**Ahora iniciaremos nuestro front con Angular**

Instalamos angular

**npm install -g @angular/cli**

Creamos un nuevo proyecto

**ng new frontEnd**

Entramos a la carpeta que nos creó y lanzamos el proyecto

**ng serve**

Maquetamos algo simple con HTML5 y CSS3, que tenga secciones 1. Aboout, 2.Projects, 3. New Projects, 4. Contact

OPCIÓN 1 OPCIÓN 2 OPCIÓN 3 OPCIÓN 4 OPCIÓN 5

CABECERA

CUERPO DE LA PÁGINA

Luego conectaremos con el backend API-REST-CRUD, con las 5 opciones, crear, listar, buscar 1, actualizar y borrar proyectos del portafolios.

Primero creamos los 5 componentes que usaremos

1. About
2. Projects
3. Create
4. Contact
5. Error

Lo hacemos en la terminal, dentro de la carpeta del proeycto “frontEnd”

**ng g component components/about**

**ng g component components/projects**

**ng g component components/create**

**ng g component components/contact**

**ng g component components/error**

Cada uno nos crea un componente nuevo dentro de la carpeta componentes

Luego hacemos nuestro archivo de rutas dentro de App, lo llamaremos app.routing.ts

1. **Importamos los módulos del router de Angular**

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

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

1. **Importamos los componentes que recien creamos**

import { AboutComponent } from './components/about/about.component';

import { ProjectsComponent } from './components/projects/projects.component';

import { CreateComponent } from './components/create/create.component';

import { ContactComponent } from './components/contact/contact.component';

import { ErrorComponent } from './components/error/error.component';

1. **Definimos las rutas con un arreglo de objetos de tipo Routes, las ponemos en español porque son objetos visibles al usuario (ruta home o por defecto, about, projects, create, contact, y la ruta 404 que se hace con \*\* para una ruta incorecta o cuado no cargue ningún componente).**

const appRoutes: Routes = [

{path: '', component: AboutComponent},

{path: 'sobre-mi', component: AboutComponent},

{path: 'proyectos', component: ProjectsComponent},

{path: 'crear-proyecto', component: CreateComponent},

{path: 'contacto', component: ContactComponent},

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

];

1. **Exportamos el servicio de rutas, que es un array de tipo any**

export const appRoutingProviders: any[]=[];

1. **Exportamos el routing que es el modulo ModuleWithProviders, que llamara al modulo RouterModule a su método forRoot(appRoutes) que cargará nuestra configuración de rutas en el servidor de rutas de angular (appRoutes) y ponerla funcional.**

export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);

1. **Importamos el archivo de rutas en el app.module.ts**

import { routing, appRoutingProviders} from './app.routing';

1. **Cargamos el routing en el imports, porque es un móulo**

imports: [

BrowserModule,

routing

],

1. **Y cargamos el appRoutingProviders en los providers, porque es un servicio**

providers: [

appRoutingProviders

],

AUN NO PODEMOS ACCESAR LAS RUTAS, PORQUE NO ESTAMOS DISPARÁNDOLAS CON EL ROUTEROUTLET, QUE ES EL QUE NOS PERMITE ACCEAR A LOS COMPONENTES DENTRO DE LAS PÁGINAS EN FUNCIÓN DE LA URL QUE ESTEMOS ACCESANDO

1. **Remplazamos el contenido del app.component.html y utilizamos el routerOutlet , y nos muestra el sitio html por default**

<!-- 9 -->

<h1>Proyecto Portafolio - FrontEnd con Angular </h1>

<ul>

<li><a [routerLink]="['/sobre-mi']">Sobre mi</a></li>

<li><a [routerLink]="['/proyectos']">Proyectos</a></li>

<li><a [routerLink]="['/crear-proyecto']">Proyecto Nuevo</a></li>

<li><a [routerLink]="['/contacto']">Contacto</a></li>

</ul>

<router-outlet></router-outlet>

1. **Creamos un estilo en CSS, dentro de la carpeta assets y le ponemos de nombre styles.css**

**El cual para que funcione tenemos que declararlo en el archivo angular.json y dentro de la llave styles**

**lo incluimos y maquetamos al gusto.**

"styles": [ /\* 10 \*/

"src/assets/css/styles.css"

],

**🡨 CREO QUE NOS QUEDAMOS AQUÍ 🡪**

1. **Yo importé unas fuentes en el CSS que me gustan para usarlas posteriormente, los puse en la carpeta assets/fonts**

/\* 11 \*/

@font-face{

font-family: "titilium";

src: url("../fonts/titilium.ttf");

}

@font-face{

font-family: "StarWars";

src: url("../fonts/SW.ttf");

}

@font-face{

font-family: "BackToTheFuture";

src: url("../fonts/BTTF.ttf");

}

@font-face{

font-family: "StrangerThings";

src: url("../fonts/ST.ttf");

}

@font-face{

font-family: "finger";

src: url("../fonts/finger.ttf");

}

@font-face{

font-family: "yanone";

src: url("../fonts/yanone.ttf");

}

1. **Aplica para todo (quitamos margen y paddigin**

\*{

margin: 0px;

padding: 0px;

}

1. **Para el body**

/\* ---cuerpo---

fondo de un patrón

fuente personal

tamaño 10

fondo fijo

\*/

body

{

background: url("../img/pattern.png");

font-family: "StrangerThings", TimesNewsRoman, Arial;

font-size: 10px;

background-attachment: fixed;

}

1. **Maquetamos la cabecera**

**/\*Cabecera\*/**

**/\* limpiar el paso entre elementos \*/**

.clearfix{

float:none;

clear:both;

}

**/\* ---cabecera---**

**elementos en bloque**

**a todo el ancho**

**90 de alto**

**un fondo medio azul**

**posición fija arriba siempre aunque nos desplacemos**

**mayúsculas**

**\*/**

header{

display: block;

width: 100%;

height: 90px;

background: rgba(53, 103, 164, 0.8);

position: fixed;

text-transform: uppercase;

}

**/\* ----Lista----**

**elementos en bloque**

**ancho 40%**

**alto heredado del padre**

**margen 0 auto**

**margen superior -3**

**\*/**

header ul{

display: block;

width: 40%;

height: inherit;

margin: 0px auto;

margin-top: -3px;

}

**/\* ----Elementos de la Lista (Barra de menú)----**

**elementos en bloque en línea**

**margen 0 auto**

**texto al centro**

**alto de la fila al alto de la cabecera**

**\*/**

header ul li{

display: inline-block;

margin: 0px auto;

text-align: center;

line-height: 88px;

}

**/\* ----Enlaces----**

**despligue en bloque**

**despegados 20 cada uno a la izq y derecha**

**altura heredada**

**sin decoración**

**color blanco**

**transición al pasar entre ellos**

**\*/**

header a{

display: block;

padding-right: 20px;

padding-left: 20px;

height: inherit;

text-decoration: none;

color:white;

transition: all 300ms;

}

**/\* ---al pasar sobre los enlaces ---**

**cambiamos el fondo**

**---al dar clic sobre ellos---**

**los dejamos marcados**

**\*/**

header a:hover,

.actived{

background: rgba(0,0,0,0.8);

}

**/\*Logotipo**

**otra fuente**

**otro tamaño**

**otra transición**

**\*/**

#logo{

font-family: "StarWars";

font-size: 20px;

transition: color 500ms, transform 500ms;

text-transform: none;

}

**/\* ---al pasar sobre el logo ---**

**cambiamos el fondo pero a transparente,**

**para que no se vea la transición con el color**

**y le cambiamos el color y el tamaño para que se haga**

**como un zoom**

**\*/**

#logo a:hover{

background: transparent;

color: black;

transform:scale(1.50,1.50);

text-transform: none;

}

1. **Contenedor principal – espacio donde se mostrarán los datos**

**/\*Contenedor principal - donde se despliga el texto**

**altura minima**

**altura auto**

**ancho 75% respecto a la página**

**margen**

**ponemos una sombra**

**bordes**

**fondo**

**y padding superior para que este debajo del menu**

**\*/**

#content{

min-height: 600px;

height: auto;

width: 75%;

margin: 0px auto;

box-shadow: 0px 0px 40px black;

border-left:1px solid #b7b7b7;

border-right:1px solid #b7b7b7;

background: rgba(255, 255, 255, 0.7);

padding-top: 120px;

}

1. **Pie de página**

/\*Pie de pagina

Ancho

margen

paddind superior debajo del body

color con opacidad

\*/

footer{

width: 75%;

margin: 0px auto;

padding-top: 15px;

color: rgba(0, 0, 0, 0.5);

}

*Lo siguiente es modificar cada uno de los componentes que hemos creado para mostrar el contenido deseado y hacer sus hojas de estilo, empezamos con el ABOUT, para lo cual abrimos el* ***about.component.html*** *y* ***about.component.ts***

1. **Acerca de…**

**En about.component.ts Dentro de la clase declaramos 3 variables y las usamos en el constructor (solo negritas)**

export class AboutComponent implements OnInit {

**public title: string;**

**public subtitle: string;**

**public email: string;**

constructor() {

**this.title = "José Alfredo Aguirre Puente";**

**this.subtitle = "Profesor";**

**this.email = "ingefreddy@hotmail.com";**

}

ngOnInit() {

}

}

**En about.component.html**

<div id="title">

<!-- LLamámos al título, subtitulo y correo-->

<h1>{{title}}</h1>

<h2>{{subtitle}}</h2>

<p>{{email}}</p>

</div>

<article class="about">

<h2>Sobre mí</h2>

<p>

CONTENIDO DE LA PÁGINA

</p>

</article>

1. **Seguimos en la hoja de estilo con el acerca de**

**/\*About**

**--- Caja de título ---**

**\*/**

#title{

width: 70%;

margin: 0px auto;

text-align: center;

margin-top: 20px;

margin-bottom: 70px;

}

**/\* nivel h1\*/**

#title h1{

font-family: "finger";

font-size: 60px;

border-bottom: 1px solid #ccc;

padding-bottom: 5px;

}

**/\* nivel h2\*/**

#title h2{

font-size: 25px;

color: gray;

margin-top: 5px;

margin-bottom: 5px;

}

**/\* Contenido de la página\*/**

article.about{

width: 70%;

margin: 0px auto;

margin-bottom: 20px;

font-size: 23px;

}

**/\* nivel h2 de la clase about\*/**

.about h2{

border-bottom: 1px solid #ccc;

padding-bottom: 5px;

margin-bottom: 5px;

}

**/\* párrafo de clase about\*/**

.about p{

padding-top: 20px;

padding-bottom: 30px;

text-align: justify;

}

1. **Aquí crearemos el formulario de crear nuevos proyectos, primero la parte lógica del funcionamiento y luego el formulario**

**Nos aseguramos que este arrancado el Mongo (mongod)**

**Arrancamos el backend (NPM start)**

**Dentro de APP creamos una carpeta para el modelo, que representa a una entidad o documento de nuestra DB**

**Dentro creamos un archivo que se llamará project.ts**

**Definimos nuestro modelo**

**/\* Definimos nuestra clase Project \*/**

export class Project{

**//usamos el constructor y dentro**

**//definimos cada propiedad de nuestro modelo**

**//como parámetro**

constructor(

public \_id: string,

public name: string,

public description: string,

public category: string,

public year: number,

public langs: string,

public image: string

){}

}

**Dentro de APP creamos una carpeta para los servicios**

**Creamos un archivo de configuración global para los servicios, lo llamamos global.ts**

// Exportamos la URL de la API - nuestro JSON

// para reutilizarla

export var Global = {

url: 'http://localhost:3700/api/'

};

**Creamos el servicio, el archivo se llamará project.service.ts**

//creamos el servicio con todos los módulos necesario

//creamos un servicio inyectable

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

//importamos para hacer peticiones AJAX y HTTP

import { HttpClient, HttpHeaders } from '@angular/common/http';

//importamos la librería ReactiveX observable que ya la incluye angular

//es una API para programación asincrona con observable stream

// es decir- controlaremos el flujo

import { Observable } from 'rxjs/observable';

//importamos nuestro Projecto que acabamos de hacer

import { Project } from '../models/project';

//importamos nuestro archivo de configuración global con la URL

import { Global } from './global';

//Con el decorador inyectable, exportamos la clase - servicio

//con la url de donde alojaremos la appi

@Injectable()

export class ProjectService{

public url:string;

//en el constructor le damos un valor a la url con la configuración Global

constructor( private \_http: HttpClient ){

this.url = Global.url;

}

/\* Creamos un método de prueba\*/

testService(){

return 'Probando el servicio de Angular';

}

}

**El el app.module.ts importamos los servicios, modelos y formularios, para httpclients (ajax) y para los bindings**

import {HttpClientModule} from '@angular/common/Http';

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

**Ahí mismo en el app.module en la parte de los imports agregamos los 2 recién creados**

HttpClientModule,

FormsModule

**--------- Si manda error RXJS ejecutar la compatibilidad con la nueva versión --------**

Npm i –save compat

**Abrimos create-components.ts**

**Realizamos las importaciones y configuraciones necesarias en create-components.ts**

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

**// Importamos el modelo y el servicio que creamos**

import { Project } from '../../models/project';

import { ProjectService } from '../../services/project.service';

import { Global } from '../../services/global';

@Component({

selector: 'app-create',

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

styleUrls: ['./create.component.css'],

**// En este decorador cargamos el objeto ProjectService**

providers: [ProjectService]

})

export class CreateComponent implements OnInit {

**//Creamos propiedades para el título**

public title: string;

**//Otro para el objeto que modificará el formulario (de tipo project)**

public project: Project;

constructor(

**//creamos las propiedades del servicio**

private \_projectService: ProjectService

){

**//Pasamos el título**

this.title = "Crear proyecto";

**//Les damos valor a esas propiedades y al modelo le pasamos datos en blanco**

**//que se sustituirán despues por los datos que se almacenarán en el documento de la DB**

**//\_id, name, description, category,langs,year, image**

this.project = new Project('','','','',1975,'','');

}

ngOnInit() {

}

}

**Creamos el formulario para crear nuevos documentos en create-components.html**

**//ponemos un div con un h2 para el título**

<div class="container" \*ngIf="project">

<h2>{{title}}</h2>

</div>

**//Dentro de ese DIV creamos un formulario**

**<!-- Aqui le decimos que es un formulario de angular que nombramos #projectForm -->**

**<form #projectForm="ngForm" (ngSubmit)="onSubmit(projectForm)">**

**<!—en el evento ngSubmit definimos lo que enviaremos y necesitamos crear ese método SUBMIT en el archivo create-components.ts que reciba por parámetro el formulario (form) -->**

onSubmit(form){

console.log(this.project);

}

**//De regreso al formulario HTML creamos los campos que se enviarán**

**//Nombre, Descripción, Categoría , Fecha, Idiomas e imagen**

**// #name=”ngModel” es el nombre del formulario de angular**

**//[(ngModel)]=”Project.name” es el Binding que modificará la propiedad de ese objeto**

**//Validamos que lleve datos**

<p>

<label for="name">Nombre</label>

<input type="text" name="name" #name="ngModel" [(ngModel)]="project.name" required />

<span class="form\_error" \*ngIf="name.touched && !name.valid">

El nombre es obligatorio

</span>

</p>

<p>

<label for="description">Descripción</label>

<textarea name="description" #description="ngModel" [(ngModel)]="project.description" required></textarea>

<span class="form\_error" \*ngIf="description.touched && !description.valid">

La descripción es obligatoria

</span>

</p>

<p>

<label for="category">Categoría</label>

<input type="text" name="category" #category="ngModel" [(ngModel)]="project.category" required />

<span class="form\_error" \*ngIf="category.touched && !category.valid">

La categoría es obligatoria

</span>

</p>

<p>

<label for="year">Año de lanzamiento</label>

<input type="number" name="year" #year="ngModel" [(ngModel)]="project.year" />

</p>

<p>

<label for="langs">Lenguajes utilizados</label>

<input type="text" name="langs" #langs="ngModel" [(ngModel)]="project.langs" required />

<span class="form\_error" \*ngIf="langs.touched && !langs.valid">

Los lenguajes son obligatorios

</span>

</p>

<p>

<label for="image">Imagen del proyecto</label>

<input type="file" name="image" placeholder="Subir imagen" />

</p>

<input type="submit" value="Enviar" [disabled]="!projectForm.form.valid" />

**</form>**

**CSS – Formulario nuevo proyecto**

/\*Sección con formulario\*/

.container{

padding: 50px;

padding-top: 10px;

}

.container .image{

float: left;

width: 47%;

}

.container .image img{

width: 100%;

}

.container .data{

float: left;

width: 48%;

margin-left:30px;

}

.container h2{

display: block;

border-bottom: 1px solid #ccc;

padding-bottom: 10px;

margin-bottom: 15px;

}

form{

width: 80%;

}

form label{

display: block;

width: 100%;

margin-top: 10px;

margin-bottom: 5px;

}

form input[type="text"],

form input[type="number"],

form input[type="email"],

form textarea{

width: 40%;

padding: 5px;

}

form button,

form input[type="submit"],

.button-edit,

.button-delete{

display: block;

font-size: 17px;

padding: 10px;

margin-top: 15px;

color:white;

background: rgba(53, 103, 164, 1);

border: 1px solid rgba(13, 67, 133, 1);

cursor: pointer;

}

.button-edit,

.button-delete{

width: 30%;

text-align: center;

float: left;

margin-right: 10px;

text-decoration: none;

}

.button-edit{

background: orange;

border: none;

}

.button-delete{

background: red;

border: none;

}

form button:hover,

form input[type="submit"]:hover{

background: rgba(53, 103, 164, 0.8);

border: 1px solid rgba(13, 67, 133, 1);

}

form input[disabled]{

opacity: 0.4;

cursor: not-allowed;

}

.form\_error{

font-size: 13px;

padding: 3px;

background: red;

color:white;

margin-left: 5px;

}

.message{

width: 50%;

padding: 5px;

border: 1px solid #eee;

}

.success{

background: green;

color: white;

}

.message a{

color: white;

}

.failed{

background: red;

color: white;

}

/\*Pagina de proyectos\*/

.project{

list-style: none;

display: block;

float: left;

width: 28%;

text-align: center;

margin-left: 20px;

margin-right: 20px;

margin-bottom: 30px;

}

.project .image{

width: 100%;

height: 150px;

overflow: hidden;

}

.project img{

width: 100%;

}

.project a{

text-decoration: none;

color: black;

}

.confirm{

display: block;

font-weight: bold;

margin-top: 20px;

}

1. **En Project.Service.ts creamos el método para almacenar los documentos, al cual le pasamos por parámetro Project, que sería nuestro proyecto, le pasamos los datos del proyecto en formato JSONString y devuelve un observable de cualquier tipo**

saveProject(project: Project): Observable<any>{

let params = JSON.stringify(project);

**Establecemos las cabeceras del cómo enviaremos los datos, lo hacemos en su propiedad** set **y le decimos que se irá como un json**

let headers = new HttpHeaders().set('Content-Type', 'application/json');

**Hacemos la petición a la URL por el método POST para agregar uno nuevo, concatenándole LA RUTA, LOS PARÁMETROS y LAS CABECERAS**

return this.\_http.post(this.url+ 'save ', params, {headers: headers});

}

**21. Ahora en la parte del create.Components.ts, donde accedemos al método submit (ligado al boton de nuestro create.Component.html) hay que implementarlo y recuperar lo que devuelva la API (método de callback suscribe) )**

this.\_projectService.saveProject(this.project). subscribe (

response => {

console.log(response);

},

error =>

{

console.log(<any>error);

}

);

**Con eso debe enviar al backend, hagamos un mensaje de éxito en la grabación en create.Components.ts**

**Creamos primero el objeto que traera el mensaje y luego en el response el IF que validará si hubo éxito o no**

public status: string;

**//Validamos si se grabo con éxito en el evento onSubmit(form), limpiamos el formulario y lo usamos en el create.Component.html**

if (response.project){

this.status = "success";

form.reset();

}

else

{

this.status="failed";

}

},

**22. // create.Component.html debajo del title, ponemos el mensaje con un dentro de un div**

<div class="message success" \*ngIf="status=='success'">

Documento almacenado con éxito

</div>

<div class="message failed" \*ngIf="status=='failed'">

Documento No pudo ser almacenado

</div>

**23. // SUBIR IMAGEN – CREAMOS OTRO SERVICIO PARA ESTO EN LA CARPETA DE services YO LE LLAMÉ upload.services.ts**

**//Importamos**

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

import {Global} from './global';

**//Definimos el servicio y lo exportamos**

@Injectable()

export class UploadService{

public url: string;

**// pasamos la URL**

constructor(){

this.url = Global.url;

}

**//Método para hacer una petición ajax para subir el archivo**

makeFileRequest(url: string, params: Array<string>, files: Array<File>, name: string){

**//Hacemos una promesa con su función callback con su resolve y su reject**

**//Por su se logra o no**

return new Promise(function(resolve, reject){

**//Definimos la petición ajax para subir el archivo simulando un formulario.**

var formData:any = new FormData();

**//Hacemos un objeto de petición asíncrona**

var xhr = new XMLHttpRequest();

**//Hacemos un for para recorrer los posibles archivos que llegan**

**// y mientras no se acaben los archivos**

**// los pegamos al formulario con el "name" que llega por parámetro**

**// añadimos el archivo y recuperamos su nombre**

for (var i = 0; i < files.length; i++) {

formData.append(name, files[i], files[i].name);

}

**//ahora la función ajax, cuando halla un cambio en su estado**

**//el primer if valida si se completo**

**//el segundo if valida si tuvo exito y en ese caso**

**//parseamos la respuesta del servicio y si no mandamos el reject**

xhr.onreadystatechange = function(){

if(xhr.readyState == 4){

if (xhr.status == 200) {

resolve(JSON.parse(xhr.response));

}

else{

reject(xhr.response);

}

}

}

**//Si todo tuvo exito hacemos la petición (POST)**

**//y enviamos el formulario con los archivos que hayan llegado**

xhr.open('POST',url, true);

xhr.send(formData);

});

}

**23. // importamos este servicio en create.Component.ts**

import { UploadService } from '../../services/upload.service';

**//cargamos el servicio en los providers**

providers: [ProjectService, UploadService],

**//creamos las propiedades del servicio en el constructor**

private \_projectService: ProjectService,

private \_uploadService: UploadService ) {

**Ahora por último en nuestro formulario create.Components.html modificamos para que suba la imagen con el evento onChange y creamos un método (yo lo llamé fileChangeEvent) y le pasamos por parámetro en una variable (que yo llamé $event) todo lo que vayamos a enviar por ese input, como el nombre, archivo escogido … y creamos ese método en el create.Component.ts**

<input type="file" name="image" placeholder="Subir imagen" **(change)="fileChangeEvent($event)"**/>

**// Creamos Objeto con el archivo a subir y luego el método**

public filesToUpload: Array<File>;

fileChangeEvent(fileInput: any){

**//Esto es para ver lo que manda al seleccionar el archivo**

//console.log(fileInput);

**// una vez que vemos lo que ocupamos (nombre del archivo)**

**// lo filtramos**

this.filesToUpload = <Array<File>>fileInput.target.files;

}

**// ya que tenemos el nombre, cuando teníamos el SUCCESS,**

**//es decir , cuando guardabamos los datos con éxito subímos la imagen**

**// usamos el servicio para subir archivos con el método que creamos para subir por ajax y le pasamos los parámetros que definimos antes (1: url con la ruta del backend y el id de la imagen, 2: opcional – lo dejamos vacio, 3: el filesToUpload, 4: y el nombre del campo que lo recibirá en el backend ue le habiamos llamado image) this.uploadService.makeFileRequest() Como es una promesa tiene su método then() y su callback, y metemos los mensajes de éxito dentro del then, es decir al lograrse la promesa de subir el archivo.**

if (response.project){

/\* this.status = "success";

form.reset(); \*/

**//filtramos lo que pcupamos de todo lo que arroja el input**

this.\_uploadService.makeFileRequest(Global.url+"upload/"+response.project.\_id, [], this.filesToUpload, 'image')

.then((result:any) => {

console.log(result);

this.status = "success";

form.reset();

});

}