. Y dentro regresaremos la información de la siguiente manera.

map((evento: ActivationEnd) => evento.snapshot.data)

1. El resultado de esto sera el elemento que yo necesdito:



1. Si empezamos a recorrer las paginas podemos observar que iremos obteniendo la información.



1. Lo único que falta es tomar el data o evento como le queramos poner en el subscribe y mostrarlo en el html pero antes de eso quitemos esa funciónalidad del constructor que es una ala practica y creemos una función propia llamada getDataRouter

getDataRoute(){

return this.router.events.pipe(

filter(evento=> evento instanceof ActivationEnd),

filter( (evento: ActivationEnd) => evento.snapshot.firstChild === null),

map((evento: ActivationEnd) => evento.snapshot.data)

)

}

1. Ahora llamémosla en el constructor:

constructor( private router: Router) {

this.getDataRoute()

.subscribe(event=>{

console.log(event);

});

}

1. Creemos una propiedad titulo de tipo string y digamos que en el constructor this.titulo sera igual a la data.titulo.

**public titulo :string;**

constructor( private router: Router) {

this.getDataRoute()

.subscribe(event=>{

console.log(event);

**this.titulo=event.titulo;**

1. Ahora intepolemos en el html

*<*div class="row page-titles"*>*

*<*div class="col-md-5 align-self-center"*>*

*<*h3 class="text-themecolor"*>*{{titulo}}*</*h3*>*

*</*div*>*

*<*div class="col-md-7 align-self-center"*>*

*<*ol class="breadcrumb"*>*

*<*li class="breadcrumb-item"*>*

*<*a href="javascript:void(0)"*>*Home*</*a*>*

*</*li*>*

*<*li class="breadcrumb-item"*>*pages*</*li*>*

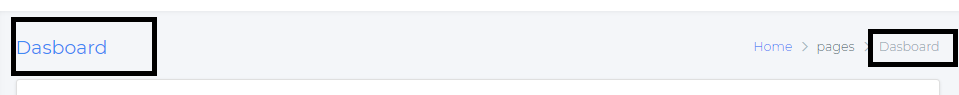
*<*li class="breadcrumb-item active"*>*{{titulo}}*</*li*>*

*</*ol*>*

*</*div*>*

*</*div*>*

1. El resultado sera el siguiente:



1. Ya esta funcional

Ahora para que en la pestaña de la pagina me aparesca el nombre de mi pagina actual utilizaremos una clase llamada title en la documentación de angular esta si la bsucamos.

<https://angular.io/api/platform-browser/Title>

se inyecta en el constructor y podemos llamar el metodo setTitle para colocarlo en la pestaña de la pagina. Hay que importar el modulo

import { Title } from '@angular/platform-browser';

constructor( private router: Router,

private title: Title) {

this.getDataRoute()

.subscribe(event=>{

console.log(event);

this.titulo=event.titulo;

this.title.setTitle(this.titulo);

});

}

Ahora si navegamos iremos cambiando el nombre de la pestaña al nombre de la pagina visitada.







Ahora para los metatags vamos ausar una clase de angular llamada meta lo importamos de

import { Title, Meta } from '@angular/platform-browser';

**Guardemos lo cambios en gitHub**

En este momento he alcanzado gracias a Dios la versión 1 para producción.

1:revisamos los cambios con **git status**.

2:agregamos esos cambios con **git add .** para incluirlos en el stagin area

3:creamos el commit con **git commit –m “sección 8 terminada breacrumbs”**

4:hagamos un **git push** para subirlo al repositorio.

5:Creemos un tag de git para algunas anotaciones como versiónes como a continuación.

**git tag –a v1.3.0 –m “Versión 1 – lista para produccion”**

6:para ver el tag usamos **git tag**.

7:para subirlo a github : **git push - -tags**



Backend server :

Esta sección tiene como objetivo preparar todo lo necesario para comenzar a crear nuestro backend server:

* Configuración de Mongo
* Robo 3T
* Conexión entre Mongo y Node
* Tip sobre colores de consola
* Aprender un poco sobre los errores de respuestas HTTP
* Configuración inicial de Express
* Establecer las bases de nuestros RESTful services

Creemos una carpeta llamada backend-server en el terminal vayamos a esa ruta y ejecutemos los comandos para inicialziar el proyecto.

* + 1. npm init pára crear el package.json
    2. instalemos express npm install express - - save

**creando el servidor express:**

creemos en la raiz el archive **app.js** este archivo sera el que inicialice express es decir el servidor, base de datos etc.

var express = require('express')

*//inicialziar variables*

var app = express();

*//escuchar peticiones*

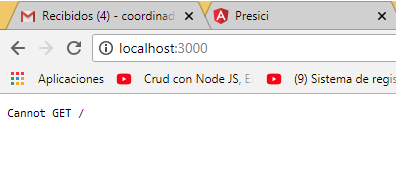
app.listen(3000, ()=>{

console.log("express server corriendo en pueto 3000 online");

});

Ejecutemos el comando **node app**.





Ya el servidor esta corriendo pero no encuentra rutas:

**Rutas:**

En app.js

*//rutas*

app.get('/', (req, res, next)=>{

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente"

})

})

Instalamos nodemon para no pasar recargando y guardando.

**npm install nodemon - -save-dev**

para usarlo debo ir al package.json y definamos un script para el nodemon de la siguiente manera.

{

"name": "backend-server",

"version": "1.0.0",

"description": "Servidor Backend para siciproyect",

"main": "index.js",

"scripts": {

**"start":"nodemon app.js",**

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "JESUS DARIO MARENCO PORTO",

"license": "ISC",

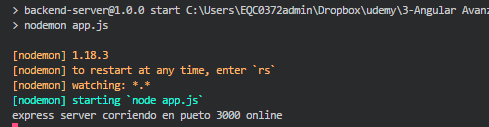
"dependencies": {

"express": "^4.16.3"

}

}

Ahora bajamos el servidor y escribimos **npm start** y podremos observar que nodemon esta corriendo:



**Instalemos mongodb y robo3t**

Ya tengo todo esto instalado

**Establecer conexión del servidor a mongodb**

Debemos instalar la librería mongoose

**Npm install mongosse - - save**

Ahora en app.js requeriré la librería y hare lo siguiente.

var mongoose = require('mongoose');

*//conexion a la db*

mongoose.connection.openUri('mongodb://localhost:27017/sici2', (err, res)=>{

if(err) throw err;

console.log("base de datos online");

});

El resultado sera



**Realicemos el backub en gitHub**

1. creemos un nuevo repositrio llamado backend-sici2
2. inicialicemos el repositorio con **git init**.
3. Para ver los cambios **git status**.
4. Creemos el archivo **.gitignore** para configurar que los modulos de node es decir el node\_modules no se suban

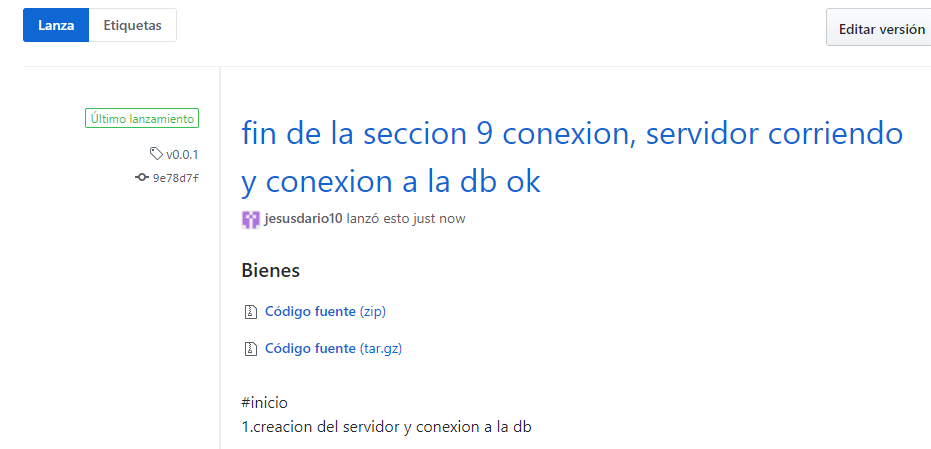
Allí escribiremos lo siguiente

node\_modules/

1. Al hacer nuevamente **git status** podremos observar que excluyo los modulos de node.
2. Agregremos los archivos al staigin área, **git add .**
3. Primer commit git commit –m “conexión al servidor y a la base de datos”
4. Agregar el repositorio remoto tenemos que ver la dirección en github

**git remote add origin https://github.com/jesusdario10/backend-sici2.git**

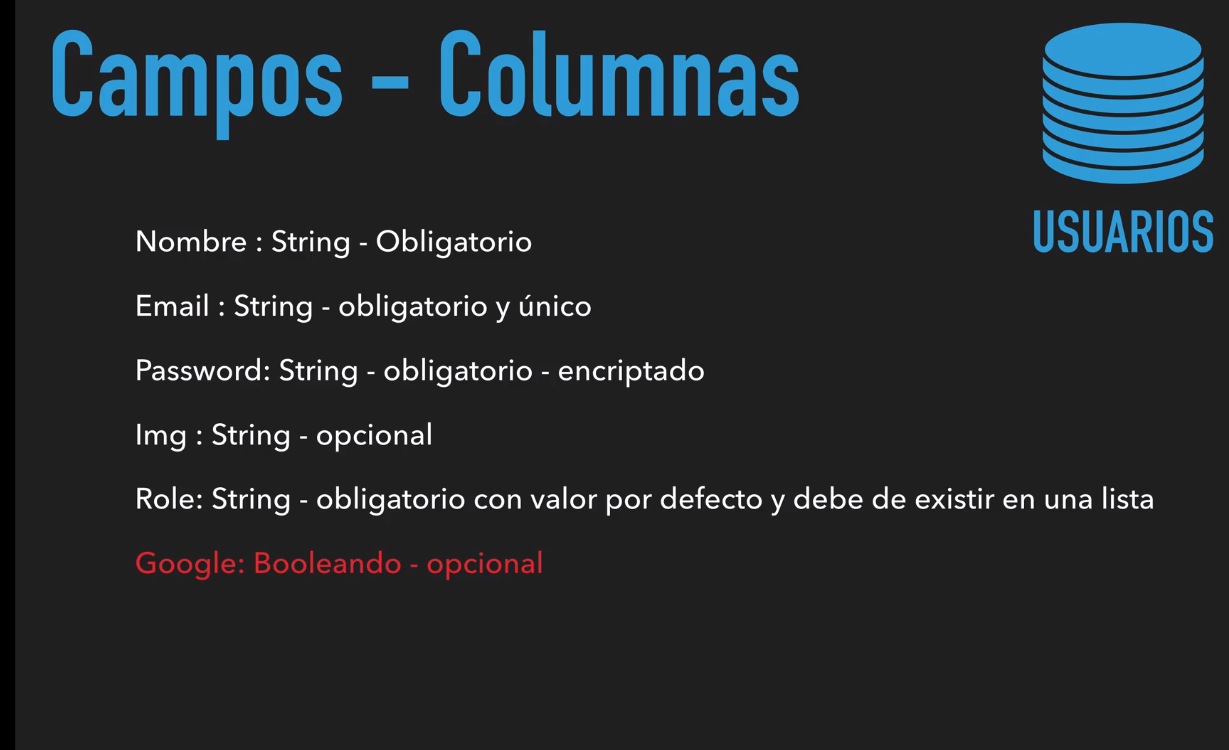
1. Subir al repositorio los archivo **git push origin master**.
2. Creemos el primer tag con **git tag -a v0.0.1 -m "fin de la seccion 9 conexion, server corriendo y conexion a la db ok"**
3. Revisamos con **git tag**.
4. Subamos el tag a github **git push - -tags**.



Esta sección tiene por objetivo trabajar fuertemente con Express, Mongo y Node:

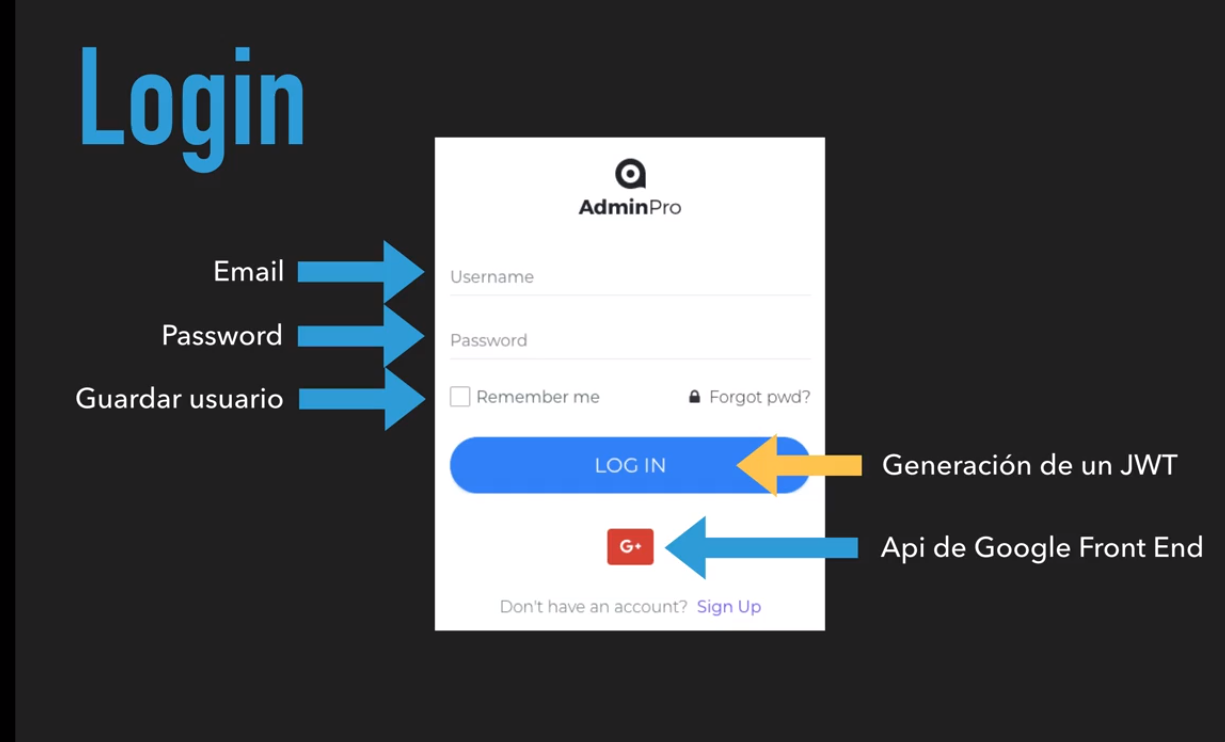
* Explicación general y alcances del Backend
* Crear colección de Usuarios
* Crear modelos en Node
* POST
* Validaciones automáticas mediante Mongoose
* Encriptar contraseñas
* PUT
* Delete
* Get
* Login de Usuario
* Generación de un JWT
* Middlewares
* Optimizar Middlewares













**Crear colección de usuarios:**

**Abramos el robo3T:**

Creemos una colección llamada usuarios

{

"nombre":"Jesus",

"email":"titan@gmail.com",

"password":"123",

"img":null,

"role":"ADMIN\_ROLE"

}

**Ahora Creemos el Modelo de usuarios en node quie nos permitirá controlar esta colección.**

Reemos el folder de modelos que se llamara **models**.

Allí dentro creare un archivo js llamado usuarioModel.js

var mongoose = require('mongoose');

var Schema = mongoose.Schema();

var usuarioSchema = new Schema({

nombre: {type:String, required: [true, "El nombre es requerido"]},

correo: {type:String, unique:true, required: [true, "El correo es requerido"]},

password: {type:String, required: [true, "La contraseña es necesaria"]},

img: {type:String, required: false},

role: {type:String, required: true, default:"USER\_ROLE"}

});

module.exports = mongoose.model('Usuario', usuarioSchema);

**definamos las rutas para los servicios rest del usuario.**

Levantemos el servidor

1-Creemos una carpeta en la raiz para las ruta llamada routes:

2-Dentro creemos un archivo llamado appRoutes.js que serán las rutas principales.

3-Cortemos la ruta definia en el app.js y peguemosla aquí.

4-requerda importar express e inicializarlo.

5-exportemos ese modulo.

var express = require('express');

*//inicialziar variables*

var app = express();

*//rutas*

app.get('/', (req, res, next)=>{

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente"

})

});

module.exports=app;

6-deberemos importar el modulo appRoutes en el app.js principal para que las rutas puedan funcionar y también deberemos declarar el middleware que lo activa.

var appRoutes = require('./routes/appRoutes');

*//middleware para las rutas*

app.use('/', appRoutes);

si probamos con el postman a la siguiente ruta una petición de tipo get obtendremos lo esperado.

<http://localhost:3000>

{

"ok": true,

"mensaje": "peticion realizada correctamente"

}

**Creemos una ruta para los usuarios ahora**

Creemos un archivo en la carpeta routes llamado usuarioRoutes.js aquí usaremos la importación del modelo de usuarios que creamos y con el metodo find de mongosse realizaremos una consulta que nos muestre todos los usuarios

var express = require('express');

*//inicialziar variables*

var app = express();

*//rutas*

app.get('/usuarios', (req, res, next)=>{

res.status(200).json({

ok:true,

mensaje:"get usuarios"

})

});

module.exports=app;

deberemos crear el middleware para que la ruta funcione. En el app.js

var express = require('express');

var mongoose = require('mongoose');

var appRoutes = require('./routes/appRoutes');

var UsuarioRoutes = require('./routes/usuarioRoutes.js');

*//inicialziar variables*

var app = express();

*//conexion a la db*

mongoose.connection.openUri('mongodb://localhost:27017/sici2', (err, res)=>{

if(err) throw err;

console.log("base de datos online");

});

*//middleware para las rutas*

app.use('/', UsuarioRoutes);

app.use('/', appRoutes);

*//escuchar peticiones*

app.listen(3000, ()=>{

console.log("express server corriendo en pueto 3000 online");

});

Ahora en las rutas de usuarios es decir usuarioRoutes.js haremos la consulta a la base de datos por todos los usuarios que tenemos pero de solo los campos **nombre, email, img, role**.

var express = require('express');

var Usuario = require('../models/usuarioModel');

*//inicialziar variables*

var app = express();

*//rutas*

app.get('/usuarios', (req, res, next)=>{

Usuario.find({},'nombre, email, img, role')

.exec((err, usuario)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

usuarios:usuario

});

});

});

module.exports=app;

Al hacer la petición get a la ruta <http://localhost:3000/usuarios> en el postman. Este es el resultado.

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

  {

"role": "ADMIN\_ROLE",

"\_id": "5b64e8d601fa9dd5aa130ef0",

"nombre": "Jesus",

"email": "titan@gmail.com",

"img": null

}

],

}

**Definamos la ruta para crear un nuevo usuario:**

Lo primero es instalar body-parser que es el middleware que nos ayuda a parsear todos los datos que nos vengan en formatojson y viceversa

1-npm instal - -save

2-en el app.js importaremos el body-parser

var bodyParser = require('body-parser;')

3-configurar el bodyParser

*//configurando el body parser*

*// parse application/x-www-form-urlencoded*

app.use(bodyParser.urlencoded({ extended: false }))

*// parse application/json*

app.use(bodyParser.json())

4-ahora en el usuarioRoutes.js creemos el metodo post para agregar usuarios.

*//post usuario*

app.post('/', (req, res, next)=>{

var loquesea = req.body

res.status(200).send({

loquesea: loquesea,

ok:true

});

});

Ahora mandemos un dato desde el postman para ver si capturan en el body

{

"loquesea": {

"parametro1": "sisenor"

},

"ok": true

}

Efectivamente capturo, ahora si capturemos el usuario

**Ahora guardemoslo en la base de datos:**

Enviemos los datos requeridos desde el postman y ejecutemos la peticion

*//post usuario*

app.post('/', (req, res, next)=>{

var body = req.body

var usuario = new Usuario({

nombre : body.nombre,

correo: body.correo,

password: body.password,

img : body.img,

role : body.role

});

usuario.save((err, usuarioGuardado)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(201).send({

usuario: usuarioGuardado,

ok:true

});

});

});

{

"usuario": {

"role": "USER\_ROLE",

"\_id": "5b6511ebcd91471a44db7427",

"nombre": "test2",

"correo": "test2@test2.com",

"password": "123456",

"\_\_v": 0

},

"ok": true

}

Validaciones adicionales para guardar documentos. En este caso definamos cuales son los roles validos que va a aceptar la aplicación.

En el modelo de usuarios haremos lo siguiente.

var rolesValidos = {

values : ['ADMIN\_ROLE', 'USER\_ROLE'],

message:'{VALUE} no es un rol permitido'

}

Y lo añadimos al modelo conla propiedad enum

role: {type:String, default:"USER\_ROLE", enum: rolesValidos}

**Encriptacion de las contraseñas en una sola via**

Para ello utilizaremos una librería llamada **bcryptjs** de modo que la instalaremos con el siguiente comando **npm install bcryptjs - - save**.

Esta librería debería importarla donde tengo el escrita la petición post para la creación del usuario en este caso es en usuarioRoutes.js.

var bcrypt = require('bcryptjs');

Para eso **bcryptjs** tiene lo que es el **hastSync** que recibe el parámetro que deseamos encriptar y un **salt**.

Entonces el post quedaría

*//post crear usuario*

app.post('/', (req, res, next)=>{

var body = req.body

var usuario = new Usuario({

nombre : body.nombre,

correo: body.correo,

password: bcrypt.hashSync(body.password, 10),

img : body.img,

role : body.role

});

El resultado sera el documento con password encriptado en una sola via

{

"usuario": {

"role": "ADMIN\_ROLE",

"\_id": "5b65166d21c17424388c7192",

"nombre": "test1",

"correo": "test1@test1.com",

"password": **"$2a$10$K7VCh9Q0ufpCwps60g6eSeVmuqsQhvkhxHad9VaVkwYzUFjiN41Vu",**

"\_\_v": 0

},

"ok": true

}

**Creemos un metodo para actualziar usuarios:**

*//put actualziar usuarios*

app.put('/:id', (req, res)=>{

var id = req.params.id;

var body = req.body;

Usuario.findById(id, (err, usuario)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

if(!usuario){

res.status(400).json({

ok:false,

mensaje:"no existe el usuario",

});

}

*//si si existe*

usuario.nombre = body.nombre;

usuario.correo = body.correo;

usuario.role = body.role;

usuario.save( (err, usuarioActualizado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"Error al actualizar usuario",

errors:err

});

}

usuarioActualizado.password =':)';

if(usuarioActualizado){

res.status(201).send({

usuario: usuarioActualizado,

ok:true

});

}

});

});

})

El resultado al aplicar la ruta seria:

<http://localhost:3000/usuarios/5b65166d21c17424388c7192>

{

"usuario": {

"role": "ADMIN\_ROLE",

"\_id": "5b65166d21c17424388c7192",

"nombre": "Nuevo test",

"correo": "test1@test1.com",

"password": ":)",

"\_\_v": 0

},

"ok": true

}

Metodo borrar usuario

*//delete usuario*

app.delete('/:id', (req, res)=>{

var id = req.params.id;

Usuario.findByIdAndRemove(id, (err, usuarioBorrado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudo borrar usuario",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

usuarios:usuarioBorrado

});

});

});

Y esta sera la respuesta al aplicar la ruta

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": {

"role": "ADMIN\_ROLE",

"\_id": "5b651e2c458f8c242cf4e602",

"nombre": "Nuevo test",

"correo": "test2@test2.com",

"password": "$2a$10$QumVKOVl9jpW02Gz6vMgT.WypXN8sl7Y8Uvf5GgogZ89xdKP.U1Ja",

"\_\_v": 0

}

}

Efectivamente el usuario se ha borrado.

**Login de usuario:**

Usaremos el correo electrónico y el password como métodos de autenticación.

El primer problema es que el password esta encriptado.

Para esto:

1-creemos una ruta para los login es decir un archivo loginRoutes.js en nuestra carpeta de routes.

2-usaremos el express, el bcrypt para confirmar si la contraseña ingresada hace mach con la que tenemos en la base de datos, y también importaremos el modelo del usuario.

var express = require('express');

var bcrypt = require('bcryptjs');

var Usuario = require('../models/usuarioModel');

var app = express();

3-exportemos este modulo desde el inicio

module.exports = app;

4-inicialicemos el middleware de la ruta del login en el app.js

var LoginRoutes = require('./routes/loginRoutes');

*//middleware para las rutas*

app.use('/login', LoginRoutes);

5-empecemos a crear el metodo de login: y probemos

*//metodo de login*

app.post('/', (req, res)=>{

res.status(200).json({

ok:true,

mensaje:"el mensaje"

})

})

**6-para continuar necesitamos recibir el correo y la conrtaseña como parte del cuerpo del login pero para que sea mas seguro usaremos JWT mas conosido como JSON WEB TOKENS.**

Una ves nosotros identificamos un login correcto deberíamos de regresar un token, ese token adicionalmente deberíamos de enviarlo a todas las peticiones que requieran autenticación. Pero iniciemos con la creación del token.

Existe una librería que se llama jsonwebtoken por lo general la podemos encontrar en github

<https://github.com/auth0/node-jsonwebtoken>

la instalación sera **npm instal jsonwebtoken - - save**

en la documentoacion podremos encontrar como crearlos como poner la vigencia etc.

Vayamos al **loginRoutes.js** y creemos un metodo para generar los token, también deberemos importar la librería anteriormente instalada.

var jwt = require('jsonwebtoken');

creemos el metodo de token en el post de login

1-creamos al variable **token** que igualamos al **jwt** que usa el metodo **sign** que **es como decir firmar** este metodo recibe una serie de parámetros. **El primero** sera mi usaurio de base de datos,  **el segundo parámetro** es algo que se conoce como el **seed** o semilla es un conjunto de caracteres únicos que definiré para hacer mas seguro mi token, **el tercer parámetro** es la fecha de expiración del token, podemos hacerla infinita, que venza en una hora etc. En este caso démosle una fecha de vigencia de 4 horas.

Antes quitemos la contraseña del usuario db para no mandrla en el token

usuarioDB.password=":)";

var token = jwt.sign({usuario:usuarioDB}, 'este-es-un-seed-dificil', {expiresIn:14400});*//4 horas*

con esto ya tenemos nuestro token, enviemoslo para revisar en el postman si

jecutamos el postman en la dirección <http://localhost:3000/login> y efectivamente ya teneos nuestro token.

{

"ok": true,

"body": {

"correo": "test1@test1.com",

"password": "123456"

},

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3VhcmlvIjp7InJvbGUiOiJBRE1JTl9ST0xFIiwiX2lkIjoiNWI2NTE2NmQyMWMxNzQyNDM4OGM3MTkyIiwibm9tYnJlIjoiTnVldm8gdGVzdCIsImNvcnJlbyI6InRlc3QxQHRlc3QxLmNvbSIsInBhc3N3b3JkIjoiOikiLCJfX3YiOjB9LCJpYXQiOjE1MzM0MDAzNzcsImV4cCI6MTUzMzQxNDc3N30.Az\_76N55omDMPgj4W5SA6JshW\_GqPD79t86cx6di5ho"

}

Vayamos a la la pagina de los jsonwebtoken que es <https://jwt.io/> con ese jwt que nosotros acabamos de generar podemos pegarlo aca. En el espacio en verde



Como podemos observar al lado izquierdo la parte roja es el **header**, lo morado es el **payload** que es done esta la data y lo que se encuentra en azul es la firma que debemos usar para poder validar el token.

Enconces pequemos nuestro token y analicemos.

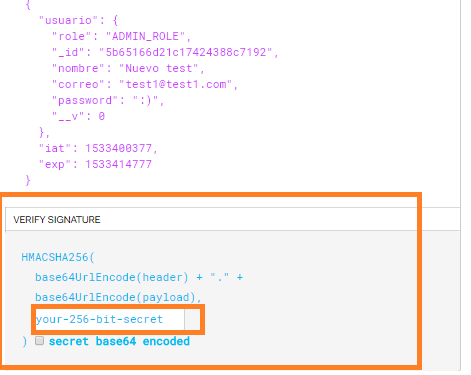


Podemos observar que en la imagen anterior encontramos toda nuestra **Data** pero **la firma no es valida.** Es decir este token no lo deberíamos usar, **(recordemos que los token los debemos usar del lado del servidor no del lado del frontend) toda la data que recibamos de cualquier aplicación de javascript tenemos uqe valdiarla en el backend.**

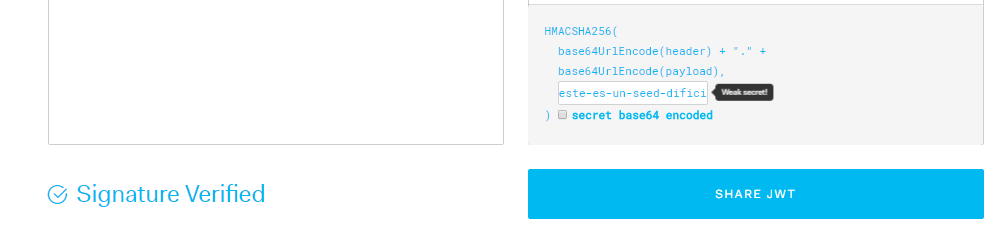
Si un token no es valido deberemos rechararlo

En esta parte nos pedirá que validemos el **seed**. En nuestro caso el seed que definimo es @este-es-un-seed-dificil

Lo copiamos y lo pegamos alla



Al nosotros ingresar nuestro seed inmediatamente se valida nuestro token



Del lado de angular nosotrs almacenaremos el token en el localStorage, nos aseguraremos que las peticiones lo lleven pero por el momento esto es lo que queríamos hacer.

**Con la generación de nuestro token ya tenemos todo lo necesario para poder bloquear peticiones como el post para crear usuarios, actualiazar usuarios y borrar usuarios. Es decir lo que haremos ahora es que las peticiones de crear, actualziar y borrar, tengan una validación para que revice el token y si no es valido o ya expiro que no pueda hacer nada esa pecicion.**

**Hay 2 formas de hacerlo:**

Ahora como recibimos el token parq aprocesarlo?

**1**-lo mas fácil es crearnos una carpeta de config y un archivo dentro de config.js donde yo tenga todas las constantes para usarlas. Esto con el fin de definir el SEED.

module.exports.SEED ='este-es-un-seed-dificil'

**2-** requerimos este modulo en el usuariosRoutes.js para poder usar esa constante. De esta forma accedo a la variabe inmendiatamente con la notación de punto.

var SEED = require('../config/config').SEED;

**3-**requiramos este modulo en el login y reemplacemos la cadena por la variable.

*//crear un token*

usuarioDB.password=":)";

var token = jwt.sign({usuario:usuarioDB}, SEED, {expiresIn:14400});*//4 horas*

**4-**vayamos al usuarioRoutes.js, importemos la constante SEED y el jwt.

**3.** vayamos al usuariosRoutesjs. Y después del metodo getUsuario y antes de el de actualziar usuario, creemos un middleware que verificara nuestros token, es decir este middleware leera el token que revicibira desde la url, procesarlo, ver si es valido, ver sino ha expirado y si funciona y si es valido deje continuar, por esa razón creamos el middleware primero que las peticiones siguientes porque del middleware en adelante todo requerirá de autnticacion por el middleware por token.

**La primera**:

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

app.use('/', (req, res, next)={

})

**3.**creamos una variable llamada token que sera igual al req.query.token;

**4**.ya teniendo el token necesitamos verificar que sea valido, para eso deberemos hacer el mismo proceso que utilizamos en la generación del token. Es decir usando el jwt y el metodo verify que recibirá 3 parametros**, el priemro** el token, **el segundo el SEED** y **el tercero un callback** que en su primer parámetro tendrá un **error** y en su **segundo parámetro un** **decoded** que contendrá la información del usuario porque eso es lo que viene por el payload es decir la data**,**

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

app.use('/', (req, res, next)=>{

var token = req.query.token;

jwt.verify( token, SEED, (err, decoded)=>{

if(err){

res.status(401).json({

ok:false,

mensaje:"token incorrecto",

errors:err

});

}

});

})

Si yo lo dejo asi cuando envie por ejempl una petición put inmediatamente me dira token incorrecto

**{**

**"ok": false,**

**"mensaje": "no se pudieron traer los datos",**

**"errors": {**

**"name": "JsonWebTokenError",**

**"message": "jwt must be provided"**

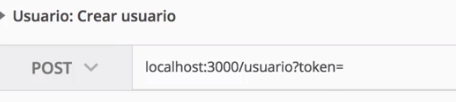
**}**

**}**

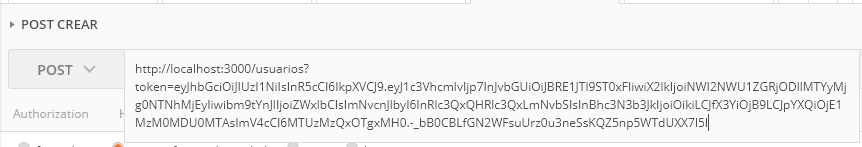
Si lo hago con un crear nisiquiera me hara la verificación de usuario o correos distintos porque como el token es incorrecto no me deja llegar alla.

**Como recibimos el token Para que valide?**

En la url deberemos enviarlo como parámetro opcional es decir:



Y después del = añadir el token que quede de la siguiente manera.



Luego antes de ejecutar la petición usaremos el parámetro next() para que siga con la ejecución

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

app.use('/', (req, res, next)=>{

var token = req.query.token;

jwt.verify( token, SEED, (err, decoded)=>{

if(err){

res.status(401).json({

ok:false,

mensaje:"token incorrecto",

errors:err

});

}

**next();**

});

})

Ahora si ejecutemos el postman

**{**

**"usuario": {**

**"role": "ADMIN\_ROLE",**

**"\_id": "5b65e989690f391af8cf572b",**

**"nombre": "test4@test4.com",**

**"correo": "test4@test4.com",**

**"password": "$2a$10$72eyBL4GnNlg3iZme65gUe8cY9E5EypbXqfDv4SrLyWDdn1e9C6PS",**

**"\_\_v": 0**

**},**

**"ok": true**

**}**

**Podremos crear el usuario sin problemas.**

**Esta forma no ofrece mucha flexibilidad hasi que hagamoslo por otro lado.**

Movamos todo el midlleware de token a un lugar diferente es decir creemos una nueva carpeta en la raiz que se llame middlewares y allí dentro un archivo llamado autenticación.js donde pegaremos todo el middleware creado en usuariosroutes. Recuerda borrarlo de alla.

También debemos importar el jsonwebtoken y el SEED. Por lo pronto debería quedar asi.

var SEED = require('../config/config').SEED;

var jwt = require('jsonwebtoken');

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

app.use('/', (req, res, next)=>{

var token = req.query.token;

jwt.verify( token, SEED, (err, decoded)=>{

if(err){

res.status(401).json({

ok:false,

mensaje:"token incorrecto",

errors:err

});

}

next();

});

})

Ahora reemos una función que exportaremos que contenga req, res, next y modificaremos la función de la siuiente manera de modo que no tengamos que usar el app que es express.

var SEED = require('../config/config').SEED;

var jwt = require('jsonwebtoken');

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

exports.verificarToken = function(req, res, next){

var token = req.query.token;

jwt.verify( token, SEED, (err, decoded)=>{

if(err){

res.status(401).json({

ok:false,

mensaje:"token incorrecto",

errors:err

});

}

*//next();*

res.status(200).json({

ok:true,

decoded:decoded

});

});

}

Para poder usar esta función debere importarla en **usuariosRoutes**.js

var mdAutenticacion = require('../middlewares/autenticacion');

ahora en cada función o petición que quiera que use el verificar token debere llamar esa función como segundo parámetro después de la ruta. Es decir por ejemplo empecemos con el crear.

*//post crear usuario*

app.post('/', mdAutenticacion.verificarToken, (req, res, next)=>{

var body = req.body

var usuario = new Usuario({

nombre : body.nombre,

correo: body.correo,

password: bcrypt.hashSync(body.password, 10),

img : body.img,

role : body.role

});

Si yo hago send de esto me traerá este resultado pero esto aun no sera un usuario creado, es lo que me trae el **decoded**.

{

"ok": true,

"decoded": {

"usuario": {

"role": "ADMIN\_ROLE",

"\_id": "5b65e5ddc89e16228453a212",

"nombre": "elel",

"correo": "test1@test1.com",

"password": ":)",

"\_\_v": 0

},

"iat": 1533408152,

"exp": 1533422552

}

}

Entonces para grabarlo debere hacer un cambio en el middleware de autenticacion que sera el siguiente.

var SEED = require('../config/config').SEED;

var jwt = require('jsonwebtoken');

*// ======================================*

*// Middleware para verificar por token*

*// ======================================*

exports.verificarToken = function(req, res, next){

var token = req.query.token;

jwt.verify( token, SEED, (err, decoded)=>{

if(err){

res.status(401).json({

ok:false,

mensaje:"token incorrecto",

errors:err

});

}

if(decoded){

req.usuario = decoded.usuario;

next();

}

});

}

Y en la ruta cambiar la respuesta: enviare el usuario token que es el usuario que hizo la solicitud de creación lo enviare en el req.usuario para capturarlo en el crear usuario y saber quien fue el usuario que creo al nuevo usuario.

*//post crear usuario*

app.post('/', mdAutenticacion.verificarToken, (req, res, next)=>{

var body = req.body

var usuario = new Usuario({

nombre : body.nombre,

correo: body.correo,

password: bcrypt.hashSync(body.password, 10),

img : body.img,

role : body.role

});

usuario.save((err, usuarioGuardado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(201).send({

usuarioGuardado :usuarioGuardado,

usuarioToken: req.usuario,

ok:true

});

});

});

Para finalizar copiemos la función de verificartoken en las peticiones put, detele, update menos en la get.

**Hagamos un backup en github y creemos un tag.**

**Git add .**

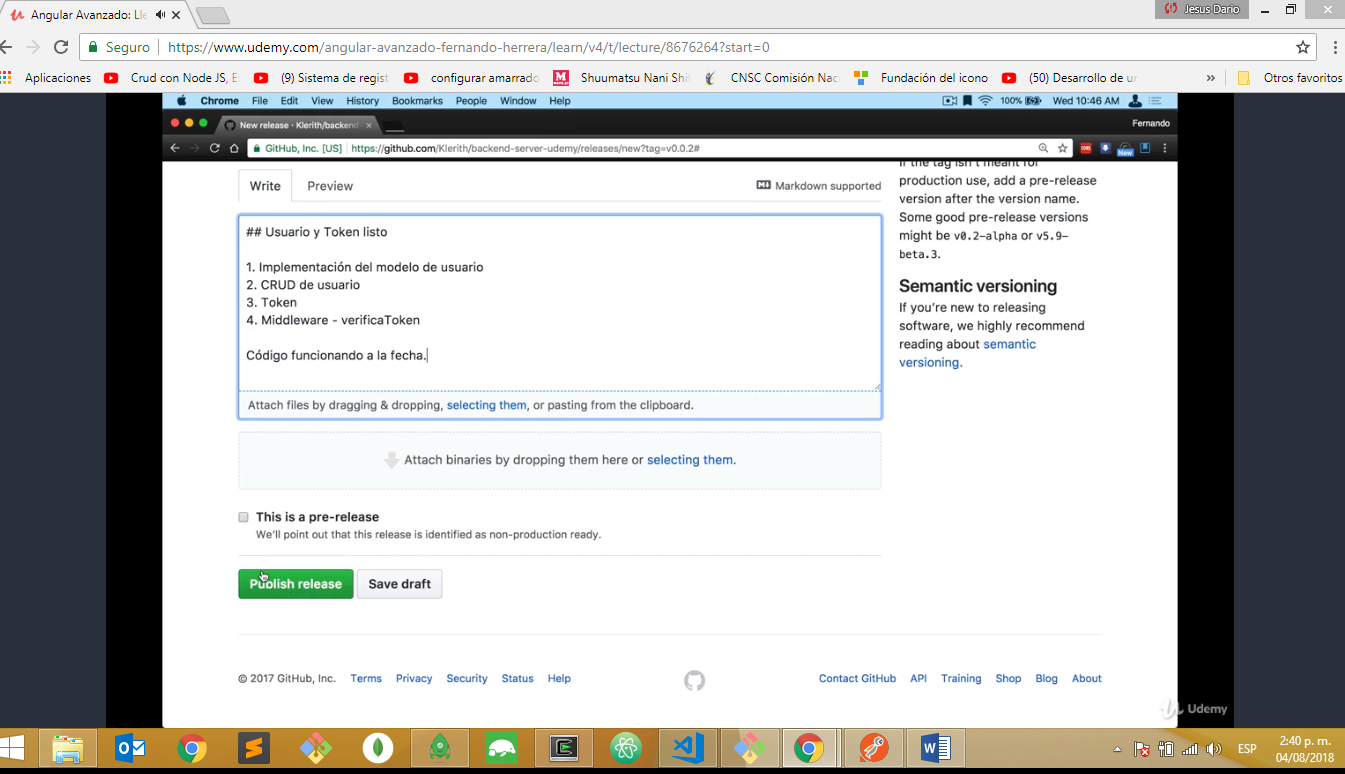
**Git commit –m “usuario y token listo”**

**Git push origin master**

**Git tag**

**Git tag -a v0.0.2 -m "login y usuarios listo seccion2"**

**Git push tag - -tags**

****

Esta sección tiene varias tareas e información importante sobre:

* CRUD de médicos y hospitales
* Una tarea robusta con su respectivo documento y resolución
* Códigos útiles de Mongoose
* Populate
* Gets
* Paginar resultados
* Búsquedas específicas y globales
* Subida de archivos al servidor
* Asignación de imagen a un hospital, medico o usuario
* Formas de exponer archivos a la web

**Creemos crud y autenticación por token de hospitales.**

**App.js**

var express = require('express');

var mongoose = require('mongoose');

var appRoutes = require('./routes/appRoutes');

var UsuarioRoutes = require('./routes/usuarioRoutes.js');

var LoginRoutes = require('./routes/loginRoutes');

var hospitalesRoutes = require('./routes/hospitalesRoutes');

var itemSolicitudesRoutes = require('./routes/itemSolicitudesRoutes');

var bodyParser = require('body-parser');

*//inicialziar variables*

var app = express();

*// parse application/x-www-form-urlencoded //configurando el body parser*

app.use(bodyParser.urlencoded({extended:false}));

app.use(bodyParser.json());

*//conexion a la db*

mongoose.connection.openUri('mongodb://localhost:27017/sici2', (err, res)=>{

if(err) throw err;

console.log("base de datos online");

});

*//middleware para las rutas*

app.use('/login', LoginRoutes);

app.use('/', UsuarioRoutes);

app.use('/', hospitalesRoutes);

app.use('/', itemSolicitudesRoutes);

app.use('/', appRoutes);

*//escuchar peticiones*

app.listen(3000, ()=>{

console.log("express server corriendo en pueto 3000 online");

});

**hospitalModels.js**

var mongoose =require('mongoose');

var Schema =mongoose.Schema;

var hospitalSchema =new Schema({

        nombre: {type: String,  required: [true, 'El    nombre  es  necesario'] },

        img: {type: String, required: false },

        usuario: {type: Schema.Types.ObjectId,  ref: 'Usuario' }

},  {   collection: 'hospitales' });

module.exports = mongoose.model('Hospital', hospitalSchema);

**Hospital.Routes.js**

var express = require('express');

var bcrypt = require('bcryptjs');

var Hospitales = require('../models/hospitalesModels');

var mdAutenticacion = require('../middlewares/autenticacion');

*//inicialziar variables*

var app = express();

*//get usuarios*

app.get('/hospitales', (req, res, next)=>{

Hospitales.find({})

.exec((err, usuario)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

usuarios:usuario

});

});

});

*//post crear usuario*

app.post('/hospitales', mdAutenticacion.verificarToken, (req, res, next)=>{

var body = req.body

var hospital = new Hospitales({

nombre : body.nombre,

img: null,

usuario: req.usuario.\_id

});

hospital.save((err, hospitalGuardado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(201).send({

usuarioGuardado :hospitalGuardado,

usuarioToken: req.usuario,

ok:true

});

});

});

*//put actualizar usuarios*

app.put('/hospitales/:id', mdAutenticacion.verificarToken, (req, res)=>{

var id = req.params.id;

var body = req.body;

Hospitales.findById(id, (err, hospital)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

if(!hospital){

res.status(400).json({

ok:false,

mensaje:"no existe el usuario",

});

}

*//si si existe*

hospital.nombre = body.nombre;

hospital.usuario = req.usuario.\_id;

hospital.img = body.img;

hospital.save( (err, hospitalActualizado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"Error al actualizar hospital",

errors:err

});

}

if(hospitalActualizado){

res.status(201).send({

hospital: hospitalActualizado,

ok:true

});

}

});

});

});

*//delete usuario*

app.delete('/hospitales/:id', mdAutenticacion.verificarToken, (req, res)=>{

var id = req.params.id;

Hospitales.findByIdAndRemove(id, (err, hospitalBorrado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudo borrar hospital",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

hospital:hospitalBorrado

});

});

});

module.exports=app;

**Creemos crud y autenticación por token de medicos.**

**MédicosModels.js**

var mongoose =require('mongoose');

var Schema =mongoose.Schema;

var hospitalSchema =new Schema({

        nombre: {type: String,  required: [true, 'El    nombre  es  necesario'] },

        img: {type: String, required: false },

        usuario: {type: Schema.Types.ObjectId,  ref: 'Usuario' }

},  {   collection: 'hospitales' });

module.exports = mongoose.model('Hospital', hospitalSchema);

**app.js**

var express = require('express');

var mongoose = require('mongoose');

var appRoutes = require('./routes/appRoutes');

var UsuarioRoutes = require('./routes/usuarioRoutes.js');

var LoginRoutes = require('./routes/loginRoutes');

var hospitalesRoutes = require('./routes/hospitalesRoutes');

var medicosRoutes = require('./routes/medicoRoutes');

var itemSolicitudesRoutes = require('./routes/itemSolicitudesRoutes');

var bodyParser = require('body-parser');

*//inicialziar variables*

var app = express();

*// parse application/x-www-form-urlencoded //configurando el body parser*

app.use(bodyParser.urlencoded({extended:false}));

app.use(bodyParser.json());

*//conexion a la db*

mongoose.connection.openUri('mongodb://localhost:27017/sici2', (err, res)=>{

if(err) throw err;

console.log("base de datos online");

});

*//middleware para las rutas*

app.use('/login', LoginRoutes);

app.use('/', UsuarioRoutes);

app.use('/', hospitalesRoutes);

app.use('/', medicosRoutes);

app.use('/', itemSolicitudesRoutes);

app.use('/', appRoutes);

*//escuchar peticiones*

app.listen(3000, ()=>{

console.log("express server corriendo en pueto 3000 online");

});

**MedicoRoutes.js**

var express = require('express');

var Medico = require('../models/medicosModels');

var mdAutenticacion = require('../middlewares/autenticacion');

*//inicialziar variables*

var app = express();

app.get('/medicoprueba', (req, res, next)=>{

res.status(200).json({

ok:true,

mensaje:"ruta correcta"

});

});

*//get usuarios*

app.get('/medico', (req, res, next)=>{

Medico.find({})

.exec((err, medicos)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

medicos:medicos

});

});

});

*//post crear usuario*

app.post('/medico', mdAutenticacion.verificarToken, (req, res, next)=>{

var body = req.body

var medico = new Medico({

nombre : body.nombre,

usuario: req.usuario.\_id,

hospital: body.hospital*//este campo lo recibiremos de la peticion post o put de angular*

});

medico.save((err, medicoGuardado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(201).send({

medicoGuardado :medicoGuardado,

usuarioToken: req.usuario,

ok:true

});

});

});

*//put actualizar usuarios*

app.put('/medico/:id', mdAutenticacion.verificarToken, (req, res)=>{

var id = req.params.id;

var body = req.body;

Medico.findById(id, (err, medico)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

if(!medico){

res.status(400).json({

ok:false,

mensaje:"no existe el usuario",

});

}

*//si si existe*

medico.nombre = body.nombre;

medico.usuario = req.usuario.\_id;

medico.hospital = body.hospital;*//este campo lo recibiremos de la peticion post o put de angular*

medico.save( (err, medicoActualizado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"Error al actualizar medico",

errors:err

});

}

if(medicoActualizado){

res.status(201).send({

medico: medicoActualizado,

ok:true

});

}

});

});

});

*//delete usuario*

app.delete('/medico/:id', mdAutenticacion.verificarToken, (req, res)=>{

var id = req.params.id;

Medico.findByIdAndRemove(id, (err, medicoBorrado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudo borrar medico",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

medico:medicoBorrado

});

});

});

module.exports=app;

Ahora obtengamos os datos de las relaciones de los documentos en la base de datos. Empecemos con los hospitales:

**Como hacemos para que en ves del id del usuario que lo creo me aparezcan todos sus datos**?

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

{

"\_id": "5b67003e9c87b9e098367517",

"nombre": "hospital1",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0"

},

}

**Esto lo haremos con una función de mongoose llamada populate**, allí nosotros vamos a especificar de que campo relacionado queremos que traigan los datos y si queremos también especificaremos los datos que queremos que traiga por sino queremos que los traiga todos, en, como **primer argumento** mandaremos **el nombre de la campo relacionado** y como **segundo argumento** enviaremos **los campos que deseamos que traiga de esa relación**.

*//get Hospitales*

app.get('/hospitales', (req, res, next)=>{

Hospitales.find({})

.populate('usuario', 'nombre correo')

.exec((err, usuario)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

usuarios:usuario

});

});

});

El resultado sera el siguiente: nos traerá los datos de la relación y solo los que definimos podemos observar que no nos trae el password ni el role.

**{**

**"\_id": "5b6700a4d6cb4420c0c30c98",**

**"nombre": "hospital3",**

**"img": null,**

**"usuario": {**

**"\_id": "5b662d5ff01576979ff4aab0",**

**"nombre": "qqqq",**

**"correo": "test1@test.com"**

**},**

**"\_\_v": 0**

**}**

**Ahora hagamoslo con los médicos este tiene un componente mas, porque en teoría tiene 2 relaciones, una entre usuariso y otra con hospitales lo que deberemos es simplemente hacer un nuevo populate:**

*//get usuarios*

app.get('/medico', (req, res, next)=>{

Medico.find({})

.populate('usuario', 'nombre correo')

.populate('hospital')

.exec((err, medicos)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

medicos:medicos

});

});

});

**"medicos": [**

**{**

**"\_id": "5b6708577c10cb0e8c529ab7",**

**"nombre": "juanito perez",**

**"usuario": {**

**"\_id": "5b662d5ff01576979ff4aab0",**

**"nombre": "qqqq",**

**"correo": "test1@test.com"**

**},**

**"hospital": {**

**"\_id": "5b6700a4d6cb4420c0c30c98",**

**"nombre": "hospital3",**

**"img": null,**

**"usuario": "5b662d5ff01576979ff4aab0",**

**"\_\_v": 0**

**},**

**"\_\_v": 0**

**},**

Podemos observar toda la información que pedimos.

**Paginacion de búsquedas: es decir filtro de cuantos resultados quiero que me aparezcan por ejemplo cuando hagamos un get me regrese los primeros 5 y luego otros 5 y asi:**

Deberemos recibir un parámetro pero que sea opcional entonces decalraremos una variable lalmada desde y esperaremos un valor en el req en el query, pero como es un parámetro opcional sino viene nada vendrá undefined de modo que definiré que sino viene nada ponga cero. Deberemos estar seguros que es un numero y no un string porque después no funcionara de modo que parsearemos lo que venga.

*//get usuarios*

app.get('/usuarios', (req, res, next)=>{

var desde = req.query.desde || 0;

desde=Number(desde);

Para realizar el filtrado usamos la función de mongoose llamada limit, que le enviaremos como parámetro el numero de registros que queremos que nos traiga y con la función skip me saltara los primeros 5.

Este es el resultado solo usando la ruta <http://localhost:3000/usuarios/> que me activa **el limit(5)** es decir **me trae los primeros 5 registros**.

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b662d5ff01576979ff4aab0",

"nombre": "qqqq",

"correo": "test1@test.com",

"password": "123456",

"img": null

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711625c65701420cf56f5",

"nombre": "test1a",

"correo": "test1a@test.com",

"password": "$2a$10$AP2hzXZULIaW6iLNVhHdSOLX4j1WShN.RSsSSSuoGVOg65Z9wlt1e"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b67116c5c65701420cf56f6",

"nombre": "test2",

"correo": "test2@test.com",

"password": "$2a$10$Z/wyEAILa98fzSDIU4K41OSBhM2ToELu7jGCNaLZUVoa4P4/XmdNm"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711725c65701420cf56f7",

"nombre": "test3",

"correo": "test3@test.com",

"password": "$2a$10$kzTCzll4WcbelwRZlzMDPuj02fivKljoNPN56.E4Gy00obaSOkJEW"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711785c65701420cf56f8",

"nombre": "test4",

"correo": "test4@test.com",

"password": "$2a$10$yBD57SN4qs4GiJrYCaxFQeYZEAF9vSYp3gXAC/S.AO9Ffo5ECoAZS"

}

]

}

Para usar el desde usare el signo de interrogación ya que no es un parámetro obligatorio, el nombre de la variable desd igual a 5 esto hara que me salte los primeros 5 resultados y me muestr los 5 que vienen. <http://localhost:3000/usuarios?desde=5>

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b67117d5c65701420cf56f9",

"nombre": "test5",

"correo": "test5@test.com",

"password": "$2a$10$wJce8JxZegJzwU5ZhNTsD.SU.N7eiFeErP4nEqFgf/05y/Zt6ZO2W"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711835c65701420cf56fa",

"nombre": "test6",

"correo": "test6@test.com",

"password": "$2a$10$7aLbi3YcVSK/7x4N5Oa4W.sFX.VXNpBMWhk1YwWxHJQ9u5oHoq/bC"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711895c65701420cf56fb",

"nombre": "test7",

"correo": "test7@test.com",

"password": "$2a$10$k08gkgPFBiFM9E/Qvk59BegD6mxQJMeZzus9Ef.oxxGiagqzzSByW"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b67118e5c65701420cf56fc",

"nombre": "test8",

"correo": "test8@test.com",

"password": "$2a$10$4Wgb1U2B8xWWpybp9PhCouT.6qfyRm1A/S5HjCqJFhV0gOSbmsFy2"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711945c65701420cf56fd",

"nombre": "test9",

"correo": "test9@test.com",

"password": "$2a$10$53m1I44eUL6mC1OVxmclGeX0TK4wCHpggq7reFnn0.S4zHpOf6Hyy"

}

]

}

Si quisiéramos el siguiente grupo de 5 deberemos ingrementar en 5 el desde es decir va a 10

<http://localhost:3000/usuarios?desde=10>

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b67119a5c65701420cf56fe",

"nombre": "test10",

"correo": "test10@test.com",

"password": "$2a$10$YKcYDjoxCt1ALQI5Uz0lIufcywFVp8OWpOycWkHcL7C/wy.90nwIC"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b67119f5c65701420cf56ff",

"nombre": "test11",

"correo": "test11@test.com",

"password": "$2a$10$U.Hll5v5JNVpuk3kgj/Q8uwzyYjiPlY5ZGYq7YW98teel3rl1nyre"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711a65c65701420cf5700",

"nombre": "test12",

"correo": "test12@test.com",

"password": "$2a$10$AG59CNR8RyP78xR9kj6tfeQyOmUIVqbb4CVYIHFfLp.x1J/7HF.L6"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711ab5c65701420cf5701",

"nombre": "test13",

"correo": "test13@test.com",

"password": "$2a$10$t.YhFIIeSGVKgU.d4bopLu1HZLDUZkz1GYCRB6.Iut/KiraTNqP0i"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711b05c65701420cf5702",

"nombre": "test14",

"correo": "test14@test.com",

"password": "$2a$10$XfEgmKSMRn7KI.rHlQC58u1yPrqcqIBrXC6NLg/eOYRzLnh3Kr7Um"

}

]

}

Ahora mostremos aparte de que me muestre los usuarios que nos muestre cuantos usuarios tenemos eso lo hacemos con un una **función de mongoose count** qu eme cuente todos los usuarios.

*//get usuarios*

app.get('/usuarios', (req, res, next)=>{

var desde = req.query.desde || 0;

desde=Number(desde);

Usuario.find({},'nombre correo img role password')

.skip(desde)

.limit(5)

.exec((err, usuario)=>{

if(err){

res.status(500).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

Usuario.count({}, (err, conteo)=>{

res.status(200).json({

ok:true,

mensaje:"peticion realizada correctamente",

usuarios:usuario,

total:conteo

});

})

});

});

Y dentro daremos la respuesta con el resultado del metodo count.

{

"ok": true,

"mensaje": "peticion realizada correctamente",

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b67119a5c65701420cf56fe",

"nombre": "test10",

"correo": "test10@test.com",

"password": "$2a$10$YKcYDjoxCt1ALQI5Uz0lIufcywFVp8OWpOycWkHcL7C/wy.90nwIC"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b67119f5c65701420cf56ff",

"nombre": "test11",

"correo": "test11@test.com",

"password": "$2a$10$U.Hll5v5JNVpuk3kgj/Q8uwzyYjiPlY5ZGYq7YW98teel3rl1nyre"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711a65c65701420cf5700",

"nombre": "test12",

"correo": "test12@test.com",

"password": "$2a$10$AG59CNR8RyP78xR9kj6tfeQyOmUIVqbb4CVYIHFfLp.x1J/7HF.L6"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711ab5c65701420cf5701",

"nombre": "test13",

"correo": "test13@test.com",

"password": "$2a$10$t.YhFIIeSGVKgU.d4bopLu1HZLDUZkz1GYCRB6.Iut/KiraTNqP0i"

},

{

"role": "ADMIN\_ROLE",

"\_id": "5b6711b05c65701420cf5702",

"nombre": "test14",

"correo": "test14@test.com",

"password": "$2a$10$XfEgmKSMRn7KI.rHlQC58u1yPrqcqIBrXC6NLg/eOYRzLnh3Kr7Um"

}

],

"total": 16

}

**Implementemos esto en los otros métodos get osea médicos y hospitales. No lo pegare aquí para ganar tiempo.**

**Hagamos una búsqueda ngeneral en todas las colecciones:**

Crearemos la ruta llamada BusquedaGeneralroutes.js la agregaremos en el app.js y empecemos.

Debemos crear procesos asíncronos que nos busquen en cada una de las collecciones y nos traigan los resultados empecemos con la de hospitales

Lo primero sera recoger por los params el paramentro de busqueda

var express = require('express');

var Hospital = require('../models/hospitalesModels');

var app = express();

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

lo segundo es convertir ese parámetro en una expresión regular.

Eso lo hacemos instanciando el onbjeto RegExp y pasando primero el parámetro recogido en este caso búsqueda y luego un segundo parámetro que es opcional que es la ‘i’ esto hara que nuestra búsqueda sea insensible a mayúsculas y minúsculas y que busque miesrtas escribe es como los %% para myql.

var regex = new RegExp(busqueda, 'i');

luego en el find pasamos por ejemplo el nombre del campo que queremos buscar y lo igualamos a la expresión regular creada.

var express = require('express');

var Hospital = require('../models/hospitalesModels');

var app = express();

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

Hospital.find({nombre:regex}, (err, hospitales)=>{

res.status(200).json({

ok:true,

mensaje:"ruta correcta",

hospitales:hospitales

});

})

});

module.exports = app;

Ahora si yo escribo la ruta

<http://localhost:3000/busqueda/todo/niños>

este sera el resultado me traerá todo lo que tenga la palabra niños

{

"ok": true,

"mensaje": "ruta correcta",

"hospitales": [

{

"\_id": "5b672940c0ee282bd8a148cf",

"nombre": "hospital de niños 3",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

},

{

"\_id": "5b672942c0ee282bd8a148d0",

"nombre": "hospital de niños 2",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

},

{

"\_id": "5b672945c0ee282bd8a148d1",

"nombre": "hospital de niños 1",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

}

]

}

**Como haríamos para buscar en varias colecciones?**

Lo que debemos hacer es crear procesos asíncronos y esperar a que respondan para poder retornar el mensaje o la respuesta con todo lo que quisimos buscar.

Empecemos con la de hospitales:

Transformemos esta respuesta en una promesa

Cremos una función llamada busquedaHospitales que reciba dos parámetros la **búsqueda** y la expresión regular **regex**, esta función retornara una promesa, dentro de esa promesa implementemos la búsqueda normal, si hay error devolvemos un mensaje de error, sino enviare la data de los hospitales.

function buscarHospitales(busqueda, regex){

return new Promise((resolve, reject)=>{

Hospital.find({nombre: regex}, (err, hospitales)=>{

if(err){

reject('error al cargar Hospitales ', err);

}else{

resolve(hospitales);

}

});

});

}

Esta función la implementaremos en el metodo get de la siguiente manera como haríamos con una promesa, bueno es que es una promesa.

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

buscarHospitales(busqueda, regex)

.then(hospitales=>{

res.status(200).json({

ok:true,

mensaje:"ruta correcta",

hospitales:hospitales

});

});

});

Hasta ahora el código de busquedaGeneralRoutes.js es la siguiente.

var express = require('express');

var Hospital = require('../models/hospitalesModels');

var app = express();

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

buscarHospitales(busqueda, regex)

.then(hospitales=>{

res.status(200).json({

ok:true,

mensaje:"ruta correcta",

hospitales:hospitales

});

});

});

function buscarHospitales(busqueda, regex){

return new Promise((resolve, reject)=>{

Hospital.find({nombre: regex}, (err, hospitales)=>{

if(err){

reject('error al cargar Hospitales ', err);

}else{

resolve(hospitales);

}

});

});

}

module.exports = app;

agreguemos ahora la parte de los medicos, lo primero es importer el modelo de medicos.

var Medicos = require('../models/medicosModels');

Clonemos la función de buscar hospitales y adecuemosla a los médicos.

function buscarMedicos(busqueda, regex){

return new Promise((resolve, reject)=>{

Medicos.find({nombre: regex}, (err, medicos)=>{

if(err){

reject('error al cargar Medicos ', err);

}else{

resolve(medicos);

}

});

});

}

Ahora como lo implementamos junto con los hospitales?

**El ECMASCRIPT 6** incluyo un metodo interesante para las Promesas y es el **Promise.all**, este metodo nos permite mandar un arreglo de promesas ejecutarlas y si todas responden correctamente podemos disparar un then que recibirá las respuestas, cada una de las respuestas de las promesas va a venir dentro del arreglo en las mismas posiciones en las que se encuentran, dentro de ese then colocaremos es res.status para dar la respuesta y haremos referencia a las posiciones de los arrglos para obtener las respuestas deseadas. y si una falla tendríamos que mandar un catch.

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

Promise.all([

buscarHospitales(busqueda, regex),

buscarMedicos(busqueda, regex)

])

.then(respuestas=>{

res.status(200).json({

ok:true,

hospitales: respuestas[0],

medicos:respuestas[1]

});

})

});

De modo que si ejecutamos nuevamente la petición get de la siguiente ruta <http://localhost:3000/busqueda/todo/niño> nos dara como resultado as búsquedas en la colección de hospitales y de médicos que coinsidan con la palabra niños

{

"ok": true,

"**hospitales**": [

{

"\_id": "5b672940c0ee282bd8a148cf",

"nombre": "hospital de niños 3",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

},

{

"\_id": "5b672942c0ee282bd8a148d0",

"nombre": "hospital de niños 2",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

},

{

"\_id": "5b672945c0ee282bd8a148d1",

"nombre": "hospital de niños 1",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

}

],

"**medicos**": [

{

"\_id": "5b673c3e1b7bb40984b9d7f7",

"nombre": "medico para niños 2",

"usuario": "5b662d5ff01576979ff4aab0",

"hospital": "5b6700afd6cb4420c0c30c99",

"\_\_v": 0

},

{

"\_id": "5b673c411b7bb40984b9d7f8",

"nombre": "medico para niños ",

"usuario": "5b662d5ff01576979ff4aab0",

"hospital": "5b6700afd6cb4420c0c30c99",

"\_\_v": 0

}

]

}

Hagamos lo mismo con los usuarios pero aquí buscaremos en 2 documentos de la colección de usuarios. **Supongamos que buscaremos tanto por el nombre como por el email**.

Para esto usamos el metodo or de mongoose que nos permitirá buscar tantos campos de un documento como queramos el resto es igual.

function buscarUsuarios(busqueda, regex){

return new Promise((resolve, reject)=>{

Usuarios.find()

.or([{'nombre':regex}, {'email':regex}])

.exec((err, usuarios)=>{

if(err){

reject('Error al cargar usuarios '+ err);

}else{

resolve(usuarios);

}

});

});

}

Ahora implementemoslo en el get:

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

Promise.all([

buscarHospitales(busqueda, regex),

buscarMedicos(busqueda, regex),

buscarUsuarios(busqueda, regex)

])

.then(respuestas=>{

res.status(200).json({

ok:true,

hospitales: respuestas[0],

medicos:respuestas[1],

usuarios:respuestas[2]

});

})

});

La respuesta final si aplico la siguiente dirección <http://localhost:3000/busqueda/todo/urgencias> sera traerme las búsquedas de todas las colecciones en las que encuentre coincidencias.

{

"ok": true,

"hospitales": [

{

"\_id": "5b67404487312521142ddaaa",

"nombre": "hospital urgencias",

"img": null,

"usuario": "5b662d5ff01576979ff4aab0",

"\_\_v": 0

}

],

"medicos": [

{

"\_id": "5b67402687312521142ddaa9",

"nombre": "medico urgencias",

"usuario": "5b662d5ff01576979ff4aab0",

"hospital": "5b6700afd6cb4420c0c30c99",

"\_\_v": 0

}

],

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b67407287312521142ddaab",

"nombre": "medico urgencias",

"correo": "urgencias@test.com",

"password": "$2a$10$5oOUNETs/B1k/pa5F8tznuxT4bgezk4QJ1KLpQQqsOZPJlYEUVk6u",

"\_\_v": 0

}

]

}

Mejoremos todo usando el populate para ver las relaciones.

var express = require('express');

var Hospital = require('../models/hospitalesModels');

var Medicos = require('../models/medicosModels');

var Usuarios = require('../models/usuarioModel');

var app = express();

app.get('/todo/:busqueda', (req, res, next)=>{

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

Promise.all([

buscarHospitales(busqueda, regex),

buscarMedicos(busqueda, regex),

buscarUsuarios(busqueda, regex)

])

.then(respuestas=>{

res.status(200).json({

ok:true,

hospitales: respuestas[0],

medicos:respuestas[1],

usuarios:respuestas[2]

});

})

});

function buscarHospitales(busqueda, regex){

return new Promise((resolve, reject)=>{

Hospital.find({nombre: regex})

.populate('usuario', 'nombre email')

.exec((err, hospitales)=>{

if(err){

reject('error al cargar Hospitales ', err);

}else{

resolve(hospitales);

}

});

});

}

function buscarMedicos(busqueda, regex){

return new Promise((resolve, reject)=>{

Medicos.find({nombre: regex})

.populate('hospital', 'nombre')

.populate('usuario','nombre')

.exec((err, medicos)=>{

if(err){

reject('error al cargar Medicos ', err);

}else{

resolve(medicos);

}

});

});

}

function buscarUsuarios(busqueda, regex){

return new Promise((resolve, reject)=>{

Usuarios.find({},'nombre correo role')

.or([{'nombre':regex}, {'email':regex}])

.exec((err, usuarios)=>{

if(err){

reject('Error al cargar usuarios '+ err);

}else{

resolve(usuarios);

}

});

});

}

module.exports = app;

el resiltado sera:

{

"ok": true,

"hospitales": [

{

"\_id": "5b67404487312521142ddaaa",

"nombre": "hospital urgencias",

"img": null,

"usuario": {

"\_id": "5b662d5ff01576979ff4aab0",

"nombre": "qqqq"

},

"\_\_v": 0

}

],

"medicos": [

{

"\_id": "5b67402687312521142ddaa9",

"nombre": "medico urgencias",

"usuario": {

"\_id": "5b662d5ff01576979ff4aab0",

"nombre": "qqqq"

},

"hospital": {

"\_id": "5b6700afd6cb4420c0c30c99",

"nombre": "hospital4"

},

"\_\_v": 0

}

],

"usuarios": [

{

"role": "ADMIN\_ROLE",

"\_id": "5b67407287312521142ddaab",

"nombre": "medico urgencias",

"correo": "urgencias@test.com"

}

]

}

Ahora agregemos un get para buscar por colleccion. Es decir una por una según la que yo elija

Lo primero es recibir la colección en la que quiero buscar y la busqueda que quiero hacer ambos los hare por params. También definiremos la expresión regular de la búsqueda, luego definamos una variable llamada promesa que sera la promesa que queremos ejecutar.

*// ==================================================*

*// Busqueda Especifica*

*// ==================================================*

app.get('/coleccion/:coleccion/:busqueda', (req, res)=>{

var coleccion = req.params.coleccion;

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

var promesa;

});

Hagamos un swith para validar que la colecccion ingresada sea la correcta.

Si se ingresa correctamente todo puedo ejecutar la promesa donde dare una respuesta. En la respuesta si quiero que tenga la colección correcta siempre tendre que usar las propiedades de objetos computadas y consiste en encerrar la variable en [tabla] para que este valor sea relacionado con lo que digita el usuario y pueda ser obsrvado de forma dinámica.

*// ==================================================*

*// Busqueda Especifica*

*// ==================================================*

app.get('/coleccion/:tabla/:busqueda', (req, res)=>{

var tabla = req.params.tabla;

console.log(tabla);

var busqueda = req.params.busqueda;

var regex = new RegExp(busqueda, 'i');

var promesa;

switch(tabla){

case 'usuarios':

promesa = buscarUsuarios(busqueda, regex);

break;

case 'medicos':

promesa = buscarMedicos(busqueda, regex);

break;

case 'hospitales':

promesa = buscarHospitales(busqueda, regex);

break;

*//en caso de que no se ingrese ninguno de los anteriores*

default:

return res.status(400).json({

ok:false,

mensaje: "solo ingrese usuarios, medicos y hospitales",

error:{ message: 'Tipo de tabla no valido'}

});

}

promesa.then(data=>{

res.status(200).json({

ok:true,

[tabla]:data

});

});

});

**Subamos archivos al servidor:**

Creemos una nueva ruta para esto llamada uploadRoutes.js qui subiremos las fotos de los médicos, hospitales y usuarios. y enlacemosla con el app.js y ahcemos la prueba.

var express = require('express');

var mdAutenticacion = require('../middlewares/autenticacion');

*//inicialziar variables*

var app = express();

app.get('/', (req, res, next)=>{

res.status(200).json({

ok:true,

mensaje:"ruta correcta"

});

});

module.exports = app;

<http://localhost:3000/upload>

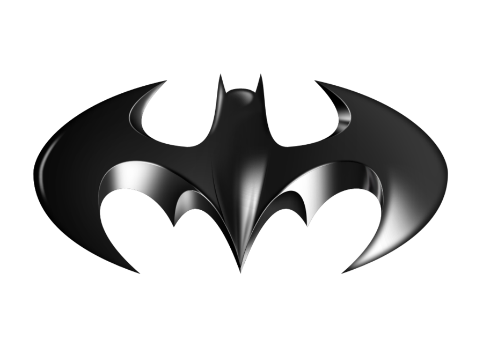
{

"ok": true,

"mensaje": "ruta correcta"

}

Tengamos una imagen a la mano, pongámosle un nombre difícil por ejemplo **batman.cosa.123.png**



Para empezar a configurar el backend para ue acepte imágenes necesitamos isntalar una librería

<https://github.com/richardgirges/express-fileupload>

la instalamos:

npm install --save express-fileupload

el ejemplo completo de la documentación nos servirá para guiarnos.

const express = require('express');

const fileUpload = require('express-fileupload');

const app = express();

// default options

app.use(fileUpload());

app.post('/upload', function(req, res) {

if (!req.files)

return res.status(400).send('No files were uploaded.');

// The name of the input field (i.e. "sampleFile") is used to retrieve the uploaded file

let sampleFile = req.files.sampleFile;

// Use the mv() method to place the file somewhere on your server

sampleFile.mv('/somewhere/on/your/server/filename.jpg', function(err) {

if (err)

return res.status(500).send(err);

res.send('File uploaded!');

});

});

Entonces empecemos en el **uploadRoutes.js**

var express = require('express');

var mdAutenticacion = require('../middlewares/autenticacion');

const fileUpload = require('express-fileupload');

Implementamos el middleware de la documentacion.

*// default options*

app.use(fileUpload());

y luego vien la peticion http, lo interesante es que cuando se suben los archivos estarán en la variable **samplesFile** como en la documentación. El siguiente paso sera mover el archico con el metodo mv y colocarlo en una dirección especifica se debería renombrar el archivo ya que otros usuarios podrían subir el archivo con el mismo nombre y lo reemplazararia.

var express = require('express');

var fileUpload = require('express-fileupload');

var app = express();

*// default options*

app.use(fileUpload());

app.post('/', (req, res, next)=>{

if (!req.files) return res.status(400).json({message:"'No files were uploaded.'"});

*//obtener nombre del archivo*

let archivo = req.files.archivo;

let nombreCortado = archivo.name.split('.');

console.log(nombreCortado);

*//ultima posicion del arreglo es la extension*

let extension = nombreCortado[nombreCortado.length -1]

*//validasion para las extensiones validas*

var extensionesValidas = ['png', 'jpg', 'git', 'jpeg'];

if(extensionesValidas.indexOf(extension) < 0){ *// si es -1 es que no encontro en el array de extenciones validas*

return res.status(400).json({

message:"extension no valida",

errors:{ message:"debe ser png, jpg, jpeg"}

});

}

if(req.files) return res.status(200).json({

message:'si hay archivo',

nombreCortado:nombreCortado,

extension:extension

});

});

module.exports = app;

lo que sigue es determinar la carpeta donde vamos a colocar la imagen, moverla a ese lugar y asignar esa imagen a un usuario.

Lo primero sera recibir para quien es la imagen, para un usuario, para un hospital o para un medico eso lo agregaremos en el params **tipo**, y por ultimo el **id** del usuario que queremos actualizar. Por tanto la petición post cambia. y recibimos los params que necesitamos

app.post('/:tipo/:id', (req, res, next)=>{

var tipo = req.params.tipo;

var id = req.params.id;

*//nombre archivo personalizado sera id\_usuario\_+numerorandom.extension del archivo*

var nombreArchivo = `${id}-${new Date().getMilliseconds()}.${extension}`;

crearemos una carpeta para las imagenes llamada upload y dentro 3 carpetas una llamada usuarios, otra médicos y otra hospitales. Validemos también que lo que venga por el tipo sea médicos, hospitales o usuarios. Hasta el momento deberemos tener el archivo de la siguiente forma.

var express = require('express');

var fileUpload = require('express-fileupload');

var app = express();

*// default options*

app.use(fileUpload());

app.post('/:tipo/:id', (req, res, next)=>{

var tipo = req.params.tipo;

var id = req.params.id;

if (!req.files) return res.status(400).json({message:"No files were uploaded."});

*//tipo valido par ingresar la ccoleccion donde se guardara la imagen*

var tiposValidos =["hospitales", "medicos", "usuarios"];

if(tiposValidos.indexOf(tipo)<0){

return res.status(400).json({

message:"extension no valida",

errors:{ message:"debe ser hospitales, medicos, usuarios"}

});

}

*//obtener nombre del archivo*

let archivo = req.files.archivo;

let nombreCortado = archivo.name.split('.');

let extension = nombreCortado[nombreCortado.length -1]

*//validasion para las extensiones validas*

var extensionesValidas = ['png', 'jpg', 'git', 'jpeg'];

*// si es -1 es que no encontro en el array de extenciones validas*

if(extensionesValidas.indexOf(extension) < 0){

return res.status(400).json({

message:"extension no valida",

errors:{ message:"debe ser png, jpg, jpeg"}

});

}

*//nombre archivo personalizado sera id\_usuario\_+numerorandom.extension del archivo*

var nombreArchivo = `${ id }-${ new Date().getMilliseconds() }.${ extension }`;

*//mover el archivo*

var path = `./uploads/${tipo}/${nombreArchivo}`;

archivo.mv( path, err=>{

if(err){

return res.status(500).json({

message:"error al mover archivo",

errors:err

});

}

return res.status(200).json({

message:'archivo movido',

nombreCortado:nombreCortado

});

})

});

module.exports = app;

ahora asignemos la imagen a un usuario, medico u hospital y validemos que si la actualiza borre la antigua y ponga la nueva. Para eso haya que crear un metodo para actualziar los registros de la base de datos y como estamos manejando 3 tipos vamos a a hacerlos todos en un mismo metodo. Esta función se llamara subirDatosdeImagenporTipo, esta función recibirá **4 parametros**, **el primero** sera el **tipo** es decir un medico, un usuario o un hospital el segundo sera el id, **el tercero el nombre del archivo** **y el cuarto** sera el res. Para poder insertar en la colleccion de usuarios necesito el modelo.

var Usuario = require('../models/usuarioModel');

para borrar las imágenes viejas en caso que se quiera cambiar de imagen voy a necsitar el sile sistema asi que también lo importare.

var fs = require('fs');

no hay que instalarlo ya que viene nativo en node.

El código de **uploadRoutes.js** quedaría de la siguiente manera.

var express = require('express');

var Usuario = require('../models/usuarioModel');

var Medico = require('../models/medicosModels');

var Hospital = require('../models/hospitalesModels');

var fileUpload = require('express-fileupload');

var fs = require('fs');

var app = express();

*// default options*

app.use(fileUpload());

app.post('/:tipo/:id', (req, res, next)=>{

var tipo = req.params.tipo;

var id = req.params.id;

if (!req.files) return res.status(400).json({message:"No files were uploaded."});

*//tipo valido par ingresar la ccoleccion donde se guardara la imagen*

var tiposValidos =["hospitales", "medicos", "usuarios"];

if(tiposValidos.indexOf(tipo)<0){

return res.status(400).json({

message:"extension no valida",

errors:{ message:"debe ser hospitales, medicos, usuarios"}

});

}

*//obtener nombre del archivo*

let archivo = req.files.archivo;

let nombreCortado = archivo.name.split('.');

let extension = nombreCortado[nombreCortado.length -1]

*//validasion para las extensiones validas*

var extensionesValidas = ['png', 'jpg', 'git', 'jpeg'];

*// si es -1 es que no encontro en el array de extenciones validas*

if(extensionesValidas.indexOf(extension) < 0){

return res.status(400).json({

message:"extension no valida",

errors:{ message:"debe ser png, jpg, jpeg"}

});

}

*//nombre archivo personalizado sera id\_usuario\_+numerorandom.extension del archivo*

var nombreArchivo = `${ id }-${ new Date().getMilliseconds() }.${ extension }`;

*//mover el archivo*

var path = `./uploads/${tipo}/${nombreArchivo}`;

archivo.mv( path, err=>{

if(err){

return res.status(500).json({

message:"error al mover archivo",

errors:err

});

}

subirDatosdeImagenporTipo(tipo, id, nombreArchivo, res);

*//return res.status(200).json({*

*// message:'archivo movido',*

*// nombreCortado:nombreCortado*

*//});*

});

});

function subirDatosdeImagenporTipo(tipo, id, nombreArchivo, res){

if(tipo==='usuarios'){

Usuario.findById(id, (err, usuario)=>{

if(!usuario){

return res.status(500).json({

message:"no existe el usuario en la db",

errors:{message:'Usuario no existe'}

});

}

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

*// ====en caso de que ya tubiera una imagen la borramos ===== //*

var pathViejo = "./uploads/usuarios/"+usuario.img;

if(fs.existsSync(pathViejo)){

fs.unlink(pathViejo)*//con unlink borramos*

}

*// =========================================================== //*

*// \*\*\*\*\*\*\*Almacenamos el nombre del archivo en la db\*\*\*\*\*\*\*\*\*\* //*

usuario.img = nombreArchivo;

*// =========Actualziamos el usuario ========================== //*

usuario.save((err, usuarioActualizado)=>{

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

return res.status(200).json({

message:"Usuario Actualizado",

usuario:usuarioActualizado

});

});

});

}

if(tipo==='medicos'){

Medico.findById(id, (err, medico)=>{

if(!medico){

return res.status(500).json({

message:"no existe el medico en la db",

errors:{message:'Medico no existe'}

});

}

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

*// ====en caso de que ya tubiera una imagen la borramos ===== //*

var pathViejo = "./uploads/medicos/"+medico.img;

if(fs.existsSync(pathViejo)){

fs.unlink(pathViejo)*//con unlink borramos*

}

*// =========================================================== //*

*// \*\*\*\*\*\*\*Almacenamos el nombre del archivo en la db\*\*\*\*\*\*\*\*\*\* //*

medico.img = nombreArchivo;

*// =========Actualziamos el medico ========================== //*

medico.save((err, medicoActualizado)=>{

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

return res.status(200).json({

message:"Medico Actualizado",

medico:medicoActualizado

});

});

});

}

if(tipo==='hospitales'){

Hospital.findById(id, (err, hospital)=>{

if(!hospital){

return res.status(500).json({

message:"no existe el usuario en la db",

errors:{message:'hospital no existe'}

});

}

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

*// ====en caso de que ya tubiera una imagen la borramos ===== //*

var pathViejo = "./uploads/hospitales/"+hospital.img;

if(fs.existsSync(pathViejo)){

fs.unlink(pathViejo)*//con unlink borramos*

}

*// =========================================================== //*

*// \*\*\*\*\*\*\*Almacenamos el nombre del archivo en la db\*\*\*\*\*\*\*\*\*\* //*

hospital.img = nombreArchivo;

*// =========Actualziamos el hospital ========================== //*

hospital.save((err, hospitalActualizado)=>{

if(err){

return res.status(400).json({

message:"error al accesar a la db",

errors:err

});

}

return res.status(200).json({

message:"Hospital Actualizado",

hospital:hospitalActualizado

});

});

});

}

}

module.exports = app;

Creemos una ruta para obtener las imagines:

Creemos un servicio que nos permita retornar la imagen de un usuario, un hospital o un medico.

1. Crear un nuevo archivo en routes que nos permita mostrar las imágenes de usuarios, médicos u hospitales, el archivo se llamara **mostrarImagenesRoutes.js**
2. Enlacemosla al app.js
3. Creemos una carpeta llamada assets que tendrá la imagen por defecto en caso de que no exista imagen en hospitales, médicos, usuarios. Guardaremos allí la imagen en este caso se llama no img-jps.
4. Necesitaremos el **tipo** que en este caso sera la colección y la **imagen** que debo buscar

var express = require('express');

var app = express();

app.get('/:tipo/:img', (req, res, next)=>{

var img = req.params.img;

var tipo = req.params.tipo;

res.status(200).json({

ok:true,

message:"peticion realizada correctamente"

});

});

module.exports = app;

ahora debemos verificar si tipo tiene una imagen sino la tiene deberemos colocar la imagen por defecto de los assets

debemos crear el path para encontrar las imágenes y para eso usumos una librería que trae node por defecto que es la librería **path** entonces importemosla.

var path = require('path');

ahora definamos el path para encontrar las imágenes:

var pathImagen = path.resolve(\_\_dirname, `../uploads/${tipo}/${img}`);

ahora para verificar si existe la imagen debere usar la librería fs también nativa de nodejs, importemosla y usémosla

var fs = require('fs');

*// =========si existe la imagen ==== //*

if(fs.existsSync(pathImagen)){

res.sendFile(pathImagen);

*//sino mostremos la de los assets*

}else{

var pathNoImage = path.resolve(\_\_dirname, '../assets/no-img.jpg');

res.sendFile(pathNoImage);

}

El archivo mostrarImagenesRoutes.js debería quedar de la siguiente manera

var express = require('express');

var path = require('path');

var fs = require('fs');

var app = express();

app.get('/:tipo/:img', (req, res, next)=>{

var img = req.params.img;

var tipo = req.params.tipo;

var pathImagen = path.resolve(\_\_dirname, `../uploads/${ tipo }/${ img }`);

*// =========si existe la imagen ==== //*

if(fs.existsSync(pathImagen)){

res.sendFile(pathImagen);

*//sino mostremos la de los assets*

}else{

var pathNoImage = path.resolve(\_\_dirname, '../assets/no-img.jpg');

res.sendFile(pathNoImage);

}

});

module.exports = app;

al hacer las pruebas en postman sobre esta ruta

<http://localhost:3000/img/hospitales/5b67003e9c87b9e098367517-370.jpg> deberia mostrar la imagen del hospital y si cambiamos el tipo y el nombre de la imagen deberiamso mostrar médicos y usuarios.

Guardemos los cambios en gitgub

Git status

Git add .

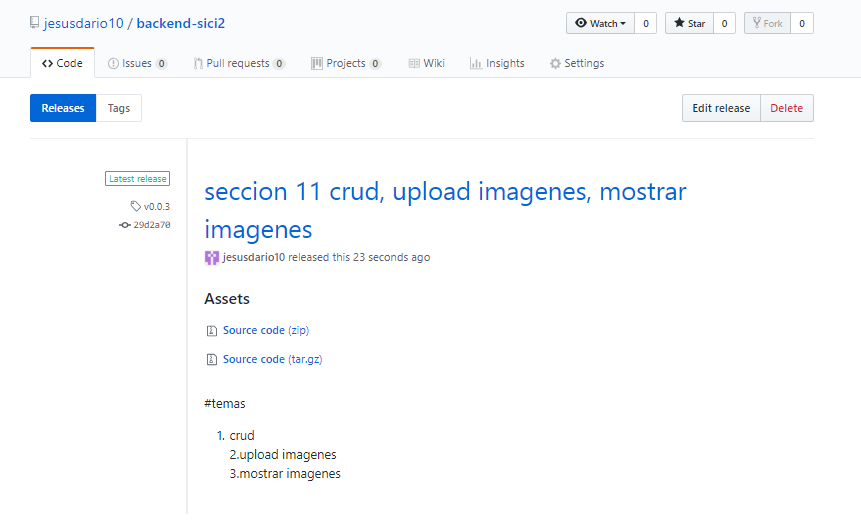
Git commit –m “crud, upload imagines, mostrar imagenes”

Git push origin master

Git tag

Git tag –a v0.0.3 –m “ seccion 11 crud, upload imagines, mostrar imagenes”

Git push - -tags



Temas puntuales que aprenderemos en esta sección

Sección 12, Clase 136

Esta sección esta especializada en implementar el Google Sign-in en nuestro backend server:

* Crear una aplicación en Google Developer Console
* Generar el ID de nuestra aplicación y un ID Secreto de servidor
* Crear un login básico de pruebas usando el API de Google
* Generar un Token desde el front-end
* Validar el Token en nuestro back-end
* Tip para generar la documentación de nuestros servicios automáticamente

Al finalizar, crearemos el release tag y realizamos los respectivos respaldos en GIT y GitHub

Página oficial para implementar un login de Google

Sección 12, Clase 137

#### **Login de Google**

En la siguiente clase, trabajaremos implementando unas medidas de seguridad referente a un Token que nuestro Front-End enviará, por lo cual, debemos de validar dicho token contra un servicio que nos ofrece Google, que nos dirá si el Token y usuario es correcto.

Favor abrir y revisar esta página rápidamente, en las siguientes clases lo usaremos.

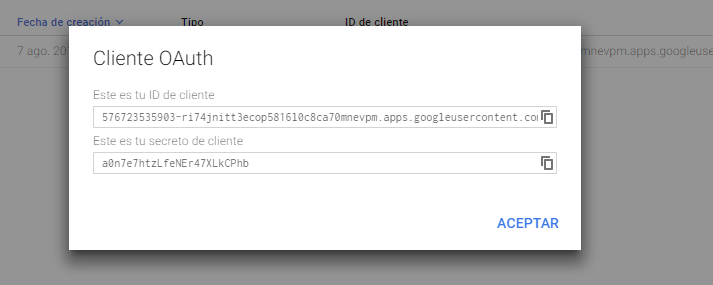
[Google API Console](https://developers.google.com/identity/sign-in/web/devconsole-project)

**crear un ID de la aplicación y un ID secreto Google Developer.**

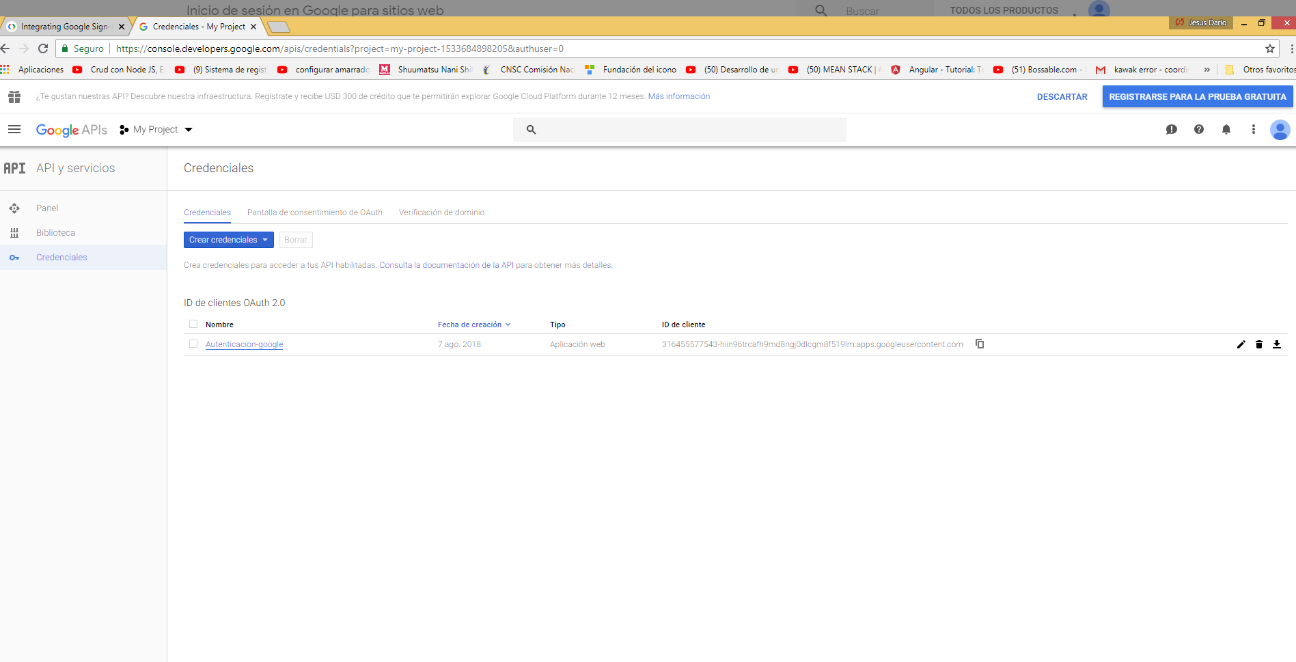
1 vayamos al link y empecemos creando nuestra aplicación de google.

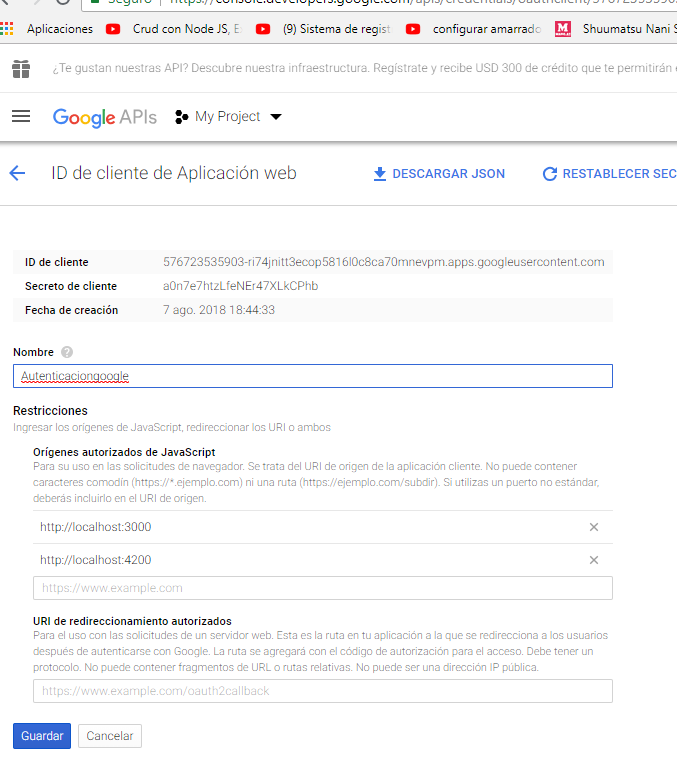
<https://developers.google.com/identity/sign-in/web/sign-in#before_you_begin>

2 demos click en confugure project y segiuimos los pasos hasta que nos genere el cliente OAuth



Cuando se cree entremosnuevamente para configurar los orígenes autorizados de javascript.





Creemos una aplciacion rápida para probar el googlesignin.

s

Temas puntuales que aprenderemos en esta sección

Sección 13, Clase 146

Esta sección tiene por objetivo implementar la seguridad de nuestra aplicación:

* Conectar el Front-end con el Back-end (login)
* Usar Sweet Alert para mostrar mensajes
* Errores de CORS
* Formas de habilitar el CORS en nuestro backend
* Login normal de usuario
* Login de Google
* LocalStorage para almacenar tokens
* Protección básica de rutas
* Logout

**Hay que levantar el frontend**

1. Trabajemos en el frontend creando el modelo de usuarios
2. Debemos crear una carpeta llamada **models** dentro de **app**.
3. Dentro un archivo llamado usuario.model.ts

El **?**, quiere decir que son parámetros opcionales.

export class UsuariosModel{

constructor(

public nombre: string,

public correo: string,

public password: string,

public img?:string,

public role? : string,

public google? : string,

public \_id? : string

){

}

}

4-creando usuarios en el frontend:

Borremos todos los usuarios, médicos y hospitales de la base de datos. La idea es que empeesmos a crear todo desde el frontend

Vayamos a la carpeta login, en esa carpeta esta el componente register, si nosotros vamos a al navegador y buscamos la vista register. <http://localhost:4200/#/register> y si la recargamos se quedara cargando porque a este componente no le hemos cargado el plugin que inicializa el custon.js entonces tambiendeberemos cargarlo.

import { Component, OnInit } from '@angular/core';

declare function init\_plugin();

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./login.component.css']

})

export class RegisterComponent implements OnInit {

constructor() { }

ngOnInit() {

init\_plugin();

}

}

Empecemos a crear nuestro formulario para trabajar en la vsita del register

1.creare un objeto llamado **forma** de tipo **FormGroup** y me pedirá importación.

import { FormGroup } from '@angular/forms';

export class RegisterComponent implements OnInit {

forma : FormGroup

en el ngOninit declarare todo mi formulario, como quiero que trabaje y cuales reglas debe seguir para eso usaremos el **FormControl(),** también deberemos importarlo. Ya que nos ayudara a crear declaraciones por cada campo del formulario.

En el **new FormControl** puedo colocar condiciones por ejemplo el primer parametroes el valor por defecto que van tener los campos. En el segundo parámetro la validación para eso usaremos el modulo Validators que nos ofrece una serie de valodaciones, ojo hay que importarlo.

import { FormGroup, FormControl, Validators } from '@angular/forms';

si hay mas de una validacion se puede mandr un arreglo con todas las validaciones.

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormControl, Validators } from '@angular/forms';

declare function init\_plugin();

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./login.component.css']

})

export class RegisterComponent implements OnInit {

forma : FormGroup

constructor() { }

ngOnInit() {

init\_plugin();

this.forma = new FormGroup({

nomre: new FormControl(null, Validators.required),

correo: new FormControl(null, [Validators.required, Validators.email]),

password : new FormControl(null, Validators.required),

password2 : new FormControl(null, Validators.required),

condiciones : new FormControl(false)

});

}

}

En las condiciones las dejare por ahora opcionales porque quiero hacer que cuando alguien haga un envio del formulario y falte un campo aparezca un popop diciendo que complete para continuar.

Como conectamos el objeto forma que contiene el formulario con la vista? Fácil con la directiva [formGroup] y lo igualamos a la forma. Esto dara un error porque ya estamos trabajando con el realticforms pero no lo tenemos importado en ningún lado, el html no lo conoce porque no lo hemos importado. Como nuestro login esta en la raiz deberemos importarlo en el app.module.ts

Esto sera en el app.module

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

imports: [

BrowserModule,

APP\_ROUTES,

PageModule,

FormsModule,

ReactiveFormsModule,

ServiceModule

],

En el html de register.component.html

*<*form [formGroup]="forma" class="form-horizontal form-material" id="registerform" action="index.html"*>*

Al guarder el error desaparecera.

Empecemos con el nombre

*<*input formControlName="nombre" name="nombre" class="form-control" type="text" required="" placeholder="Nombre"*>*

El correo

*<*input formControlName="correo" name="correo" class="form-control" type="email" required placeholder="Email"*>*

El password1

*<*input formControlName="password" name="password" class="form-control" type="password" required placeholder="Contraseña"*>*

El password2

*<*input formControlName="password2" name="password2" class="form-control" type="password" required placeholder="Confirmar contraseña"*>*

El elcheck podriamos hacer una pantalla para el contrato pero por ahora solo manejemos el check

*<*input formControlName="condiciones" name="condiciones" id="checkbox-signup" type="checkbox"*>*

En el form obtengamos el submit con (ngSubmit)=”registrarUsuario()”

*<*form [formGroup]="forma" (ngSubmit)="registrarUsuario()"

Declarare esa function en el componente y hare un clg del this.forma.value para ver que tiene.

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormControl, Validators } from '@angular/forms';

declare function init\_plugin();

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./login.component.css']

})

export class RegisterComponent implements OnInit {

forma : FormGroup

constructor() { }

ngOnInit() {

init\_plugin();

this.forma = new FormGroup({

nombre: new FormControl(null, Validators.required),

correo: new FormControl(null, [Validators.required, Validators.email]),

password : new FormControl(null, Validators.required),

password2 : new FormControl(null, Validators.required),

condiciones : new FormControl(false)

});

}

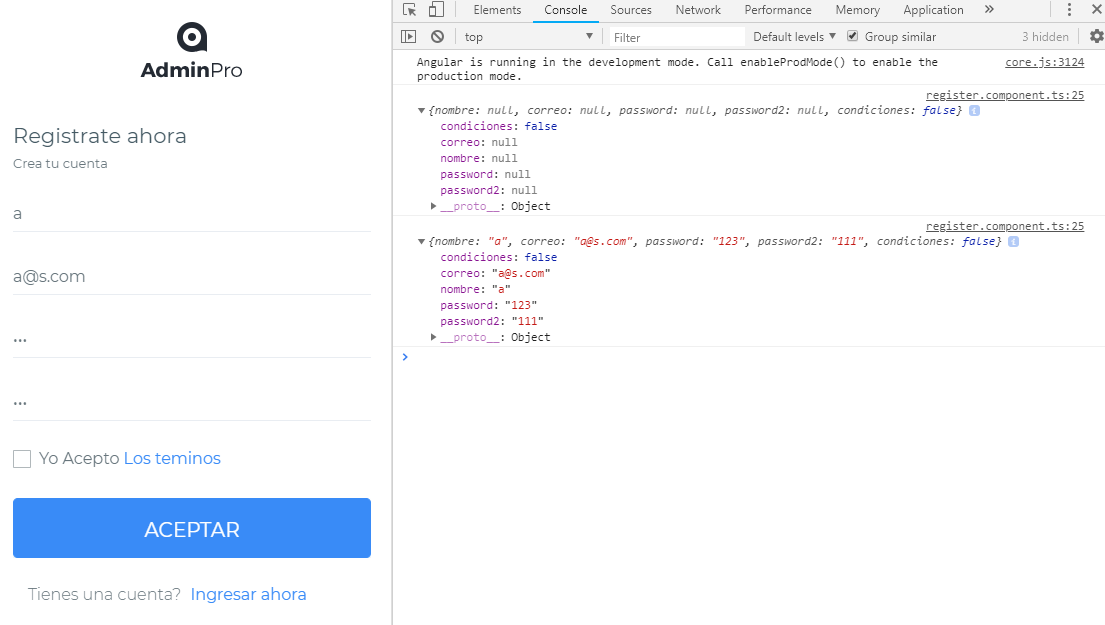
registrarUsuario(){

console.log(this.forma.value);

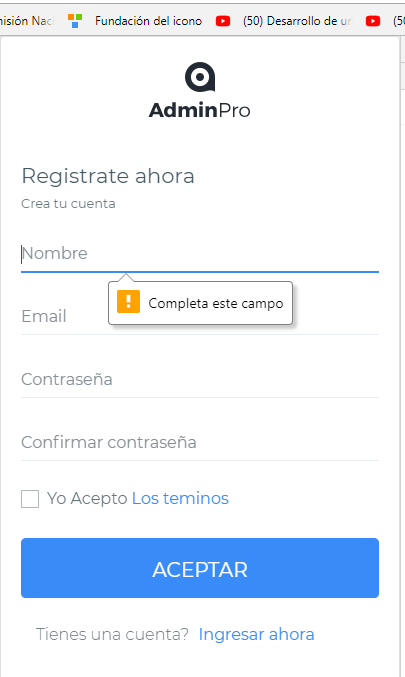
}

}

Si doy en el botón aceptar podremos observar que me esta capturando los null sino agrego nada y me esta dando los datos si agrego información en los campos



Cuando trabajamos con angular se desactivan las valdiaciones del html si queremos dejárselas usamos en l vista la directiva **ngNativeValidate** en el form. Esto para que funciones los atributos de validación de html por ejemplo ahora no nos dejara envir el formulario si los campos requeridos están vacios



Validemos las contraseñas iguales usando Reactiveforms esto lo puedo ahcer después de las laves del this.froma = new FormGroup({}, es decir aqui)

this.forma = new FormGroup({

nombre: new FormControl(null, Validators.required),

correo: new FormControl(null, [Validators.required, Validators.email]),

password : new FormControl(null, Validators.required),

password2 : new FormControl(null, Validators.required),

condiciones : new FormControl(false)

}, {validators: this.sonIguales('password', 'password2')});

Deberemos crear la function sonIguales y enviarla con la estructura de una función reactiveForm de la siguiente forma. Pero que nos de este resultado, **si son iguales devuelveme null**, **sino son iguales devuelveme true**,

Hagamos ahora en el formulario un texto que avise al usuario que las contraseñas deben ser iguales y que no mande el formulario sino es valido.

*<*div class="form-group" \*ngIf="forma.errors?.sonIguales && !forma.pristine"*>*

*<*p class="text-danger"*>*

Las contraseñas deben ser iguales

*</*p*>*

*</*div*>*

Y deberemos agregar al registrar una condición para cuando el formulario sea invalido no haga nada.

registrarUsuario(){

if(this.forma.invalid){

return;

}

if(!this.forma.value.condiciones){

console.log("Debe aceptar las condiciones");

return

}

}

Instalando sweetAlert para mensajes de información.

<https://github.com/t4t5/sweetalert>

lo instalamso como dice la documentación

npm install --save sweetalert

lo importamos de esta forma

import swal from 'sweetalert';

ahora vamos al componete register.component.ts y lo implementamos de la siguiente forma, coloquemoslo en el metodo registrarUsuario()

registrarUsuario(){

if(this.forma.invalid){

return;

}

if(!this.forma.value.condiciones){

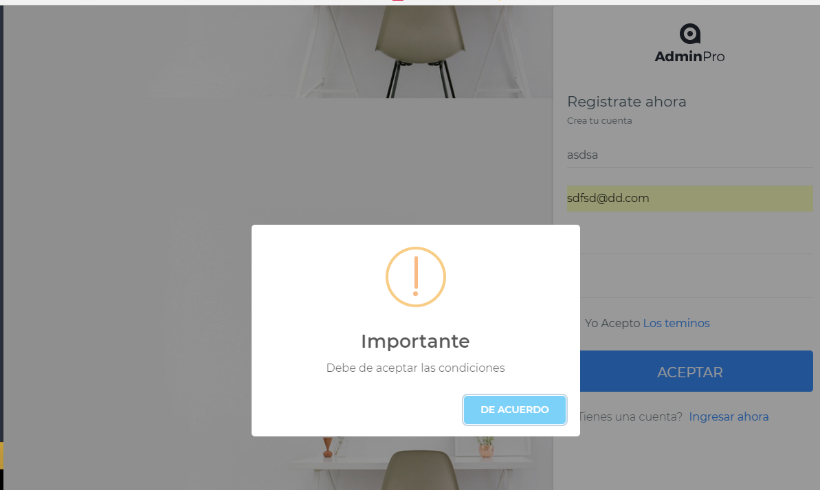
swal('Importante', 'Debe de aceptar las condiciones', 'warning');

return;

}

}

Al revisar en el navegador y no aceptar las codiciones aparecerá el siguiente mensaje



Llamando al servicio de crear el usuario.

Vayamos al backend en el crear usuario y quitemos la autenticación.

*//post crear usuario*

app.post('/usuarios', (req, res, next)=>{

var body = req.body

var usuario = new Usuario({

nombre : body.nombre,

correo: body.correo,

password: bcrypt.hashSync(body.password, 10),

img : body.img,

role : body.role

});

usuario.save((err, usuarioGuardado)=>{

if(err){

res.status(400).json({

ok:false,

mensaje:"no se pudieron traer los datos",

errors:err

});

}

res.status(201).send({

usuarioGuardado :usuarioGuardado,

usuarioToken: req.usuario,

ok:true

});

});

});

Probemos el servicio para ver si sigue funcionando.

{

"usuarioGuardado": {

"role": "ADMIN\_ROLE",

"\_id": "5b69dfbdad5aff24e0aa5e04",

"nombre": "medico urgencias",

"correo": "urgencias@test.com",

"password": "$2a$10$qZfN4nhqSe4NJxoicLZuBu1SFI4tGTKHphZkZpm6Gh.6B6DeA9bEG",

"\_\_v": 0

},

"ok": true

}

Todo va perfecto. Vamos al frontend y creemos un nuevo servicio llamado usuario.

Ng g s services/usuario/usuario - -spec = false

El servicio se creara y debere importarlo en mi service index que es mi servicio generaldode recojo tosos los servicios.

Servicio creado:

import { Injectable } from '@angular/core';

@Injectable()

export class UsuarioService {

constructor() {

console.log('Servicio de usuario listo');

}

}

Services.index.ts

**export { UsuarioService } from './usuario/usuario.service';**

export { SettingsService } from './settings/settings.service';

export { SharedService } from './shared/shared.service';

export { SidebarService } from './shared/sidebar.service';

el la parte del service.module.ts debere de proveerlo también en los providers

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { SettingsService, SidebarService, SharedService, UsuarioService } from './service.index';

@NgModule({

imports: [

CommonModule,

],

providers: [

SettingsService,

SidebarService,

SharedService,

UsuarioService

],

declarations: []

})

export class ServiceModule { }

ahora usemos el sevicio creado en el componente register.component.ts, debemos inyectarlo en el constructor e importarlo.

import { UsuarioService } from '../services/service.index';

export class RegisterComponent implements OnInit {

forma : FormGroup

constructor(

public \_usuarioService : UsuarioService

) { }

Si hacemos uun clg podremos observar la respuesta del servicio.



Ahora vamos al servicio **usuario.service.ts** y creemos un metodo que me llame el crearUsuario el backend. Este metodo recibirá un usuario de tipo del Modelo de usuario asi que deberemos importarlo.

Para recibir peticiones http debemos importar el servicio **HttpClient** en el usuario.service.ts e inicialicemos el constructor y Este **HttpClientModule** deberemos también importarlo en el service.module.ts en los imports

Usuario.service.ts:

import { Injectable } from '@angular/core';

import { UsuariosModel } from '../../models/usuario.model';

import { HttpClient } from '@angular/common/http';

@Injectable()

export class UsuarioService {

constructor(

public http: HttpClient

) {

console.log('Servicio de usuario listo');

}

crearUsuario(usuario:UsuariosModel){

}

}

Services.module.ts

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { SettingsService, SidebarService, SharedService, UsuarioService } from './service.index';

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [

CommonModule,

HttpClientModule

],

providers: [

SettingsService,

SidebarService,

SharedService,

UsuarioService

],

declarations: []

})

export class ServiceModule { }

ahora creemos un archivo de configuración para las rutas en app crearemos un nuevo folder llamado config y dentro un archivo llamado config.ts que exportara la ruta que usaremos para consumir el backend

config.ts tendrá

export const URL\_SERVICIOS = 'http://localhost:3000';

y esto lo importaremos en nuestro usuario.services.ts para poder usarlo.

Ahora creemos el metodo de crear usuario en el servicio. Se llamara crearUsuario, recibirá un usuario.

Usamos el http y el metodo post que reciben 2 parametros el primero es la ruta y el segundo el usuario que viene.

El necesario que esta función pueda ser subscrita por si ocurre algún error por lo que retornara la petición http ya que con el return podremos regresar un observador

import { Injectable } from '@angular/core';

import { UsuariosModel } from '../../models/usuarioModel';

import { HttpClient } from '@angular/common/http';

import { URL\_SERVICIOS } from '../../config/config';

@Injectable()

export class UsuarioService {

constructor(

public http: HttpClient

) {

console.log("servicio de usuario listo");

}

crearUsuario(usuario:UsuariosModel){

let url = URL\_SERVICIOS+'/usuarios';

return this.http.post(url, usuario);

}

}

Vayamos al register.componet.ts

En el metodo registrar usuario haremos uso del servicio crear usuario, para eso debemos instancial el modelo de usuarios y pasar los parámetro correspondientes del modelo, acuerdte que los campos obligatorios son nombre, correo y password.

registrarUsuario(){

if(this.forma.invalid){

return;

}

if(!this.forma.value.condiciones){

swal('Importante', 'Debe de aceptar las condiciones', 'warning');

return;

}

console.log(this.forma.valid + " es valido?");

console.log(this.forma.value);

let usuario = new UsuariosModel(this.forma.value.nombre, this.forma.value.correo, this.forma.value.password);

}

Ahora llamemos el servicio

registrarUsuario(){

if(this.forma.invalid){

return;

}

if(!this.forma.value.condiciones){

swal('Importante', 'Debe de aceptar las condiciones', 'warning');

return;

}

let usuario = new UsuariosModel(this.forma.value.nombre, this.forma.value.correo, this.forma.value.password);

console.log(usuario);

this.\_usuarioService.crearUsuario(usuario)

.subscribe(resp =>{

console.log(resp);

});

}

Esto aun no va a funcionar porque no hmos configurado el CORS del backend que permite que se realicen peticiones desd otros dominios.

Para esto vamos al backend.

Existe un middleware que nos permite facilitar el uso del CORS en el backend y es express cors

<https://github.com/expressjs/cors> y aqui podremos ver documentación donde nos mostraran como configurar <https://enable-cors.org/server.html> seleccionamos nuestro servidor y nos aparecerá el siguiente código.

app.use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

app.get('/', function(req, res, next) {

// Handle the get for this route

});

app.post('/', function(req, res, next) {

// Handle the post for this route

});

Copiamos la primera parte que es el middleware es decir

app.use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

Lo pegaremos en el app.js del backend justo debajo del var app = express().

De la siguiente manera. Y Añadiremos otra mas para los métodos http de la siguiente manera.

*//CORS PARA LAS CABECERAS DE LOS DATOS*

app.use((req, res, next) => {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Allow-Request-Method');

res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');

res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');

next();

});

El con esto hecho deberíamos estar guardando usaurios en la db, hagoa hagamos el login.

Vayamos a servicios y creemos el servicio.

*// =======================Login===================================== //*

login(usuario: UsuariosModel, recordar:boolean=false):Observable<any>{

let url = URL\_SERVICIOS+'/login';

let params = JSON.stringify(usuario);

let headers = new HttpHeaders().set('Content-Type','application/json');

return this.\_http.post(url, params, {headers:headers});

}

Vayamos al componente login y definamos el metodo par utilizar el servicio.

ingresar(forma: NgForm){

*//this.router.navigate(['/dashboard']);*

console.log(forma.valid);

console.log(forma.value);

if(forma.invalid){

return;

}

let usuario = new UsuariosModel(null, forma.value.correo, forma.value.password);

this.\_usuarioService.login(usuario, forma.value.recuerdame)

.subscribe(resp=>{

console.log(resp);

})

}

Vayamos a la vista de login y definamos las variables del formulario, el ngModel y los names.

*<!-- Main wrapper - style you can find in pages.scss -->*

*<!-- ============================================================== -->*

*<*section id="wrapper" class="login-register login-sidebar" style="background-image:url(../assets/images/background/login-register.jpg);"*>*

*<*div class="login-box card"*>*

*<*div class="card-body"*>*

*<*form ngNativeValidate #f="ngForm" class="form-horizontal form-material" (ngSubmit)="ingresar(f)"*>*

*<*a href="javascript:void(0)" class="text-center db"*><*img src="../assets/images/logo-icon.png" alt="Home" */><*br*/><*img src="../assets/images/logo-text.png" alt="Home" */></*a*>*

*<*div class="form-group m-t-40"*>*

*<*div class="col-xs-12"*>*

*<*input ngModel name="correo" class="form-control" type="email" required placeholder="Correo del usuario"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group"*>*

*<*div class="col-xs-12"*>*

*<*input ngModel name="password" class="form-control" type="password" required placeholder="Contraseña"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group row"*>*

*<*div class="col-md-12"*>*

*<*div class="checkbox checkbox-primary pull-left p-t-0"*>*

*<*input [(ngModel)]="recuerdame" name="recuerdame" id="checkbox-signup" type="checkbox" class="filled-in chk-col-light-blue"*>*

*<*label for="checkbox-signup"*>* Recuerdame *</*label*>*

*</*div*>*

*<*a href="javascript:void(0)" id="to-recover" class="text-dark pull-right"*><*i class="fa fa-lock m-r-5"*></*i*>* Olvide Contraseña?*</*a*>* *</*div*>*

*</*div*>*

*<*div class="form-group text-center m-t-20"*>*

*<*div class="col-xs-12"*>*

*<*button class="btn btn-info btn-lg btn-block text-uppercase btn-rounded" type="submit"*>*Ingresar*</*button*>*

*</*div*>*

*</*div*>*

*<*div class="row"*>*

*<*div class="col-xs-12 col-sm-12 col-md-12 m-t-10 text-center"*>*

*<*div class="social"*><*a href="javascript:void(0)" class="btn btn-facebook" data-toggle="tooltip" title="Login with Facebook"*>* *<*i aria-hidden="true" class="fa fa-facebook"*></*i*>* *</*a*>* *<*a href="javascript:void(0)" class="btn btn-googleplus" data-toggle="tooltip" title="Login with Google"*>* *<*i aria-hidden="true" class="fa fa-google-plus"*></*i*>* *</*a*>* *</*div*>*

*</*div*>*

*</*div*>*

*<*div class="form-group m-b-0"*>*

*<*div class="col-sm-12 text-center"*>*

No tienes cuenta? *<*a href="#/register" class="text-primary m-l-5"*><*b*>*Crear cuenta*</*b*></*a*>*

*</*div*>*

*</*div*>*

*</*form*>*

*<*form class="form-horizontal" id="recoverform" action="index.html"*>*

*<*div class="form-group "*>*

*<*div class="col-xs-12"*>*

*<*h3*>*recuperar contraeseña*</*h3*>*

*<*p class="text-muted"*>*Escriba su correo electronico y siga los pasos *</*p*>*

*</*div*>*

*</*div*>*

*<*div class="form-group "*>*

*<*div class="col-xs-12"*>*

*<*input class="form-control" type="text" required="" placeholder="Email"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group text-center m-t-20"*>*

*<*div class="col-xs-12"*>*

*<*button class="btn btn-primary btn-lg btn-block text-uppercase waves-effect waves-light" type="submit"*>*Reset*</*button*>*

*</*div*>*

*</*div*>*

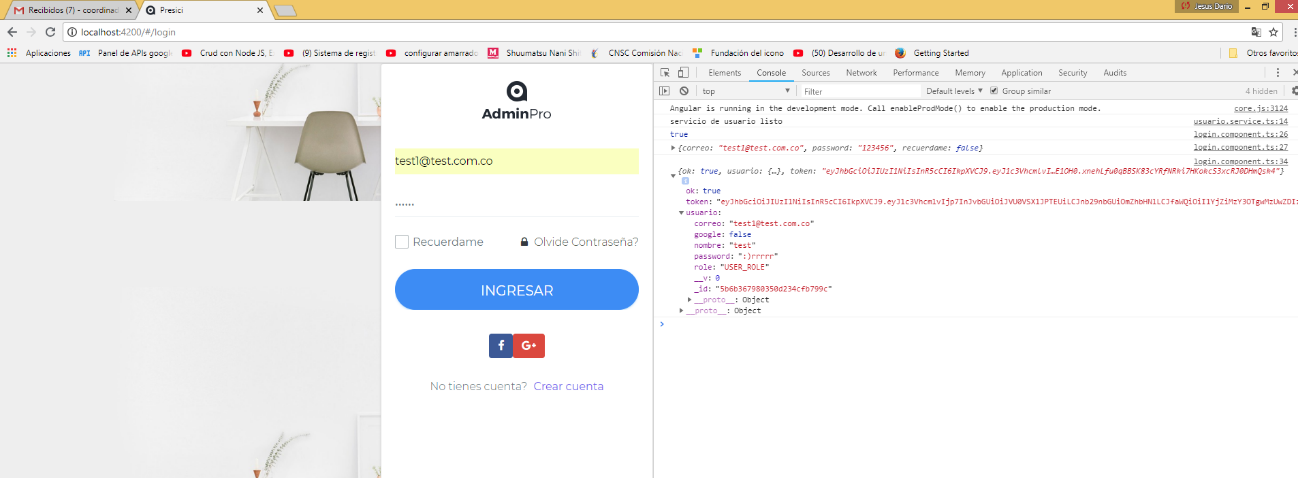
*</*form*>*

*</*div*>*

*</*div*>*

*</*section*>*

Deberíamos en la vista estar haciendo un login y recibiendo un token.



Guardemos la información del usuario en el localStorage para almacenarla aunque el usuario cierre el navegador web y lo vuelva abril.

Vallamos al usuario.service.ts IMPORTEMOS EL OBSERVABLE MAP

import { map } from "rxjs/operators";

ahora almacenemos en el local strorage

*// =======================Login=========================================== //*

login(usuario: UsuariosModel, recordar:boolean=false):Observable<any>{

let url = URL\_SERVICIOS+'/login';

let params = JSON.stringify(usuario);

let headers = new HttpHeaders().set('Content-Type','application/json');

return this.\_http.post(url, params, {headers:headers}).pipe(

map((resp: any) =>{

console.log(resp);

*//return resp.usuario;*

localStorage.setItem('id', resp.usuario.\_id);

localStorage.setItem('token', resp.token);

localStorage.setItem('usuario', JSON.stringify(resp.usuario));

}));

}

Implementemos esto en el componente login y que redireccione a dhasboard:

ingresar(forma: NgForm){

if(forma.invalid){

return;

}

let usuario = new UsuariosModel(null, forma.value.correo, forma.value.password);

this.\_usuarioService.login(usuario, forma.value.recuerdame)

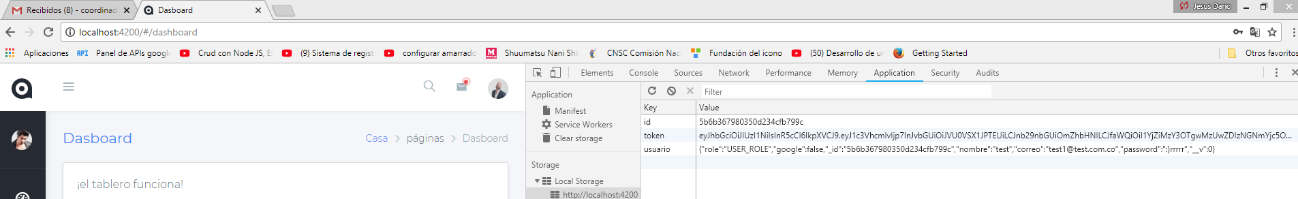
.subscribe(correcto=>{

this.router.navigate(['/dashboard']);

});

}

Podremos opservar que los datos se guararon en el localStorage



Función recuerdame:

En el sevicio lo guardaremos en el localStorage.

*// =======================Login=========================================== //*

login(usuario: UsuariosModel, recordar:boolean=false):Observable<any>{

if(recordar){

localStorage.setItem('email', usuario.correo);

}else{

localStorage.removeItem('email');

}

Luego en el componente en declaro una vriable email de tipo string y en el ngOninit obtengo el valor del localStorage

public email :string

ngOnInit() {

init\_plugin();

this.email = localStorage.getItem('email') || '';

if(this.email.length>1){

this.recuerdame = true;

}

}

Luego en la vista relacionare el campo correo con la variable email

*<*input [ngModel]="email" name="correo"

Proteger las rutas de mi aplicación con guards

De nada me sirve autenticarme sino protejo las rutas de mi aplicación, esto lo hacemos con los guards

**Creemos el servicio de guards.**

ng g guard servies/guards/loginGuard --spec=false

nos creara un archive llamado login-guard.guard.ts

import { Injectable } from '@angular/core';

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class LoginGuardGuard implements CanActivate {

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

return true;

}

}

De esto no vamos a usar todas las importaciones asi que limpiemos el achivo y en el metodo canActivate regresemos un booleano , true si el usuario lo puede ver y un false sino lo puede ver. Debería quedar asi:

import { Injectable } from '@angular/core';

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

@Injectable({

providedIn: 'root'

})

export class LoginGuardGuard implements CanActivate {

canActivate(){

return true;

}

}

Implementemos este guard en el service.index, los guardse al final son servicios.

export { SettingsService } from './settings/settings.service';

export { SharedService } from './shared/shared.service';

export { SidebarService } from './shared/sidebar.service';

export { UsuarioService } from './usuario/usuario.service';

export { LoginGuardGuard } from './guards/login-guard.guard';

y ahora en el services.module.ts lo importaremos.

import { SettingsService, SidebarService, SharedService, UsuarioService, LoginGuardGuard } from './service.index';

y colocaremos en los prividers, con esto el servicio quedara funcional en toda la aplicación.

providers: [SettingsService, SidebarService, SharedService, UsuarioService, LoginGuardGuard],

coloquemos un mensaje de prueba.

})

export class LoginGuardGuard implements CanActivate {

canActivate(){

console.log("paso por el login guard");

return true;

}

}

Ahora en el pages.routes.ts llamaremos al loginGuad en una propiedad llamada canActivate después del pagesComponent. Deber importar el Loginguard

path: '',

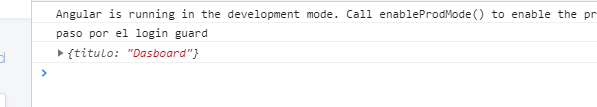
component: PagesComponent,

canActivate:[LoginGuardGuard],

children:[

{path: 'dashboard', component: DashboardComponent, data:{titulo:'Dasboard'}},

Guardamos y si estamos en el dashboard nos saldrá en la consola



Ahora como podemos saber si un unuario esta autenticado o no?

Vamos al usuario.services.ts y creemos un metodo que se llame estaLogeado si existe token es porque esta logeado.

@Injectable()

export class UsuarioService {

usuario : UsuariosModel;

token : string;

estaLogeado(){

return (this.token.length > 5) ? true : false;

}

Debemos inicializar las variables token y usuario para poder seguir eso lo haremos en un metod que llamaremos cargarStorage() que tendrá lo siguiente.

cargarStorage(){

if(localStorage.getItem('token')){

this.token = localStorage.getItem('token');

this.usuario = JSON.parse(localStorage.getItem('usuario'));

}else{

this.token = '',

this.usuario = null;

}

}

Y llamamos a esta función cada que el servicio se inicialice, por tando debo invocarla en el constructor.

Ahora regresemos al login-guard e injectemos el servicio de usuarios.

import { UsuarioService } from '../service.index';

export class LoginGuardGuard implements CanActivate {

constructor(

public \_usuarioService : UsuarioService

){

Ahora en el metodo canActivate usaremos ese servicio y si es bloqueado que me envie al login, acuérdate importar el Router.

import { CanActivate, Router } from '@angular/router';

canActivate(){

if(this.\_usuarioService.estaLogeado()){

console.log("paso el guard");

this.router.navigate(['/dashboard]);

}else{

console.log("bloqueado por el guard");

this.router.navigate(['/login']);

return false;

}

}

Con esto debería funcionar. Y redirigir al dashboad

Implementemos el logout.

Creemos el metodo en el servicio de usuarios. Este metodo lo que hara sera resetear el token y el usuario y borrara el locakStorage.

logout(){

this.usuario = null;

this.token = '';

localStorage.removeItem('token');

localStorage.removeItem('usuario');

this.router.navigate(['/login']);

}

Este servicio lo vamos a implementar en el header.component.ts ya que allí es donde se encuentra el cerrar sesión o el logout.

import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../../services/service.index';

@Component({

selector: 'app-header',

templateUrl: './header.component.html',

styles: []

})

export class HeaderComponent implements OnInit {

constructor(

public \_usuarioService : UsuarioService

) { }

ngOnInit() {

}

}

Ahora vayamos a la vista header.component.html y busquemos el logout para implementar el servicio.

*<*li*><*a href="#"*><*i class="fa fa-power-off"*></*i*>* Logout*</*a*></*li*>*

Quedaria de la siguiente manera. Con un evento click haciendo referencia al metodo logout del componente.

*<*li*><*a class="pointer" (click)="logout()"*><*i class="fa fa-power-off"*></*i*>* Logout*</*a*></*li*>*

Para que salga el icono de la fecha al tocar esta opción definamos la clase en los styles.css y luego en el elemento.

.pointer{

cursor: pointer !important;

}

Ahora el metodo del componente.

import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../../services/service.index';

@Component({

selector: 'app-header',

templateUrl: './header.component.html',

styles: []

})

export class HeaderComponent implements OnInit {

constructor(

public \_usuarioService : UsuarioService

) { }

ngOnInit() {

}

**logout(){**

**this.\_usuarioService.logout();**

**}**

}

Ahora hagamos lo mismo para la vista del sidebar donde también esta la opción de logout. Por lo que iremos al sidebar y haremos lo mismo.

Guardemos cambios en github

Git status

Git add .

Git commit –m “seccion 13 terminada”

Git push origin master

Git tag –a v2.0.0 –m “registro de user, login y logout”



Temas puntuales que aprenderemos en esta sección

Sección 14, Clase 166

Esta sección se enfoca en varios temas relacionados al perfil del usuario:

* Módulo de pipes
* Pipe para controlar la imagen a desplegar
* Subida de imagen desde el Front-end hasta el Back-end
* Crear el componente del perfil del usuario
* Notificar actualización de imagen
* Vista previa de la imagen seleccionada en tiempo real (sin subirla al backend)

Al finalizar, siempre haremos la creación del release tag en nuestro repositorio local y remoto

Creemos pipes que nos permitan controlar las imágenes del usuario.

Creemos un modulpo que trabaje con pipes.

ng g p pipes/imagen --spec=false

CREATE src/app/pipes/imagen.pipe.ts (201 bytes)

UPDATE src/app/app.module.ts (950 bytes)

Podemos observar que se actualize el app.module.ts agregando el pipe pero no lo quiero allí asi que removámoslo.

Ahora quiero crearme un modulo que me controle tosos los futuros pipes.

$ ng g m pipes/pipes --spec=false --flat

CREATE src/app/pipes/pipes.module.ts (189 bytes)

Vamos a el y quitemos el commonModule ya que no lo vamos a necesitar en esta parte.

Pero si importemos el imagenPipe y lo agregaremos a las declaraciones, añadamos un exports para decir que modulos se podrán usar fuera de este y coloquemos también el imagenPipe.

Debería quedar de la siguiente manera.

import { NgModule } from '@angular/core';

import { ImagenPipe } from './imagen.pipe';

@NgModule({

imports: [],

declarations: [

ImagenPipe

],

exports:[

ImagenPipe

]

})

export class PipesModule { }

este modulo deberemos importarlo en el lugar donde trabajemos con los pipes de imagen entonces seria en **pages.module.ts**, por el momento coloquemoslo alla en los imports e importemoslo.

imports:[

SharedModule,

PAGES\_ROUTES,

FormsModule,

ChartsModule,

PipesModule

]

**Comprobamos que el pipe creado es decir imagen.pipe funciona?**

Ayamos al pipe y hagamos lo siguiente. En el transform retornemos un string que diga funciona

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'imagen'

})

export class ImagenPipe implements PipeTransform {

transform(value: any, args?: any): any {

return 'FUNCIONA'

}

}

Vayamos al header.html y probémoslo allí en la parte del nombre del usuario

*<*div class="dw-user-box"*>*

*<*div class="u-img"*><*img src="assets/images/users/1.jpg" alt="user"*></*div*>*

*<*div class="u-text"*>*

*<*h4*>*Steave Jobs*</*h4*>*

*<*p class="text-muted"*>*varun@gmail.com*</*p*><*a href="pages-profile.html" class="btn btn-rounded btn-danger btn-sm"*>*View Profile*</*a*></*div*>*

*</*div*>*

Por interpolación usemos el pipe se usan de la siguiente manera si el pipe se llama imagen lo usamos asi.

{{‘Cualquier comentario’} | **imagen** }

Si guardamos y recargamos

*<*h4*>*{{'Cualquier Comentario' | imagen}}*</*h4*>*

Vamosa tener un error porque aun no importamos el modulo de pipes en el no ha sido importado en el modulo de los shared donde se encuentra el header, si lo importamos alla debería funcionar.



Podemos darnos cuenta que la palabra funciona sale que definimos en el imagen.pipe sale en la foto de perfil que estamso afectando

Ahora cambiemos la imagen del perfil de usuario en toda la page, regresemos al imagen.pipe.

Y en el transform recibamos los parámetros que deberíamos recibir en el path deñl backend.

<http://localhost:3000/img/usuarios/5b6711625c65701420cf56f5-883.jpg>

entonces en el transform llamaremos la url y recibiremos los parámetros necesarios.

Haremos una condición para cuando no llega la imagen, otra para cuando llega por cuenta de google y otra para una cuenta normal.

import { Pipe, PipeTransform } from '@angular/core';

import { URL\_SERVICIOS } from '../config/config';

@Pipe({

name: 'imagen'

})

export class ImagenPipe implements PipeTransform {

transform(img: string, tipo: string = 'usuario'): any {

let url = URL\_SERVICIOS+'/img';

*//si no viene imagen ejecuta la siguiente url*

if(!img){

return url +'/usuarios/xxx';

}

*//si viene un https es una imagen de google.*

if(img.indexOf('https')>=0){

return img;

}

*//pero sino es una imagende google*

switch(tipo){

case 'usuario':

url += '/usuarios/'+img

break;

case 'medico':

url += '/medicos/'+img

break;

case 'hospital':

url += '/hospitales/'+img

break;

default:

console.log("el tipo no existe, debe ser usuario, medicos u hospitales.");

return url +'/usuarios/xxx';

}

return url;

}

}

Ahora vayamos al header.component.html y obtengamnos el usuario que nos trae el localStogage.

Creamos una variable usuario, para que se cargue inmediatamente esaasignacion debe ser en el ngOnInit.

import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../../services/service.index';

import { UsuariosModel } from '../../models/usuarioModel';

@Component({

selector: 'app-header',

templateUrl: './header.component.html',

styles: []

})

export class HeaderComponent implements OnInit {

public img :string;

public usuario : UsuariosModel

constructor(

public \_usuarioService : UsuarioService

) { }

ngOnInit() {

this.usuario = this.\_usuarioService.usuario;

}

logout(){

this.\_usuarioService.logout();

}

}

Ya con esto podemos ir cambiando por interpolación los datos del header, cambiemos primero el nombre.

*<*div class="dw-user-box"*>*

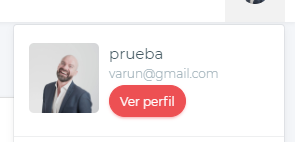
*<*div class="u-img"*><*img src="assets/images/users/1.jpg" alt="user"*></*div*>*

*<*div class="u-text"*><*h4*>*{{usuario.nombre}}*</*h4*>*

*<*p class="text-muted"*>*{{usuario.correo}}*</*p*><*a href="pages-profile.html" class="btn btn-rounded btn-danger btn-sm"*>*View Profile*</*a*>*

*</*div*>*

*</*div*>*



Podemos observar que ya traemos el usuario y el nombre

Ahora en la imagen usare el pipe de la siguiente manera

*<*div class="dw-user-box"*>*

*<*div class="u-img"***><*img [src]="usuario.img | imagen"** alt="user"*></*div*>*

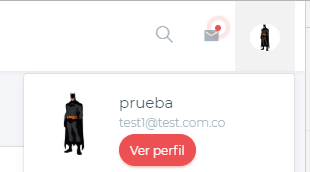
*<*div class="u-text"*>*

*<*h4*>*{{usuario.nombre}}*</*h4*>*

Y me trae la imagen



Cambiemos el resto.



Ahora cambiemos la del sidebar y hagamos lo mismo.

import { Component, OnInit } from '@angular/core';

import { SidebarService, UsuarioService } from '../../services/service.index';

import { UsuariosModel } from '../../models/usuarioModel';

@Component({

selector: 'app-sidebar',

templateUrl: './sidebar.component.html',

styleUrls: ['./sidebar.component.css']

})

export class SidebarComponent implements OnInit {

public usuario : UsuariosModel

constructor(

public \_sidebar:SidebarService,

public \_usuarioService : UsuarioService

) { }

ngOnInit() {

this.usuario = this.\_usuarioService.usuario;

}

logout(){

this.\_usuarioService.logout();

}

}

Al html del sidebar le agregamos el pipe imagen html del sidebar

*<*a class="has-arrow waves-effect waves-dark" href="#" aria-expanded="false"*><*img [src]="usuario.img | imagen" alt="user" */><*span class="hide-menu"*>*Steave Jobs *</*span*></*a*>*

Creemos ahora el component para el perfil del usuario. Este lo crearemos en pages

$ ng g c pages/profile --spec=false

CREATE src/app/pages/profile/profile.component.html (26 bytes)

CREATE src/app/pages/profile/profile.component.ts (273 bytes)

CREATE src/app/pages/profile/profile.component.css (0 bytes)

UPDATE src/app/pages/pages.module.ts (1873 bytes)

Par apoder accesar a esa ruta debemos definirlo en el pages.routes.ts

import { ProfileComponent } from './profile/profile.component';

lo importamos y creamos la ruta

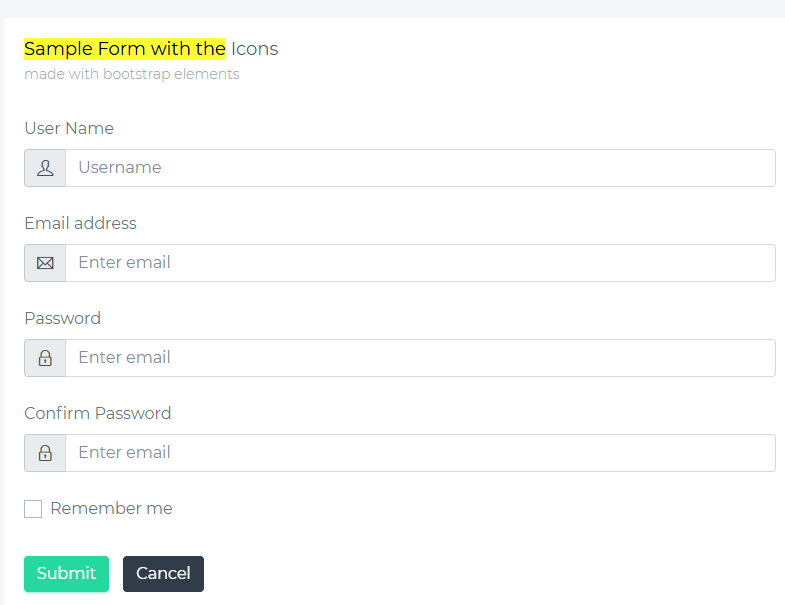
{path: 'perfil', component: ProfileComponent, data:{titulo:'Perfil de usuario'}},

Ahora hagamos un routerlink y routerlink active al sidebar.html en la opción de perfil.

*<*li*><*a routerLinkActive="active" routerLink="/perfil"*>*My Profile *</*a*></*li*>*

Cambiemos tambien el del header.html para ue redireccione al perfil

En el material del template busquemos un html llamado **form-layout.html,**  aquí encontraremos una gran cantidad de formulrios con estilos profesionales elijamos el que mas nos guste, en nuestro caso usemos el que se llama sample form with the icons



Tomemos ese código metámoslo en un row y peguemoslo en el perfil.component.html debería y adecuemoslo a nuestra necesidad. Debe quedar de la siguiente forma.

*<*div class="row"*>*

*<*div class="col-lg-6"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Perfil de usuario*</*h4*>*

*<*h6 class="card-subtitle"*>*{{usuario.nombre}}*</*h6*>*

*<*form class="form p-t-20"*>*

*<*div class="form-group"*>*

*<*label for="exampleInputuname"*>*User Name*</*label*>*

*<*div class="input-group"*>*

*<*div class="input-group-addon"*><*i class="ti-user"*></*i*></*div*>*

*<*input type="text" class="form-control" id="exampleInputuname" placeholder="Username"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group"*>*

*<*label for="exampleInputEmail1"*>*Email address*</*label*>*

*<*div class="input-group"*>*

*<*div class="input-group-addon"*><*i class="ti-email"*></*i*></*div*>*

*<*input type="email" class="form-control" id="exampleInputEmail1" placeholder="Enter email"*>*

*</*div*>*

*</*div*>*

*<*button type="submit" class="btn btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Guardar*</*button*>*

*</*form*>*

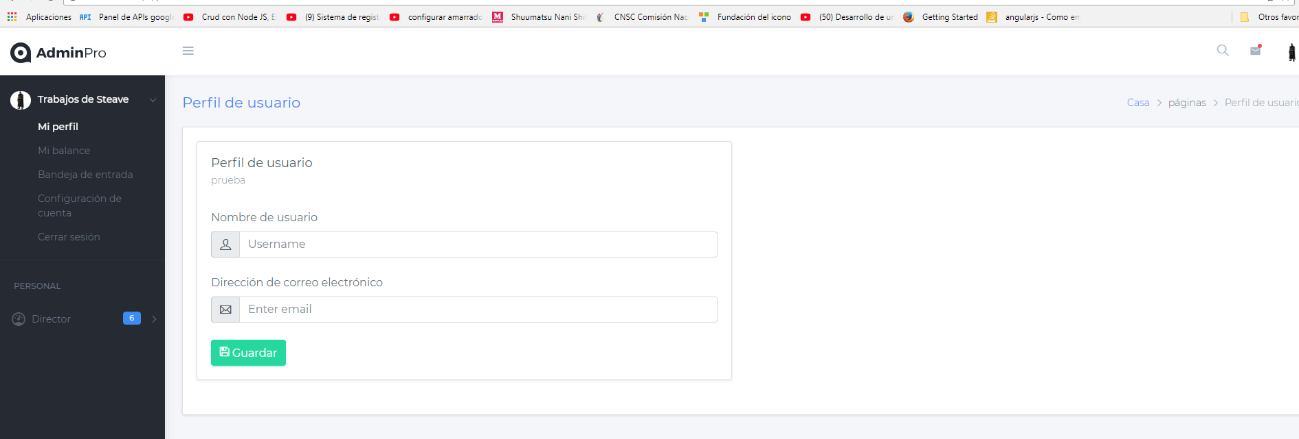
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Quedaria de la siguiente forma



Al lado coloquemos otra casa como la del formulario pero con la imagen

*<*div class="col-lg-5"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Fotografia del usuario*</*h4*>*

*<*h6 class="card-subtitle"*>*{{usuario.nombre}}*</*h6*>*

*<*img style="height: 240px;" [src]="usuario.img | imagen" alt=""*>*

*<*br*>*

*<*br*>*

*<*input type="file"*>*

*<*br*>*

*<*br*>*

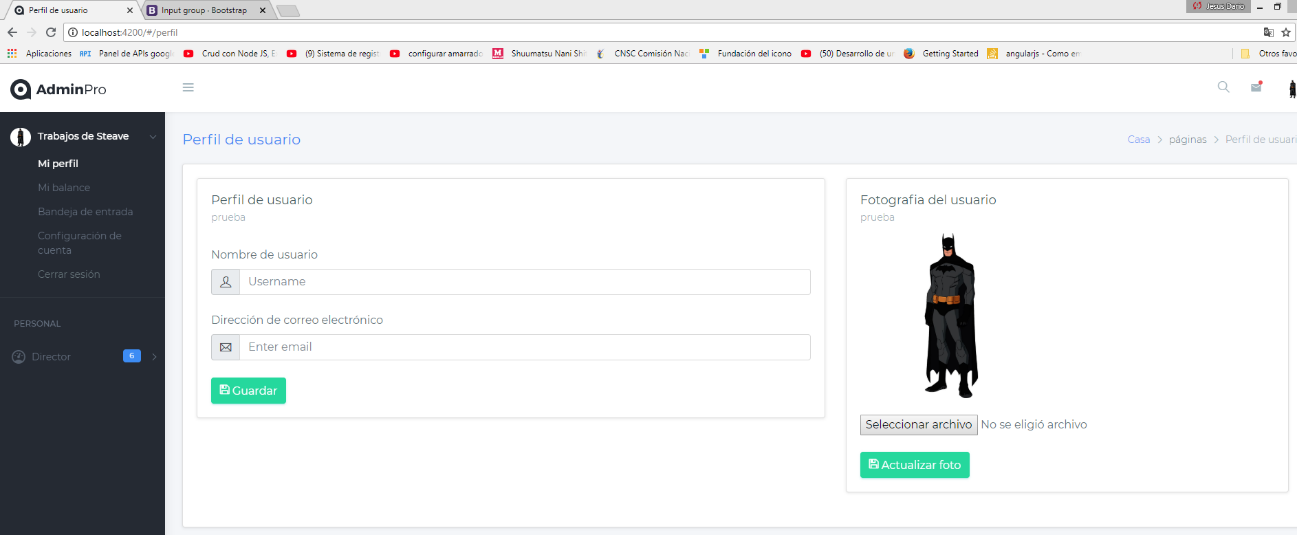
*<*button type="button" class="btn btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Actualizar foto*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

Se debería ver asi



**Actualicemos el perfil del usuario:**

Vayamos al profilecomponent.ts, carguemos el modelo de usuario y el servicio de usuarios e inicialicemos el servicio en el constructor y el usuario model en una propiedad publica.

import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../../services/service.index';

import { UsuariosModel } from '../../models/usuarioModel';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html'

})

export class ProfileComponent implements OnInit {

public usuario : UsuariosModel

constructor(

public \_usuarioService : UsuarioService

) { }

ngOnInit(){

this.usuario = this.\_usuarioService.usuario;

}

}

Ahora hagamos que en los input se coloque la información con ayuda el [ngModel]

1 colocar todos los names de los campos, colocarle el nombre al formulario en este caso #f=”ngForm”, activar las validaciones de html con ngNativeValidate.

*<*form ngNativeValidator #f="ngForm" class="form p-t-20"*>*

*<*div class="form-group"*>*

*<*label for="exampleInputuname"*>*User Name*</*label*>*

*<*div class="input-group"*>*

*<*div class="input-group-addon"*><*i class="ti-user"*></*i*></*div*>*

*<*input [ngModel]="usuario.nombre" name="nombre" type="text" class="form-control" id="exampleInputuname" placeholder="Username"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group"*>*

*<*label for="exampleInputEmail1"*>*Email address*</*label*>*

*<*div class="input-group"*>*

*<*div class="input-group-addon"*><*i class="ti-email"*></*i*></*div*>*

*<*input [ngModel]="usuario.correo" name="correo" type="email" class="form-control" id="exampleInputEmail1" placeholder="Enter email"*>*

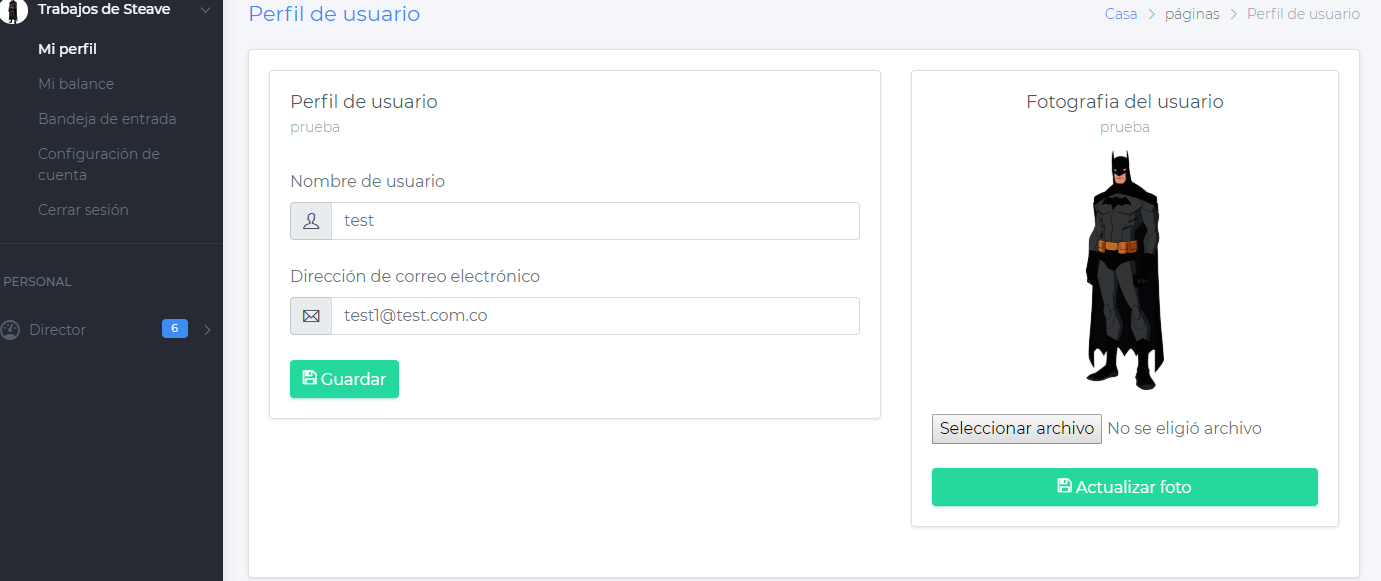
*</*div*>*

*</*div*>*

*<*button type="submit" class="btn btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Guardar*</*button*>*

*</*form*>*

Con esto ya nos estaría trayendo la información del usuario.



Ahora creemo un evento en el form (ngSubmit) que dispare un metodo guarar en el cmponente y que valide el formulario.

*<*form ngNativeValidator #f="ngForm" (ngSubmit)="guardar(f.value)" class="form p-t-20"*>*

Entonces en el component creamos el metodo y actualizamso con los dtos capturados

guardar(usuario:UsuariosModel){

this.usuario.nombre = usuario.nombre,

this.usuario.correo = usuario.correo

}

Ahora creemos el metodo en el usuario.services.ts para actualizar el usuario.

actualizarUsuario(usuario: UsuariosModel):Observable<any>{

let url = URL\_SERVICIOS+'/usuarios/'+usuario.\_id;

let params = JSON.stringify(usuario);

let headers = new HttpHeaders().set('Content-Type','application/json');

return this.\_http.put(url, params, {headers:headers});

}

Implementemos este servicio en el componente perfil en el metodo guardar.

actualizarUsuario(usuario: UsuariosModel){

let url = URL\_SERVICIOS+'/usuarios/'+usuario.\_id;

return this.\_http.put(url, usuario)

}

Ahora consumamoslo en el profile.component.ts debemos inicializar el valor del usuario por medio del servicio en el constructor y luego si haremos el resto.

constructor(

public \_usuarioService : UsuarioService

) {

this.usuario = this.\_usuarioService.usuario;

}

Aquí todo como va

import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../../services/service.index';

import { UsuariosModel } from '../../models/usuarioModel';

import swal from 'sweetalert';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html'

})

export class ProfileComponent implements OnInit {

public usuario : UsuariosModel

constructor(

public \_usuarioService : UsuarioService

) {

this.usuario = this.\_usuarioService.usuario;

}

ngOnInit(){}

guardar(usuario:UsuariosModel){

this.usuario.nombre = usuario.nombre;

this.usuario.correo = usuario.correo;

console.log(this.usuario);

this.\_usuarioService.actualizarUsuario(this.usuario)

.subscribe(resp =>{

console.log(resp);

})

}

}

Ahora deberemos actualizar el localStorage ya que sigue con los datos antiguos. Hacemos lo siguiente en el servicio

*// =======================Actualziar Datos de usuario======================= //*

actualizarUsuario(usuario: UsuariosModel):Observable<any>{

let url = URL\_SERVICIOS+'/usuarios/'+usuario.\_id;

console.log("paso1");

return this.\_http.put(url, usuario).pipe(

map((resp: any) =>{

console.log("voy por aqui");

let usuarioDB : UsuariosModel = resp.usuario

this.guardarStoage(usuarioDB.\_id, this.token, usuarioDB)

swal('Actualizado', 'Datos Actualizados Correctamente', 'success');

return true;

}));

}

**Ahora creemos un servicio para subir las imágenes.**

$ ng g s services/subir-archivo/subirArchivo --spec=false

CREATE src/app/services/subir-archivo/subir-archivo.service.ts (141 bytes)

Debemos exportarlo en service.index.ts

export { SubirArchivoService } from './subir-archivo/subir-archivo.service';

e importarlo en service.module.ts lo agrego a los providers y lo importo.

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { SettingsService,

SidebarService,

SharedService,

UsuarioService,

LoginGuardGuard,

SubirArchivoService } from './service.index';

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [

CommonModule,

HttpClientModule

],

providers: [SettingsService,

SidebarService,

SharedService,

LoginGuardGuard,

UsuarioService,

SubirArchivoService]

declarations: []

})

export class ServiceModule { }

ahora vayamos al servicio creado y creemos un metodo llamado subirArchivo que recibirá un archivo de tipo File, también el tipo colleccion a la que quiero subir la imagen es decir imagen de un usuarios, hospitales, que sera de tipo string y el id del elemento de la colección al que quiero subir la imagen también de tipo string.

subirArchivo(archivo : File, tipo:string, id:string){

}

import { Injectable } from '@angular/core';

import { URL\_SERVICIOS } from '../../config/config';

@Injectable()

export class SubirArchivoService {

constructor() { }

subirArchivo(archivo : File, tipo:string, id:string){

return new Promise ((resolve, reject)=>{

let formData = new FormData();

let xhr = new XMLHttpRequest();

formData.append('archivo', archivo, archivo.name);

xhr.onreadystatechange = function(){

if(xhr.readyState === 4){

if(xhr.status ===200){

console.log("imagen subida");

resolve(xhr.response);

}else{

console.log('fallo la subida');

reject(xhr.response);

}

}

}

let url = URL\_SERVICIOS+'/upload/'+tipo+'/'+id

xhr.open(POST, url, true);

xhr.send(formData)

});

}

}

Vamos al profile component y agreguemos algo para saber cuando una imagen es seleccionada para subirla.

Creemos una propiedad llamada imagenSubir de tipo file

export class ProfileComponent implements OnInit {

public usuario : UsuariosModel;

public imagenSubir :File

ahora detectemos la selección del input file. Vamos al html busquemos el input file y agreguemos el evento changue que ejecute el metodo selecciónImagen y que reciba un evento hagamos esas 2 cosas por lo pronto.

*<*input (change)="seleccionImagen($event)" type="file"*>*

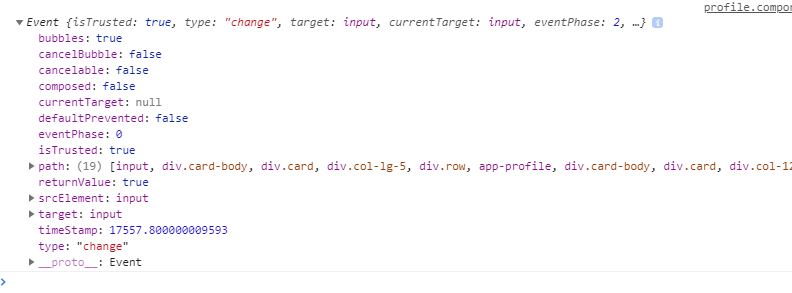
Creemos el metodo

seleccionImagen(event){

console.log(event);

}

Ahora si yo selecciono un archive deberia ejecutarse el clg del evento. Y nos daremos cuenta



Este evento tiene una propiedad llamada target que es el **input del formulario** que dentro tendrá la propiedad files y allí dentro tendremos la imagen.



Entonces modifiquemos el evento en el html para ue me traiga la imagen que sera la primera osea la posición cero porque solo se cargara una imagen.

*<*input (change)="seleccionImagen($event.target.files[0])" type="file"*>*

Con esto la respuesta del evento seran todos los datos de la imagen.



Hagamos la validación de si no existe el archivo y si si lo recibimos.

seleccionImagen(archivo){

this.imagenSubir= null;

if(!archivo){

return

}

*//si si viene archivo*

this.imagenSubir = archivo

}

Mostremos el previw de la imagen que queiro subir

En el html en el botón actualizar foto agregare un evento click que tendrá un metodo llamado cambiarImagen() pero el botón igualmente estará desabilitado sino hay una imagen a subir.

*<*button type="button" (click)="cambiarImagen()" [disabled]="!imagenSubir" class="btn btn-block btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Actualizar foto*</*button*>*

Vayamos al usuarioServices y creemos un metodo para cambiar la imagen desde allí ya que en este lugar es donde optengo los datos del usuario. Este metodo recibirá un archivo de tipo File y un id de tipo string

cambiarImagen(file: File, id: string){

}

Ahora importemos el servicio creado llamado subir-archivo.services.ts en usuarioServices.ts para consumirlo

import { SubirArchivoService } from '../subir-archivo/subir-archivo.service';

y lo inicializamos en el constructor

constructor(public \_http: HttpClient,

public router: Router,

public \_subirArchivoService: SubirArchivoService) {

this.cargarStorage();

console.log(this.cargarStorage());

}

Ahora llamamos al servicio en el metodo cambiar imagen. Y le pasamos los parámetros que el servicio importado necesita. Esto regresa una promesa por eso usamnos el then y el catch

*// =====================Subir Archivos de usuarios ==================================//*

cambiarImagen(archivo: File, id: string){

this.\_subirArchivoService.subirArchivo(archivo, 'usuarios', id)

.then(resp=>{

console.log(resp);

})

.catch(resp=>{

console.log(resp);

});

}

Ahora vayamos el profilecomponent.ts y en el metodo cambiarImagen llamemos al servicio de usuarios donde esta el metodo que necesitamos.

Ahora si seleccionamos una imagen y datos actualziar nos debería traer los datos del usuario incluidos la imagen. La respuesta que nos viene es un String no es un objeto entonces en el subir-archivo.services.ts haremos el parseo en el resolve. Con esto podemos trabajarlo como un objeto.

xhr.onreadystatechange = function(){

if(xhr.readyState === 4){

if(xhr.status ===200){

console.log("imagen subida");

resolve(JSON.parse(xhr.response));

}else{

console.log('fallo la subida');

reject(JSON.parse(xhr.response);

}

}

}

Coloquemos un sweet en el cambiarImagen del usuarioService para avisar que todo salio bien. Y actualicemos el storage.

**Guardemos los cambios en github**

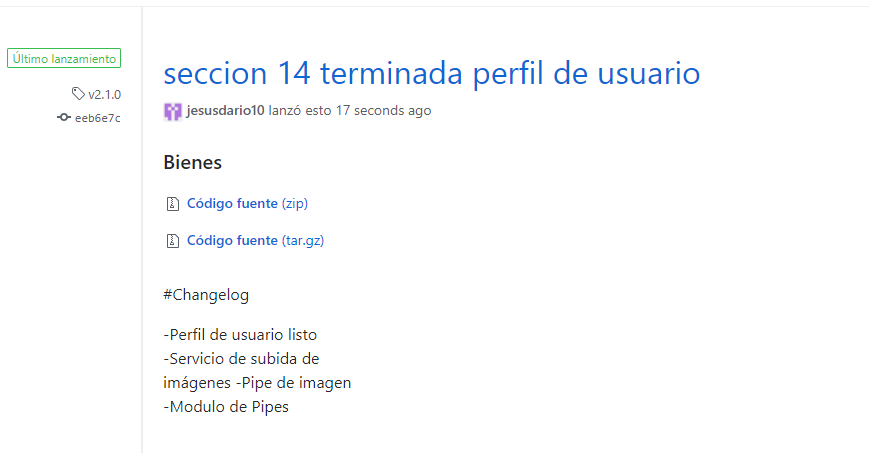
Git add .

Git commit –m “sección 14 terminada”

Git push para enviar al repositorio

Git tag –a v2.1.0 –m “perfil listo”

Git push –tags



Esta sección tiene 2 objetivos generales, el primero es crear un mantenimiento de usuarios completo y el segundo, es crear un componente re utilizable que nos permita subir fácilmente imágenes de Hospitales, Usuarios y Médicos.

Veremos sobre:

* Crear componente de usuarios
* Búsqueda de usuarios
* Borrar usuario
* Actualizar Rol del usuario
* Crear un modal para la subida de la imagen
* Emitir notificaciones de cambio en imagenes

Al finalizar, tendremos respaldos locales y remotos, junto a su release tag

**Necesitaremos una nueva opción en el menú:**

Vamos al servicio sidebar.services.ts que es donde tenemos configurado el menú.

Deberemos agregar un nuevo objeto para otra opción independiente a las que tenemos.

menu : any = [

{

titulo: "Principal",

icono: "mdi mdi-gauge",

submenu:[

{ titulo:"Dashboard", url:"/dashboard" },

{ titulo:"ProgressBar", url:"/progress" },

{ titulo:"Grafica1", url:"/graficas1" },

{ titulo:"Grafica2", url:"/graficas2" },

{ titulo:"Promesas", url:"/promesas" },

{ titulo:"RXJS", url:"/rxjs" }

]

},

{

titulo: "Mantenimientos",

icono: "mdi mdi-folder-lock-open",

submenu:[

{titulo:"Usuarios", url:'/usuarios'},

{titulo:"Hospitales", url:'/hospitales'},

{titulo:"Medicos", url:'/medicos'},

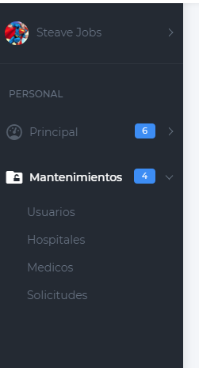
{titulo:"Solicitudes", url:'/solicitudes'}

]

}

];

Recargamos y tendremos una nueva opción en el menú



Vayamos al nopagefound.component.ts y carguemos el plugin que hay que poner manualmente que es el init\_plugins() ya que si caemos allí se va a quedar cargando para siempre sino lo ponemos. Hagamos esto antes de empezar

import { Component, OnInit } from '@angular/core';

declare function init\_plugin();

@Component({

selector: 'app-nopagefound',

templateUrl: './nopagefound.component.html',

styles: []

})

export class NopagefoundComponent implements OnInit {

constructor() { }

ngOnInit() {

init\_plugin()

}

}

Ahora si empecemos a crear la pagina del amntnimiento de usuarios para eso crearemos el componente usuarios ddentro de pages

$ ng g c pages/usuarios -is --spec=false

CREATE src/app/pages/usuarios/usuarios.component.html (27 bytes)

CREATE src/app/pages/usuarios/usuarios.component.ts (248 bytes)

UPDATE src/app/pages/pages.module.ts (1968 bytes)

Creamos la ruta para navegar a el eso es el pages.routes.ts

{path: 'usuarios', component: UsuariosComponent, data:{titulo:'Mantenimiento de Usuario'}},

Vayamos al html del componente usuarios.

Una caja para buscar usuarios una tabla de adminsitracion de usuarios.

*<*div class="row animated fadeIn"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*input #input type="text" class="form-control" placeholder="Buscar Usuario..."*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn"*>*

*<*div class="col-sm-12"*>*

*<*div class="alert alert-warning text-center"*>*

*<*strong*>*Cargando*</*strong*>*

*<*br*>*

*<*i class="fa fa-refresh fa-spin fa-2x"*></*i*>*

*<*br*>*

*<*span*>*Espere por favor*</*span*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h3 class="card-title"*>*Usuarios registrados (*<*small*>*123*</*small*>*)*</*h3*>*

*<*table class="table table-hover"*>*

*<*thead*>*

*<*tr*>*

*<*th*>*Imagen*</*th*>*

*<*th*>*Correo*</*th*>*

*<*th*>*Nombre*</*th*>*

*<*th*>*Role*</*th*>*

*<*th*>*Auth*</*th*>*

*<*th*></*th*>*

*</*tr*>*

*</*thead*>*

*<*tbody*>*

*<*tr*>*

*<*td*></*td*>*

*<*td*></*td*>*

*<*td*></*td*>*

*<*td*></*td*>*

*<*td*></*td*>*

*<*td*>*

*<*button class="btn btn-primary"*>* *<*i class="fa fa-save"*></*i*></*button*>*

*<*button class="btn btn-danger"*>* *<*i class="fa fa-trash-o"*></*i*></*button*>*

*</*td*>*

*</*tr*>*

*</*tbody*>*

*</*table*>*

*<*button class="btn btn-secondary"*>*Anteriores*</*button*>*-

*<*button class="btn btn-secondary"*>*Siguientes*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora mostremos a todos los usuarios. Vayamnos al usuarios.component.ts

Creemos una propiedad usuarios de tipo usuario model y digamos que es un array de usuarios.

Creems una propiedad desde de tipo number igualada a 0 y otra llamada totalregistros de tipo number también igual a 0.

Y también creemo un mmetodo llamado cargar usuarios que sera el que llame al servicio que crearemos ahora.

import { Component, OnInit } from '@angular/core';

import { UsuariosModel } from '../../models/usuarioModel';

@Component({

selector: 'app-usuarios',

templateUrl: './usuarios.component.html',

styles: []

})

export class UsuariosComponent implements OnInit {

usuarios : UsuariosModel[] = [];

desde : number = 0;

totalRegistros : number = 0;

constructor() { }

ngOnInit() {

}

cargarUsuarios(){

}

}

Ahora creemos el servicio que nos traerá los usuarios.

Vayamos a usuarios.services.ts y creemos el metodo cargarUsuarios que recibirá un parámetro desde inicializado en cero que nos ayudara con la segmentación de la información, traigamos la url y el resto es igual a un servicio.

*// =====================Cargar Usuarios ==================================//*

cargarUsuarios(desde: number = 0){

let url = URL\_SERVICIOS+'/usuarios?desde='+desde;

return this.\_http.get(url);

}

Vayamos al usuarios component y utilicemos este servicio a ver el resultado que nos arroja

ngOnInit() {

this.cargarUsuarios();

}

cargarUsuarios(){

this.\_usuarioService.cargarUsuarios(this.desde)

.subscribe(resp=>{

console.log(resp);

});

}

La respuesta que nos da es la siguiente. Me dice que tengo 15 registos pero me muesta los primeros 5 porque ese fue el metodo de extracción que use en el backend y aquí nos sirve porque cuando le demos siguiente nos mostrara los otros 5 y asi sucesivamente.



Ahora mostremos los datos en el html primero debemos cambiar algo en el metodo.

cargarUsuarios(){

this.\_usuarioService.cargarUsuarios(this.desde)

.subscribe((resp:any)=>{

console.log(resp);

this.totalRegistros = resp.total;

this.usuarios = resp.usuarios;

});

}

Ahora vayamos al html y mostremos los resultados. Debes de importar el

import { BrowserModule } from '@angular/platform-browser';

en el modulo principal de la aplicación para poder usar las directivas ngif, ngform etc.

*<*h3 class="card-title"*>*Usuarios registrados (*<*small*>*{{totalRegistros}}*</*small*>*)*</*h3*>*

*<*table class="table table-hover"*>*

*<*thead*>*

*<*tr*>*

*<*th*>*Imagen*</*th*>*

*<*th*>*Correo*</*th*>*

*<*th*>*Nombre*</*th*>*

*<*th*>*Role*</*th*>*

*<*th*>*Auth*</*th*>*

*<*th*></*th*>*

*</*tr*>*

*</*thead*>*

*<*tbody*>*

*<*tr \*ngFor="let usuario of usuarios"*>*

*<*td*></*td*>*

*<*td*>*{{usuario.correo}}*</*td*>*

*<*td*>*{{usuario.nombre}}*</*td*>*

*<*td*>*{{usuario.role}}*</*td*>*

*<*td*>*

*<*label classs="label label-dargen"*>*Google*</*label*>*

*<*label classs="label label-info"*>*Normal*</*label*>*

*</*td*>*

*<*td*>*

*<*button class="btn btn-primary"*>* *<*i class="fa fa-save"*></*i*></*button*>*

*<*button class="btn btn-danger"*>* *<*i class="fa fa-trash-o"*></*i*></*button*>*

*</*td*>*

*</*tr*>*

*</*tbody*>*

Ahora los botones de atrás y siguiente hagamos que funcionen.

*<*button (click)="cambiardesde(-5)" class="btn btn-secondary"*>*Anteriores*</*button*>*-

*<*button (click)="cambiardesde(5)" class="btn btn-secondary"*>*Siguientes*</*button*>*

Creemos ese metodo.

cambiarDesde(valor:number){

let desde = this.desde + valor;

console.log(desde);

if(desde>=this.totalRegistros){

return

}

if(desde<0){

return

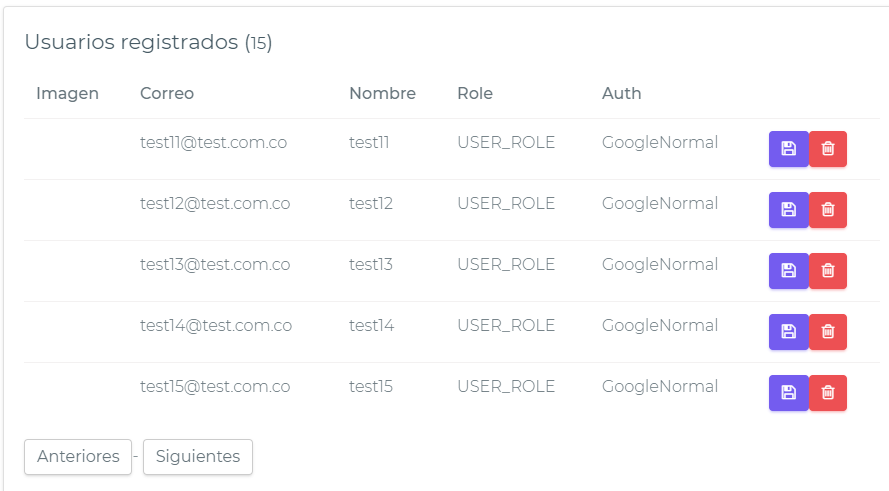
}

this.desde += valor

this.cargarUsuarios();

}

Ya los botones están funcionando.



Ahora hagamos la búsqueda de usuarios por el campo input funcione y que cuando cargue se remueva el loading.

Declaremos una propiedad llamada cargando de tipo boolean igual a true.

En el cargar usaurios colcoaremos el this.cargando sea igual a true. Y cuando termine de cargar que seal false.

cargarUsuarios(){

this.cargando = true;

this.\_usuarioService.cargarUsuarios(this.desde)

.subscribe((resp:any)=>{

console.log(resp);

this.totalRegistros = resp.total;

this.usuarios = resp.usuarios;

this.cargando = false;

});

}

Luego en el html juguemos con el \*ngIf

*<*div class="row animated fadeIn" \*ngIf="cargando"*>*

*<*div class="col-sm-12"*>*

*<*div class="alert alert-warning text-center"*>*

*<*strong*>*Cargando*</*strong*>*

*<*br*>*

*<*i class="fa fa-refresh fa-spin fa-2x"*></*i*>*

*<*br*>*

*<*span*>*Espere por favor*</*span*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn" \*ngIf="!cargando"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h3 class="card-title"*>*Usuarios registrados (*<*small*>*{{totalRegistros}}*</*small*>*)*</*h3*>*

**Creemos la function buscarUsuario**

Necesitamos el termino de busqueda que sea de tipo string

buscarUsuario(termino:string){

}

Ahora en el htm con el evento keyup le dire que cuando alguien precione una tecla se dispare el vento y me ejecute buscar usuario donde le enviare el valor del input como parámetro.

*<*input #input (keyup)="buscarUsuario(input.value)"

Ahora creemos un metodo en el servicio de usuarios para utilizer el backend.

*// =====================Buscar Usuarios ==================================//*

buscarUsuarios(termino: string):Observable<any>{

let url = URL\_SERVICIOS+'/busqueda/usuarios/'+termino;

return this.\_http.get(url).pipe(

map((resp:any)=>{

return resp.usuarios;

})

);

}

}

Implementemolo en el componente

buscarUsuario(termino:string){

console.log(termino);

let url = URL\_SERVICIOS+'/busqueda/usuarios/'+termino;

this.\_usuarioService.buscarUsuarios(termino)

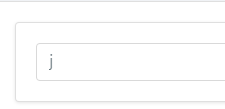
.subscribe((usuarios:any)=>{

console.log(usuarios);

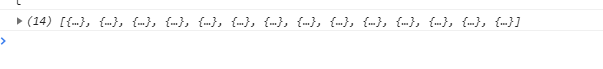
});

}

Ahora si yo presiono en el input la telca j debería traerme un solo elemento ya que tengo 1 con j



Si digo t tengo 14 elementos con la letra t



Ahora apliquemos esto al html para ue sea dinamico en la vista esto esta fácil, solo abra que actualziar el valor de la propiedad usuarios

buscarUsuario(termino:string){

console.log(termino);

let url = URL\_SERVICIOS+'/busqueda/usuarios/'+termino;

this.\_usuarioService.buscarUsuarios(termino)

.subscribe((usuarios:any)=>{

console.log(usuarios);

this.usuarios = usuarios;

});

}

Y listo si coloco una letra la vsita se actualzia automáticamente.

Validemos si el usuario borra y el parámetro termino queda vacio.

buscarUsuario(termino:string){

if(termino.length<=0){

this.cargarUsuarios();

return;

}

this.cargando = true;

console.log(termino);

let url = URL\_SERVICIOS+'/busqueda/usuarios/'+termino;

this.\_usuarioService.buscarUsuarios(termino)

.subscribe((usuarios:any)=>{

this.usuarios = usuarios;

this.cargando = false;

});

}

Ahora terminemos la parte del auth del html

*<*label \*ngIf="usuario.google" classs="alert alert-success"*>*Google*</*label*>*

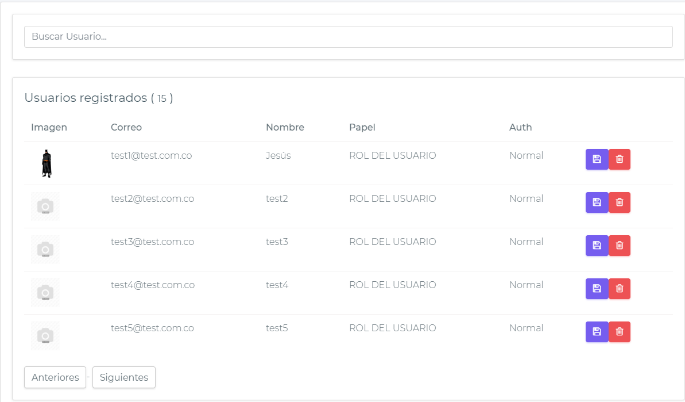
*<*label \*ngIf="!usuario.google" classs="alert alert-danger"*>*Normal*</*label*>*

Agreguemos la imagen

*<*td*>*

*<*img [src]="usuario.img | imagen" style="height: 50px;"*>*

*</*td*>*



Ahora borremos los usuarios.

Agreguemos el evento click al boton de borrar. Envio todo el usuario que hace click.

*<*button (click)="borrarUsuario(usuario)" class="btn btn-danger"*>* *<*i class="fa fa-trash-o"*></*i*></*button*>*

Creemos la function borrar usuario. Y hagamos una condición para que un usuario no pueda borrarse asi mismo.

borrarUsuario(usuario:UsuariosModel){

console.log(usuario);

if(usuario.\_id === this.\_usuarioService.usuario.\_id){

swal("No puede borrar usuario", "no se puede borrar asi mismo", "error");

return;

}

}

Ahora hagamos un condicional de confirmación para el borrado. Hayq ue declarar una variable swal de tipo eny antes de la creación de la clase

declare var swal:any;

borrarUsuario(usuario:UsuariosModel){

console.log(usuario);

if(usuario.\_id === this.\_usuarioService.usuario.\_id){

swal("No puede borrar usuario", "no se puede borrar asi mismo", "error");

return;

}

swal({

title: "Esta seguro?",

text: "OncEsta a punto de borrar a "+usuario.nombre,

icon: "warning",

buttons: true,

dangerMode: true,

})

.then((borrar) => {

if (borrar) {

swal("El usuario ha sido borrado", {

icon: "success",

});

} else {

swal("El usuario no sera borado");

}

});

}

Ahora creemos en el servicio una función para borrar los usuarios.

*// =====================Borrar Usuarios ==================================//*

borrarUsuario(id:string):Observable<any>{

let url = URL\_SERVICIOS+'/usuarios/'+id;

return this.\_http.delete(url).pipe(

map((resp:any)=>{

return resp.usuarios;

})

);

}

Ahora implementemos ese servicio en el componente.

borrarUsuario(usuario:UsuariosModel){

console.log(usuario);

if(usuario.\_id === this.\_usuarioService.usuario.\_id){

swal("No puede borrar usuario", "no se puede borrar asi mismo", "error");

return;

}

swal({

title: "Esta seguro?",

text: "OncEsta a punto de borrar a "+usuario.nombre,

icon: "warning",

buttons: true,

dangerMode: true,

})

.then((borrar) => {

if (borrar) {

this.\_usuarioService.borrarUsuario(usuario.\_id)

.subscribe((usuarios:any)=>{

swal("El usuario ha sido borrado", {

icon: "success",

});

});

} else {

swal("El usuario no sera borado");

}

});

}

**Actualiemos el Role del usuario.**

En el html de usuarios.component en la parte de role creemos un select con [(ngModel)] para que me traiga el role y también lo pueda cambiar.

*<*td*>*

*<*select name="role" [(ngModel)]="usuario.role" id="" class="form-control"*>*

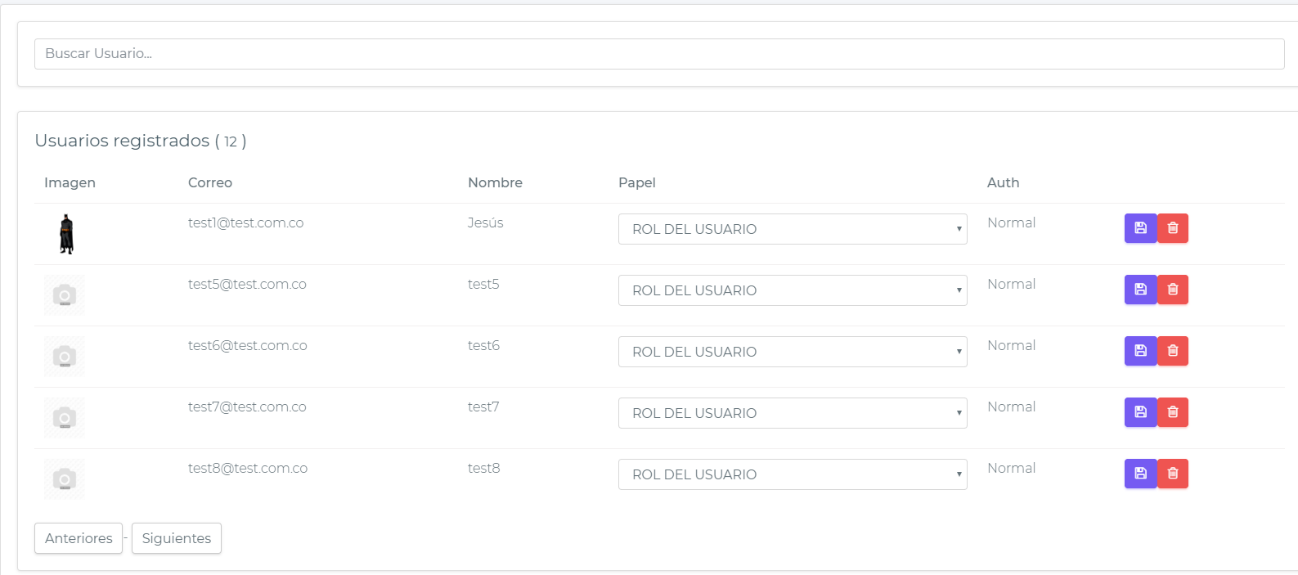
*<*option value="ADMIN\_ROLE"*>*ADMIN\_ROLE*</*option*>*

*<*option value="USER\_ROLE"*>*USER\_ROLE*</*option*>*

*</*select*>*

*</*td*>*

Se va aver asi



La idea es que seleccionemos un role le demos en guardar y ese cambio impacte la basede datos.

Vamos a crear un evento click en el botón que de guardar para que haga esto.

*<*button (click)="guardrUsuario(usuario)" class="btn btn-primary"*>* *<*i class="fa fa-save"*></*i*></*button*>*

Ahora vayamos al servicio de usuarios y hagamos una pequeña modificación al actualizar.

*// =======================Actualziar Datos de usuario======================= //*

actualizarUsuario(usuario: UsuariosModel):Observable<any>{

let url = URL\_SERVICIOS+'/usuarios/'+usuario.\_id;

console.log("paso1");

return this.\_http.put(url, usuario).pipe(

map((resp: any) =>{

if(usuario.\_id===this.usuario.\_id){

console.log("voy por aqui");

let usuarioDB : UsuariosModel = resp.usuario

this.guardarStoage(usuarioDB.\_id, this.token, usuarioDB)

}

swal('Actualizado', 'Datos Actualizados Correctamente', 'success');

}));

}

Ahora vayamos al componentey utilicemos ese servicio

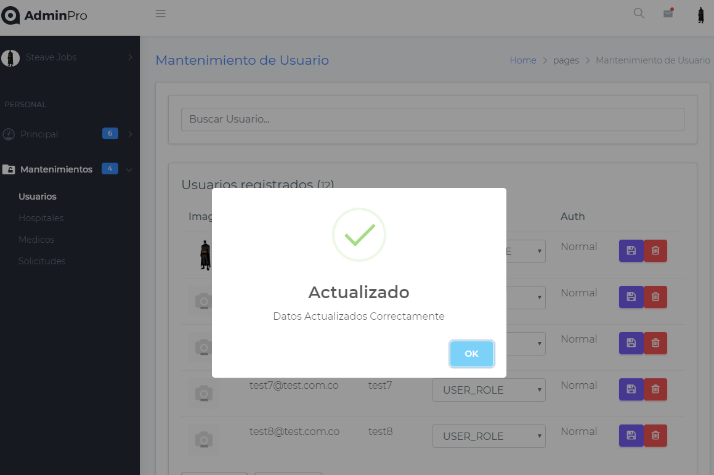
guardrUsuario(usuario: UsuariosModel){

this.\_usuarioService.actualizarUsuario(usuario)

.subscribe();

}

Listo tenemos el usuario actualizado



Cremos un componente especializado para la subida de imágenes que nos permita cambiar la imagen de los usuarios desde el rol admin por si suben algo obseno.

Entonces en la vista de actual trabajaremos una funcionalidd que al hacer click en la imagen me aparesca un modal que me permita seleccionar una imagen, subirla y actualziar la base de datos.

Creemos un componente llamado modalUpload y que se cargue en el pages.module.ts

$ ng g c components/modalUpload --spect=false -is

CREATE src/app/components/modal-upload/modal-upload.component.html (31 bytes)

CREATE src/app/components/modal-upload/modal-upload.component.ts (259 bytes)

UPDATE src/app/app.module.ts (1006 bytes)

Debemos quitar este modulo del app.module.ts ya que se cargo alli y lo necesitamos es en pages. Asi que quitémoslo de app.module.ts y coloquemoslo en pages.module.ts en declaratiosn.

Importamos en pages.module.ts

import { ModalUploadComponent } from '../components/modal-upload/modal-upload.component';

agregamos al array de declaratios

@NgModule({

declarations:[

DashboardComponent,

ProgressComponent,

Graficas1Component,

PagesComponent,

IncrementadorComponent,

GraficoDonaComponent,

Graficas2Component,

GraficoRadarComponent,

AccountSettingsComponent,

PromesasComponent,

RxjsComponent,

ProfileComponent,

UsuariosComponent,

ModalUploadComponent

],

Como vamos a utilizar el componente ModalUploadComponent en todos los componentes de pages, vamos a agregar su etiqueta en el pages.component.html, es decir es la estructura html principal del pages.

*<!-- ===============Modal Upload========================= -->*

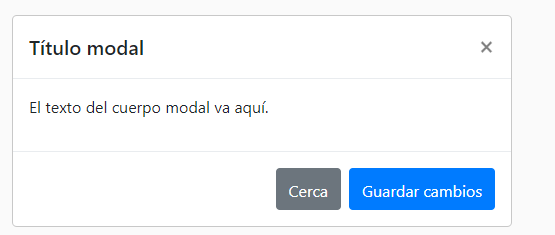
*<*app-modal-upload*></*app-modal-upload*>*

Hagamos una prueba para ver si todo esta bien, en el constructor de modal-upload hagamos un console.log que diga modal listo y revisemos si en todos los pages sale el mensaje modal listo



Efectivamente

Busquemos en bootstrap y en ladocumentacion busquemos modal y copiemos en html para crear esta caja.



<div class="modal" tabindex="-1" role="dialog">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Modal title</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Modal body text goes here.</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary">Save changes</button>

<button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

Este código lo pegaremos en modal-upload.component.html pero con una pequeña modificación, los modal vienen ocultos por defecto asiq ue para que aparezcan vamos a colocar un style=”display:block;”

*<*div class="modal" **style="display: block;"** tabindex="-1" role="dialog"*>*

*<*div class="modal-dialog" role="document"*>*

*<*div class="modal-content"*>*

*<*div class="modal-header"*>*

*<*h5 class="modal-title"*>*Modal title*</*h5*>*

*<*button type="button" class="close" data-dismiss="modal" aria-label="Close"*>*

*<*span aria-hidden="true"*>*&times;*</*span*>*

*</*button*>*

*</*div*>*

*<*div class="modal-body"*>*

*<*p*>*Modal body text goes here.*</*p*>*

*</*div*>*

*<*div class="modal-footer"*>*

*<*button type="button" class="btn btn-primary"*>*Save changes*</*button*>*

*<*button type="button" class="btn btn-secondary" data-dismiss="modal"*>*Close*</*button*>*

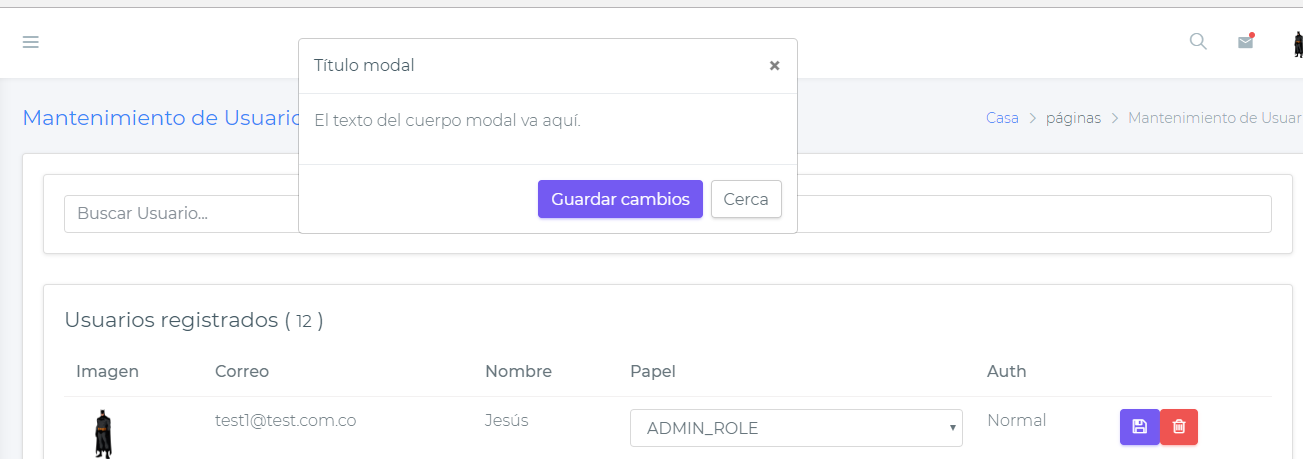
*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Podemos observar que el modal ya apareció



Es de color blanco hagamos un par de cambios para que se vea mas oscuro y que este bloqueado todo lo que este por fuera.

*<*div class="fondo-negro"*>*

*<*div class="modal" style="display: block;" tabindex="-1" role="dialog"*>*

*<*div class="modal-dialog" role="document"*>*

*<*div class="modal-content"*>*

*<*div class="modal-header"*>*

*<*h5 class="modal-title"*>*Modal title*</*h5*>*

*<*button type="button" class="close" data-dismiss="modal" aria-label="Close"*>*

*<*span aria-hidden="true"*>*&times;*</*span*>*

*</*button*>*

*</*div*>*

*<*div class="modal-body"*>*

*<*p*>*Modal body text goes here.*</*p*>*

*</*div*>*

*<*div class="modal-footer"*>*

*<*button type="button" class="btn btn-primary"*>*Save changes*</*button*>*

*<*button type="button" class="btn btn-secondary" data-dismiss="modal"*>*Close*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

La cllase aplicada seria.

.fondo-negro{

background-color: rgba(0, 0, 0, 0.4);

position: fixed;

top: 0px;

left: 0px;

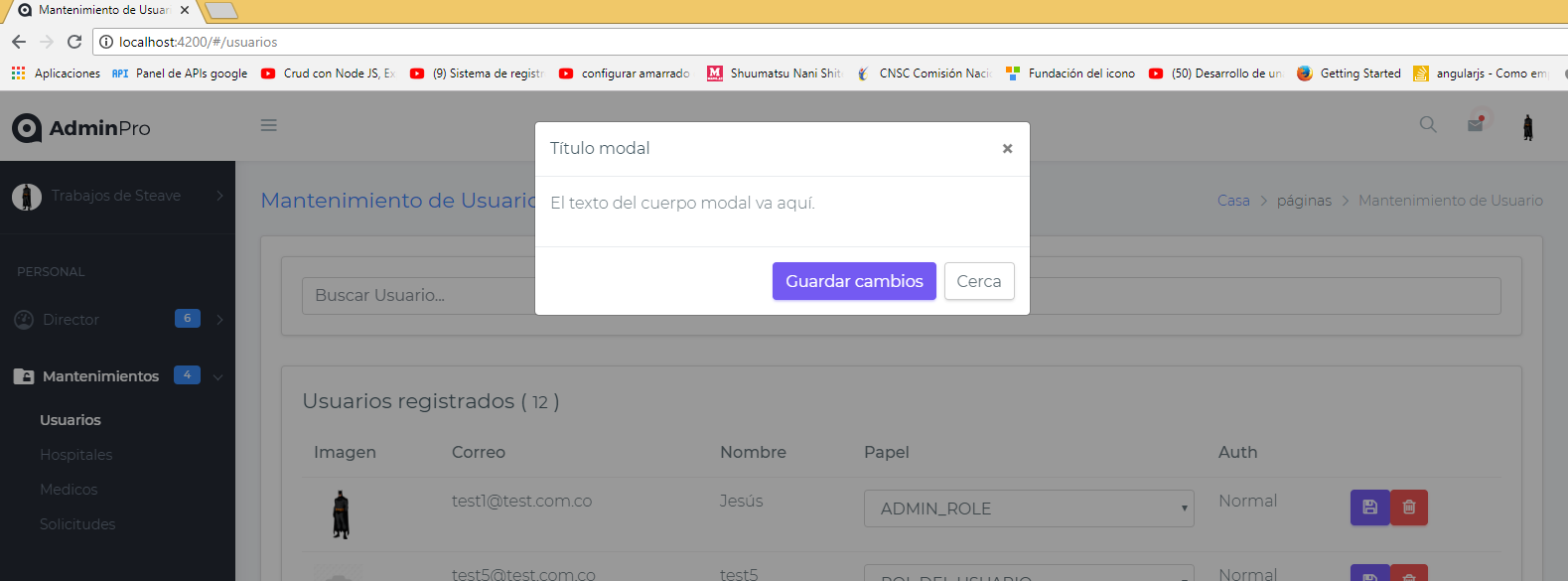
width: 100%;

height: 100%;

z-index: 9999;

}

Y se veria de la siguiente forma.



Como podemos ocultar el modal? Utilicemos una clase para ocultarlo, creemos la clase se llamara oculto. Con un didplay none para que la oculte. Y vamos a agrergarla al html.

Pero antes de eso vamos a darle funcionalidad a los botones de cerrar del modal.

Para esto creemos en el componente una propiedad llamad oculto de tipo string

export class ModalUploadComponent implements OnInit {

oculto : string;

luego a los botones cerrar del html les voy a agregar un evento click que al darle click cambiara el valor de esa definida oculto a oculto = oculto.

*<*button (click)="oculto='oculto'" type="button" class="close" data-dismiss="modal" aria-label="Close"*>*

Luego con un [(ngClass)]=”oculto” en el contenedor principal enlazaremos con la propiedad oculto y de este modo agregar la clase oculto.

Si probamos todo va a funcionar.

Ahora terminemos de llenar la pagina del modal con lo que necesitamos.

*<*div class="modal-body"*>*

*<*img style="height: 240px;" [src]="usuario.img | imagen" alt=""*>*

*<*br*><*br*>*

*<*input name="achivo" (change)="seleccionImagen($event.target.files[0])" type="file"*>*

*</*div*>*

Ahora creemos la función de selección de imagen que se dispara en el evento change

Esta función necesitara antes declarar dos propiedades

imagenSubir: File;

imagenTemp : string;

seleccionImagen(archivo){

this.imagenSubir= null;

if(!archivo){

return

}

*//este if es para comprobar si el archivo es una imagen*

if(archivo.type.indexOf('image')<0){

swal('Solo imagenes', 'El archivo selecconado no es una imagen', 'error');

this.imagenSubir = null;

return;

}

*//si si viene archivo*

this.imagenSubir = archivo

*//cargar la imagen temporal*

let reader = new FileReader();

let urlImagenTemp = reader.readAsDataURL(archivo);

reader.onload=()=>this.imagenTemp = reader.result;

}

El html a este punto deberíamos tenerlo asi:

*<*div class="fondo-negro fadeIn" [ngClass]="oculto"*>*

*<*div class="modal" style="display: block;" tabindex="-1" role="dialog"*>*

*<*div class="modal-dialog" role="document"*>*

*<*div class="modal-content"*>*

*<*div class="modal-header"*>*

*<*h5 class="modal-title"*>*Modal title*</*h5*>*

*<*button (click)="oculto='oculto'" type="button" class="close" data-dismiss="modal" aria-label="Close"*>*

*<*span aria-hidden="true"*>*&times;*</*span*>*

*</*button*>*

*</*div*>*

*<*div class="modal-body" *>*

*<*div style="text-align: center"*>*

*<*img \*ngIf="!imagenTemp" style="height: 240px;" [src]="'xxx' | imagen" alt=""*>*

*<*img \*ngIf="imagenTemp" style="height: 240px;" [src]="imagenTemp" alt=""*>*

*</*div*>*

*<*br*>*

*<*input name="achivo" (change)="seleccionImagen($event.target.files[0])" type="file"*>*

*</*div*>*

*<*div class="modal-footer"*>*

*<*button type="button" class="btn btn-primary"*>*Subir Imagen*</*button*>*

*<*button (click)="oculto='oculto'" type="button" class="btn btn-secondary" data-dismiss="modal"*>*Cerrar*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Ahora desabilitemos el boton de subir imagen si no existe una imagen seleccionada.

*<*button (click)="subirImagen()" disabled="!imagenTemp" type="button" class="btn btn-primary"*>*Subir Imagen*</*button*>*

Probemos creando la function subirImagen y hagamos un clg

Perfecto todo va bien.

Ahora debemos hacer el proceso de carga de imágenes, ya este proceso lo tenemos hecho en el servicio subir archivo que refcibe el archivo el tipo seria usuario y el id de ese usuario. Entonces en el constuctor hagamos la inyección del servicio subir-archivo pero ahora como k podemos obtener el usuario?

Para eso crearemos otro servicio dentro del modal-upload que medie entre la comunicación de los componetes con el componente modal-upload.component.

Empecemos a crear ese servicio

$ ng g s components/modal-upload/modalUpload --spec=false

CREATE src/app/components/modal-upload/modal-upload.service.ts (140 bytes)

Ester servicio deberemos importarlo en services.module.ts como es un servicio lo colocamos en los providers.

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { SettingsService,

SidebarService,

SharedService,

UsuarioService,

LoginGuardGuard,

SubirArchivoService } from './service.index';

import { HttpClientModule } from '@angular/common/http';

import { ModalUploadService } from '../components/modal-upload/modal-upload.service';

@NgModule({

imports: [

CommonModule,

HttpClientModule

],

providers: [SettingsService,

SidebarService,

SharedService,

LoginGuardGuard,

UsuarioService,

SubirArchivoService,

ModalUploadService],

declarations: []

})

export class ServiceModule { }

ahora importemos este Nuevo servicio en el modal-upload.component.ts

constructor(

public \_subirArchivoService: SubirArchivoService,

public \_modalUploadService: ModalUploadService

) {

console.log("modal listo");

}

en el constructor del modal-upload.services.ts probemos si todo funciona con un clg.

Pérfecto vamos bien.

Que es lo que queremos hacer con el servicio de **modalUploadServices**?

1-poder recibir el tip de archivo que quiero subir es decir si es un hospital, un usuario o un medico

2-tambien traer el id de ese tipo que quiero subir osea el id del usuario, el id del medico o del hospital.

Por tanto declararemos en el modalUploadServices un tipo de tipo string y un id de tipo string,

otra cosa que podemos controlar mediante este servicio es si el modal esta oculto o no entonces la propiedad sera oculto de tipo string.

Otra cosa que podemos controlar mediante este servicio es poder emitir algo para que los otros componentes donde usan el servicio del modal se puedan suscribir para estar escuchando cuando se sube la imagen. Esta es la parte crucial para poder notificar del modal a las otras patallas o componentes que ya se subio una imagen.

Esto lo controlaremos con una propiedad llamada notificación n que sera un new EventEmitter que emitirá un dato de tipo any que sera el objecto respuesta del servicio de carga de imagen deberemos hacer una importación como lo vemos a continuación.

import { Injectable, **EventEmitter** } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class ModalUploadService {

public id :string;

public tipo :string;

public oculto:string="";

**public notificacion = new EventEmitter<any>();**

constructor() {

console.log(" esta listo el servicio modal");

}

}

Creemos un funcion llamada ocultar modal y otra mostrar modal

import { Injectable, EventEmitter } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class ModalUploadService {

public id :string;

public tipo :string;

public oculto:string="";

public notificacion = new EventEmitter<boolean>();

constructor() {

console.log(" esta listo el servicio modal");

}

ocultarModal(){

}

mostrarModal(){

}

}

La función mostrar modal sera la encargada de hacer que aparesca el modal y eso lo haremos eso?.

Como el modal debe estar oculto le agregaremos un valor a la propiedad oculto de oculto.

import { Injectable, EventEmitter } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class ModalUploadService {

public id :string;

public tipo :string;

public oculto:string="oculto";

regresemos al modalupload.component y esa propiedad no la vamos a manejar desde el componente sino desde el servicio eliminemosla de aca. Y usémosla desde el mismo servicioen el html de esta forma. Si recargamos la pagina ya no debería verse el modal. De mono que si yo elimino el valor de la propiedad oculto en el servicio el modal pueda mostrarse.

Entonces regresemos al modal-upload.services.ts y configuremos los etodos ocultarmodal y mostrar modal para que hagan esto.

A la función de mostrar el modal le agregaremos 2 párametros, el tipo de collection y el id de esa collection las inicializamos las dos variables

A la función ocultar deberemos inicializar las variables tipo y id en null para que no queden guardados esos valores.

import { Injectable, EventEmitter } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class ModalUploadService {

public id :string;

public tipo :string;

public oculto:string="oculto";

public notificacion = new EventEmitter<boolean>();

constructor() {

console.log(" esta listo el servicio modal");

}

ocultarModal(){

this.oculto ="oculto"

this.tipo= null;

this.id = null;

}

mostrarModal(tipo:string, id:string){

this.oculto="";

this.tipo= tipo;

this.id = id;

}

}

A la función de mostrar el modal le agregaremos 2 párametros, el tipo de collection y el id de esa collection las inicializamos las dos variables.

Ahora en modalUploadComponent.html haremos un cambio en las opciones de cerrar para ocultar el modal. En los botones en el evento click ejecutaremos una función llamada cerrarmodal(). Que estará en modaluploadComponent.ts

*<*button (click)="cerrarModal()"

Esa función a su ves inicializara las variables imagenTemp en null e imagenSubir en null y usara el servicio modalUploadServices para usar el metodo ocultarModal del servicio.

cerrarModal(){

this.imagenTemp=null;

this.imagenSubir=null;

this.\_modalUploadService.ocultarModal()

}

Ahora aquí mismo en el modalUploadComponent.ts creemos el metodo de subir imagen que usara el servicio de subirArchivosservices para subirla y eso me devolverá una promesa.

subirImagen(){

this.\_subirArchivoService.subirArchivo(this.imagenSubir,

this.\_modalUploadService.tipo,

this.\_modalUploadService.id)

.then(resp=>{

console.log(resp);

this.\_modalUploadService.notificacion.emit(resp);*//emito respuesta*

this.\_modalUploadService.ocultarModal();*//oculto el modal*

})

.catch(err=>{

console.log("srror en al carca...");

});

}

Ahora vayamos al html de usuarios.component.html donde esta la lista de usuarios y en la imagen creemos un evento click para que dispare una funcion mostrarModal en la que enviaremos el id del usuario, ahora vayamos a usuarios.component.ts y para crear esa funcion que usara el servicio ModalUploadServices para disparar la función que hace eso.

*<*td*>*

*<*img (click)="mostrarModal()" [src]="usuario.img | imagen" style="height: 50px;" class="pointer"*>*

*</*td*>*

importando el servicio y creando la función.

import { ModalUploadService } from '../../components/modal-upload/modal-upload.service';

constructor(

public \_usuarioService : UsuarioService,

public \_modalUploadService: ModalUploadService

) { }

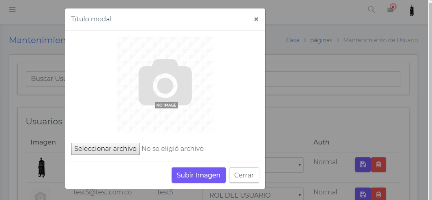
Creando al function.

mostrarModal(id:string){

this.\_modalUploadService.mostrarModal('usuarios', id);

}

Ahora si le doy click a la imagen deberia desplegarse el modal.



Si yo subo una imagen la sube correctamente



Pero no me actualiza la pantalla , para corregir esto en el ngOnInit nos suscribiremos a cualquier emicion que se haga en modalUploadServices.

ngOnInit() {

this.cargarUsuarios();

this.\_modalUploadService.notificacion

.subscribe(resp=>{

this.cargarUsuarios()

})

}

Con esto actualiza, perfecto.

Guardemos los cambios en github

Git status

Gut add .

Git commit –m “seccion 15 terminada”

Git push

Git tag

Git tag –a v2.2.0 –m “seccion 15 lista usuarios y modal de carga”

Git push - -tags



Esta sección tiene por objetivo poner a prueba todo lo aprendido hasta el momento, hay una tarea bastante grande que pueden demorar mínimo unos 20 minutos en hacerla  (si no se equivocan en nada)

Pero puntualmente, estos son los temas:

* Explicación de la tarea
* Documentos de la tarea
* Detalles sobre el pipe de las imagenes
* Resolución de la tarea
* Componente de Médico y Médicos
* Borrar médicos
* Crear nuevo médico
* Mostrar información en base a una selección de un hospital
* Cargar médico
* Actualizar registro de un médico

Al finalizar, tendremos respaldo en GitHub, release tag y commits locales para proteger nuestro trabajo

**hacer**

**crear componente hospitales**

$ ng g c pages/hospitales --spec=false -is

CREATE src/app/pages/hospitales/hospitales.component.html (29 bytes)

CREATE src/app/pages/hospitales/hospitales.component.ts (254 bytes)

UPDATE src/app/pages/pages.module.ts (2279 bytes)

**Crear servicio de hospitals ubicado en services/hospital/hospital.services.ts**

$ ng g s services/hospital/hospital --spec=false -is

CREATE src/app/services/hospital/hospital.service.ts (137 bytes)

Agregando el servicio al indexservices el servicio creado.

export { SettingsService } from './settings/settings.service';

export { SharedService } from './shared/shared.service';

export { SidebarService } from './shared/sidebar.service';

export { UsuarioService } from './usuario/usuario.service';

export { LoginGuardGuard } from './guards/login-guard.guard';

export { SubirArchivoService } from './subir-archivo/subir-archivo.service';

export { HospitalService } from './hospital/hospital.service';

lo importamos en services.module.ts e incluimos en los providers

providers: [SettingsService,

SidebarService,

SharedService,

LoginGuardGuard,

UsuarioService,

SubirArchivoService,

ModalUploadService,

HospitalService],

Componente relacionado de médicos

Creación

$ ng g c pages/medicos --spec=false -is

CREATE src/app/pages/medicos/medicos.component.html (26 bytes)

CREATE src/app/pages/medicos/medicos.component.ts (245 bytes)

UPDATE src/app/pages/pages.module.ts (2370 bytes)

Creemos uno adicional para la edicion de un medico

$ ng g c pages/medicos/medico --spec=false -is --flat

CREATE src/app/pages/medicos/medico.component.html (25 bytes)

CREATE src/app/pages/medicos/medico.component.ts (242 bytes)

UPDATE src/app/pages/pages.module.ts (2370 bytes)

Creemos el servicio para los medicos

$ ng g s services/medico/medico --spec=false

CREATE src/app/services/medico/medico.service.ts (135 bytes)

Añadamoslo al services.index y luego al services

export { SettingsService } from './settings/settings.service';

export { SharedService } from './shared/shared.service';

export { SidebarService } from './shared/sidebar.service';

export { UsuarioService } from './usuario/usuario.service';

export { LoginGuardGuard } from './guards/login-guard.guard';

export { SubirArchivoService } from './subir-archivo/subir-archivo.service';

export { HospitalService } from './hospital/hospital.service';

export { MedicoService } from './medico/medico.service'

ahora al services.module.ts

providers: [SettingsService,

SidebarService,

SharedService,

LoginGuardGuard,

UsuarioService,

SubirArchivoService,

ModalUploadService,

HospitalService,

MedicoService],

Creemos las rutas.

{path: 'medicos', component: MedicosComponent, data:{titulo:'Mantenimiento de Medicos'}},

{path: 'medico/:id', component: MedicoComponent, data:{titulo:'Mantenimiento de Medicos'}},

Creemos ahora la vista del componente medico

*<*div class="row animated fadeIn"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*input #input (keyup)="buscarMedico(input.value)" type="text" class="form-control" placeholder="Buscar Hospital..."*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div style="text-align: center" class="card-body"*>*

*<*button (click)="crearMedico()" type="button" class="btn waves-effect waves-light btn-rounded btn-primary"*><*font style="vertical-align: inherit;"*><*font style="vertical-align: inherit;"*>*Crear hospital*</*font*></*font*></*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn" \*ngIf="cargando"*>*

*<*div class="col-sm-12"*>*

*<*div class="alert alert-warning text-center"*>*

*<*strong*>*Cargando*</*strong*>*

*<*br*>*

*<*i class="fa fa-refresh fa-spin fa-2x"*></*i*>*

*<*br*>*

*<*span*>*Espere por favor*</*span*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="row animated fadeIn" \*ngIf="!cargando"*>*

*<*div class="col-12"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h3 class="card-title"*>*Medicos registrados (*<*small*>*{{\_medicoServices.totalMedicos}}*</*small*>*)*</*h3*>*

*<*table class="table table-hover"*>*

*<*thead*>*

*<*tr*>*

*<*th*>*Imagen*</*th*>*

*<*th*>*Nombre*</*th*>*

*<*th*>*Hospital*</*th*>*

*<*th*></*th*>*

*</*tr*>*

*</*thead*>*

*<*tbody*>*

*<*tr \*ngFor="let medico of medicos"*>*

*<*td*>*

*<*img (click)="actualizarImagen(medico)" [src]="medico.img | imagen:'medico'" style="height: 50px;" class="pointer"*>*

*</*td*>*

*<*td*>*

*<*input name="nombre" [(ngModel)]="medico.nombre" type="text" class="form-control" placeholder="Nombre del Hospital"*>*

*</*td*>*

*<*td*>*{{medico.hospital.nombre}}*</*td*>*

*<*td*>*

*<*button [routerLink]="['/medico', medico.\_id]" class="btn btn-primary"*>* *<*i class="fa fa-save"*></*i*></*button*>*

*<*button (click)="borrarMedico(medico)" class="btn btn-danger"*>* *<*i class="fa fa-trash-o"*></*i*></*button*>*

*</*td*>*

*</*tr*>*

*</*tbody*>*

*</*table*>*

*<*button (click)="cambiarDesde(-5)" class="btn btn-secondary"*>*Anteriores*</*button*>*-

*<*button (click)="cambiarDesde(5)" class="btn btn-secondary"*>*Siguientes*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

Creemos la pagina del medico

*<*div class="row"*>*

*<*div class="col-lg-7"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*h4 class="card-title"*>*Medico: Nombre medico*</*h4*>*

*<*h6 class="card-subtitle"*>*Nombre del medico*</*h6*>*

*<*form ngNativeValidate #f="ngForm" (ngSubmit)="guardarMedico(f)" class="form p-t-20"*>*

*<*div class="form-group"*>*

*<*label for="exampleInputuname"*>*User Name*</*label*>*

*<*div class="input-group"*>*

*<*div class="input-group-addon"*><*i class="ti-user"*></*i*></*div*>*

*<*input name="nombre" type="text" required class="form-control" id="exampleInputuname" placeholder="Username"*>*

*</*div*>*

*</*div*>*

*<*div class="form-group"*>*

*<*label for="exampleInputEmail1"*>*Hospital*</*label*>*

*<*select class="form-control" name="" id="" required*>*

*<*option value=""*>*Seleccione hospital*</*option*>*

*<*option \*ngFor="let hospital of hospitales" [value]="hospital.\_id"*>*

{{hospital.nombre}}

*</*option*>*

*</*select*>*

*</*div*>*

*<*button type="submit" class="btn btn-block btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Guardar*</*button*>*

*</*form*>*

*</*div*>*

*</*div*>*

*</*div*>*

*<*div class="col-lg-5"*>*

*<*div class="card"*>*

*<*div class="card-body"*>*

*<*div style="text-align: center"*>*

*<*h4 class="card-title"*>*Fotografia del Medico*</*h4*>*

*<*h6 class="card-subtitle"*>*nombre del medico*</*h6*>*

*<*img style="height: 240px;" alt=""*>*

*</*div*>*

*<*br*>*

*<*input name="achivo" (change)="seleccionImagen($event.target.files[0])" type="file"*>*

*<*br*>*

*<*br*>*

*<*button type="button" (click)="cambiarImagen()" [disabled]="!imagenSubir" class="btn btn-block btn-success waves-effect waves-light m-r-10"*><*i class="fa fa-save"*></*i*>* Actualizar foto*</*button*>*

*</*div*>*

*</*div*>*

*</*div*>*

*</*div*>*

El componente del medico

import { Component, OnInit } from '@angular/core';

import { NgForm } from '@angular/forms';

import { HospitalModel } from '../../models/hospitalModel';

import { MedicoService, HospitalService } from '../../services/service.index';

@Component({

selector: 'app-medico',

templateUrl: './medico.component.html',

styles: []

})

export class MedicoComponent implements OnInit {

hospitales: HospitalModel[]=[];

constructor(

public \_medicoService : MedicoService,

public \_hospitalServices : HospitalService

) { }

ngOnInit( )

{

this.\_hospitalServices.cargarHospitales()

.subscribe(hospitales=>this.hospitales = hospitales)

}

guardarMedico(f:NgForm){

console.log(f.valid);

console.log(f.value);

}

}