1) INSTALAMOS ANGULAR EN EL PROYECTO

a. NPM INSTALL -G @ANGULAR/CLI → [PARA INSTALAR EN LA CARPETA DEL PROGRAMA]
b. CONSULTAR NPM -V
c. NG NEW NOMBRE_PROYECTO (DIFERENTE A NOMBRE CARPETA)
d. NG SERVE [PARA LEVANTAR EL SERVIDOR] → LEVANTA EL SERVIDOR DE FRONT

Do you want to enforce stricter type checking and stricter bundle budgets in the workspace? This setting helps improve maintainability and catch bugs ahead of time. For more information, see https://angular.io/stric
¿Desea hacer cumplir una verificación de tipos más estricta y presupuestos de paquetes más estrictos en el espacio de trabajo? Esta configuración ayuda a mejorar la capacidad de mantenimiento y detectar errores con anticipación.(Y/**N**)

2) CREAMOS LOS COMPONENTES DENTRO DE LA CARPETA (DENTRO DEL TERMINAL)

```
Ng g component component/about
Ng g component component/projects
Ng g component component/create
Ng g component component/contact
Ng g component component/error
```

3) CREAMOS APP.ROUTING.TS

a. IMPORTAMOS LOS MÓDULOS DE LA RUTA

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

b. IMPORTAMOS LOS COMPONENTES DE LA RUTA (GENERADOS POR LOS COMPONENTES

```
/* IMPORTAMOS LOS COMPONENTES CREADOS PARA CADA PAGINA  */
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';
import { DetailComponent } from './components/detail/detail.component';
import { EditComponent } from './components/edit/edit.component';
```

c. DEFINIR RUTA

```
/* GENERAMOS LAS RUTAS DE LOS COMPONENTES */


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: 'proyecto/:id', component: DetailComponent},
    {path: 'editar-proyecto/:id', component: EditComponent},
    {path: '**', component: ErrorComponent}
];
```

d. EXPORTAR LA CONFIGURACIÓN DE LAS RUTAS:

```
/* EXPORTAMOS LOS MODULOS  */
export const appRoutingProviders: any[] =[];
export const Routing: ModuleWithProviders<any> = RouterModule.forRoot(AppRoutes);
```

4) CARGAMOS EL ROUTING EN APP.MODULE.TS

a. CARGAMOS EL ROUTING GENERADO EN APP.ROUTING.TS

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

b. CARGAMOS LOS COMPONENTES DENTRO DE @NGMODULE [ROUTING (MODULO) Y APPROUTINGPROVIDERS (SERVICIO)]

APP.MODULE.TS

```
@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ProjectsComponent,
    CreateComponent,
    ContactComponent,
    ErrorComponent
  ],
  imports: [
    BrowserModule,
    Routing, // → COMO MODULO
    HttpClientModule,
    FormsModule
  ],
  providers: [
    appRoutingProviders // → COMO SERVICIO
  ],
  bootstrap: [AppComponent]
})
```

5) CARGAR PAGINA EN FUNCION DE LA RUTA Y GENERAR MENU

a. DENTRO DE APP.COMPONENT.HTML

APP.COMPONENT.HTML

```
<header>
  <ul id="header_list">
    <li> <a [routerLink]="['/sobre-mi']" [routerLinkActive]="['actived']">Sobre mi</a></li>
    <li><a [routerLink]="['/proyectos']" [routerLinkActive]="['actived']">Proyectos</a></li>
    <li id="logo"><a href="#">VR</a></li>
    <li><a [routerLink]="['/crear-proyecto']" [routerLinkActive]="['actived']">Crear
proyecto</a></li>
    <li><a [routerLink]="['/contacto']" [routerLinkActive]="['actived']">Contacto</a></li>
  </ul>
</header>

<section id="content">
  <router-outlet></router-outlet>

</section>

<footer>
  Master en JavaScript de victorroblesweb.es &copy;
</footer>
[PROPIEDAD] → ES PARA UN CAMBIO EN EL CODIGO
(EVENTO) → ES UN EVENTO, COMO ONCLICK SOLO QUE VA SIN EL "ON"
[(DIRECTIVA )] → ES PROPERTY BINDING, SE UTILIZA HACER  CAMBIOS EN TIEMPO REAL EN LA BASE DE DATOS

[routerLink]="['/url']" → GENERA PROPIEDAD QUE ANADE AL LINK LA TERMINACION,
```

[routerLinkActive] ="['actived']" → AL ESTAR ACTIVADA LA CLASE ANADE EL ESTILO CSS .ACTIVED

6) MAQUETACION DE PAGINA SOBRE MI (ABOUT)

a. CREAMOS STYE.CSS [ASSETS → CSS → STYLE.CSS]

b. VINCULAMOS LA CUENTA DENTRO DE ANGULAR.JSON

EN ANGULAR.JSON

```
"styles": [
   "src/assets/css/styles.css",
   "src/assets/bxslider/dist/jquery.bxslider.min.css"
],
```

→ VAMOS A CREAR UN STYLE.CSS QUE UNIFICARA TODOS LOS CSS, PARA ESO CREAMOS UN STYLES.CSS DENTRO DE LA CARPETA SRC/ASSETS/CSS Y LO IMPORTAMOS DENTRO DE ANGULAR.JSON

c. DEFINIMOS LA MAQUETACIÓN EN STYLE.CSS

STYLE.CSS (EN ASSETS/CSS)

```
/* --- IMPORTAR FUENTES ---*/
@font-face{
    font-family: "yanone";
    src: url("fonts/yanone.ttf"); → FONDO PARA LETRA RECOMENDADO
}
@font-face{
    font-family: "finger";
    src: url("fonts/finger.ttf");
}

/* --- MAQUETACION GENERAL ---*/
*{
    margin: 0px;
    padding: 0px;
}
body{
    font-family: "yanone",Arial, Helvetica, sans-serif;
    font-size: 20px;
    background: url("../img/pattern.png");
    background-attachment: fixed; /*GENERA QUE NO SE MUEVA LA IMAGEN*/
}
/* --- GENERAMOS CLASE PARA LIMPIAR ---*/

.clearfix{
    float:none;
    clear:both;
}

/*--- BARRA DE NAVEGACION / CABECERA ---*/

header{
    display: block;
    width: 100%;
    height: 90px;
    background: rgba(53, 103, 164, 0.8);
    position: fixed;
    text-transform: uppercase;
}
```

```css
header ul{
    display: block;
    width: 40%;
    height: inherit;
    margin: 0px auto;
    margin-top: -3px;
}
```

→ **height: 100% coincidirá con la altura del padre del elemento, independientemente del valor de altura del padre.**

→ **height: inherit como su nombre lo indica, heredará el valor de su padre. Si el valor del padre es height: 50%, entonces el hijo también tendrá el 50% de la altura de su padre. Si el tamaño del padre se define en valores absolutos (por ejemplo height: 50px), entonces height: inherity height: 100%tendrá el mismo comportamiento para el niño.**

```css
header ul li{
    display: inline-block;
    margin: 0px auto;
    text-align: center;
    line-height: 88px;
}

header a{
    display: block;
    padding-right: 20px;
    padding-left: 20px;
    height: inherit;
    text-decoration: none;
    color:white;
    transition: all 300ms;
}

header a:hover,
.actived{
    background: rgba(0,0,0,0.8);
}

/* --- MODIFICAR LOGO --- */

#logo{
    font-family: "finger";
    font-size: 30px;
}

#logo a{
    transition: color 500ms, transform 500ms;
        /* TRANSISION SOLO AFECTA A COLOR Y TRANSFORM DEL OVER*/
}

#logo a:hover{
    background: transparent;
    color: black;
    transform:scale(1.50,1.50);
}
```

```
→ transform: scale( 1.50, 1.50) genera el efecto de hacer las letras/números más grandes

/* --- CONTENEDOR PRINCIPAL --- */

#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;
}

/* --- PIE DE PAGINA ---*/
footer{
    width: 75%;
    margin: 0px auto;
    padding-top: 15px;
    color: rgba(0, 0, 0, 0.5);
}

/* *-*-*-*-*- COMPONENT/ABOUT ME -*-*-*-*-* */
/*Sobre mi*/
#title{
    width: 70%;
    margin: 0px auto;
    text-align: center;
    margin-top: 20px;
    margin-bottom: 70px;
}

#title h1{
    font-family: "finger";
    font-size: 60px;
    border-bottom: 1px solid #ccc;
    padding-bottom: 5px;
}

#title h2{
    font-size: 25px;
    color: gray;
    margin-top: 5px;
    margin-bottom: 5px;
}

article.about{
    width: 70%;
    margin: 0px auto;
    margin-bottom: 20px;
    font-size: 23px;
}
```

```css
.about h2{
    border-bottom: 1px solid #ccc;
    padding-bottom: 5px;
    margin-bottom: 5px;
}

.about p{
    padding-top: 20px;
    padding-bottom: 30px;
    text-align: justify;
}

/* --- CREAR PROYECTO --- */

.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;
}
```

```css
.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;
}
```

7) **LEVANTAR API PARA "CREAR PROYECTO"**

   a. **ABRIEMOS EL MONGOD**

b. LEVANTAMOS LA APP DE ANGULAR MEDIANTE NG SERVE

c. LEVANTAMOS EL API (CREADO ANTERIORMENTE EN 02-PROYECTO-NODEJS) O GENERAMOS UNO NUEVO

d. CREAMOS LA CARPETA APP → MODELS → PROJECT.TS, PARA DEFINIR UN MODELO

MODELS – PROJECT.TS

```
export class Project{
    constructor(
        public _id: string,
        public name: string,
        public description: string,
        public category: string,
        public year: number,
        public lang: string,
        public image: string
    ){}
}
```

CON ESTO GENERAMOS UN MODELO IGUAL AL QUE TENEMOS DENTRO DE LA BASE DE DATOS DE MONGO, LA CUAL LA PODREMOS UTILIZAR PARA ESTA BASE DE DATOS.

e. CREAMOS LA CARPETA APP – SERVICE – GLOBAL.TS

SERVICE – GLOBAL.TS

```
export var Global = {
    url: 'localhost:3700/api/'
}
```

LA UTILIZAREMOS PARA GENERA LA VARIABLE GLOBAL, LA CUAL ES LA URL DEL API

localhost:3700/api/

f. CREAMOS CARPETA APP – SERVICE- PROJECT.SERVICE.TS

SERVICE – PROJECT.SERVICE.TS

```
import { Injectable } from '@angular/core'; //MODULO PARA IMPORTAR SERVICIOS
import { HttpClient, HttpHeaders } from '@angular/common/http';
//IMPORTA PETICIONES AJAX Y MODIFICAR CABECERAS
import { Observable } from 'rxjs';
//RECOGE INFORMACION DE LA APIS MEDIANTE LIBRERÍA EN ANGULAR
SI PRESENTAMOS ERROR EN {OBSERVABLE}
DEBEMOS DE INSTALAR $NPM INSTALL –SAVE RXJS -COMPAT (DESDE BACKEND)

import { Project } from "../models/project"; // IMPORTAMOS EL MODELO
import { Global } from "./global";

@Injectable()
export class ProjectService {
    public url: string;

    constructor(
        private _http: HttpClient
    ) {
        this.url = Global.url;
    }
    testService() {
        return "Probando el servicio de angular";
    }
}
```

g. IMPORTAMOS A APP.MODULE.TS E IMPORTAMOS LOS MÓDULOS

APP.MODULE.TS

```
import { HttpClientModule } from "@angular/common/http";
//PARA PETICIONES AJAX CON HTTPCLIENTE
import {FormsModule } from "@angular/forms";
//PARA UTILIZAR EL FORMULARIO Y TWO-DATA-BINDING

@NgModule({
  imports: [
    BrowserModule,
    Routing,
    HttpClientModule,
    FormsModule
  ],
```

8) EN CREATE.COMPONENT:

a. EN CREATE.COMPONENT.TS

1.INVOCAMOS LOS COMPONENTES

```
import { Project } from "../../models/project";
// INVOCAMOS AL MODELO PROJECT
import { ProjectService } from "../../services/project.service";
// INVOCAMOS AL SERVICIO
```

2.CREAMOS LOS MÉTODOS DENTRO DEL COMPONENTE

```
  public project: Project; // CREAMOS UN MÉTODO A TRAVES DEL MODELO
  public title: string;
```

3.CREAMOS LAS PROPIEDADES DE SERVICIO DENTRO DEL CONSTRUCTOR Y DEFINIMOS VALOR A LOS MÉTODOS

```
constructor(
    private _projectService: ProjectService
//CREAMOS UN MÉTODO PRIVADO A TRAVES DEL SERVICIO
  ) {
    this.title = "Crear Proyecto";
    this.project = new Project("", "", "","", 2019, "","");
//DEBEN SER LOS MISMOS CONTENIDOS QUE EN EL MODELO
  }
```

b. CREAMOS EL FORMULARIO EN CREATE.COMPONENT.HTML Y VALIDAMOS FORMULARIO

```
<div class="container">
    <h2>{{title}}</h2>
    <form #projectForm="ngForm" (ngSubmit)="onSubmit(projectForm)">
        <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</label>
            <input type="number" name="year" #year="ngModel"
             [(ngModel)]="project.year" required />
            <span class="form_error" *ngIf="year.touched && !year.valid">
             El año es obligatorio</span>
        </p>
        <p>
            <label for="lang">Lenguajes utilizados</label>
            <input type="text" name="langs" #langs="ngModel"
             [(ngModel)]="project.lang" 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" required
              (change)="fileChangeEvent($event)"/>
 </p>
        <input type="submit" value="Enviar" [disabled]="!projectForm.form.valid" />
    </form>

</div>
```

*** VALORES DEL FORMULARIO ***

```
#projectForm     => Nombre Formulario en Angular
"ngForm"         => Informamos que es de tipo formulario angular
(ngSubmit)="onSubmit(projectForm)" => Evento ngSubmit llama al método onSubmit
                                      (desde create.component.ts) para que
                                      modifique projectForm
for="name"                   => Genera un valor al label
type="text/number"           => Genera de tipo texto o numero
name="name"                  => Nomenclatura para el input
#name="ngModel"              => Informamos que el input sera un ngModel
[(ngModel)]="project.name"   => Se utiliza el two way data binding para modificar a
                                la vez la base de dato con lo escrito
<input type="submit" value="Enviar"> => Se genera un valor de boton
```

*** VALIDACION DE FORMULARIO ***

```
required => Genera que el valor sea obligatorio para continuar
<span *ngIf="name.touched && !name.valid"></span> =>Genera que si se toca y no tiene
                                                    contenido se lance el span
[disabled]="!projectForm.form.valid"  => Genera contenido en disable sino hay contenido dentro del formulario
```

c. MAQUETAMOS EL FORMULARIO

```
/* --- CREAR PROYECTO --- */

.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,
```

```css
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;
}
```

9) GUARDAR EL FORMULARIO EN LA BASE DE DATOS

a. GENERAMOS EL MÉTODO PARA GUARDAR EL PROYECTO (SAVE PROJECT)

EN PROJECT.SERVICE.TS

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { Project } from '../models/project';
import { Global } from './global';

@Injectable()
export class ProjectService{
    public url:string;

    constructor(
        private _http: HttpClient
    ){
        this.url = Global.url;
    }

    testService(){
        return 'Probando el servicio de Angular';
    }

    saveProject(project: Project): Observable<any>{
        let params = JSON.stringify(project);
        let headers = new HttpHeaders().set('Content-Type','application/json');

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


}


                            *** METODO PARA GUARDAR PROYECTO ***

let params = JSON.stringify(project) => SE SOLICITA LOS DATOS DE LA PÁGINA
                                        WEB Y LOS CONVIERTE EN STRING JSON
```

```
                                PARA LA BASE DE DATOS
let headers = new HttpHeaders().set('Content-Type','application/json'); =>
                TENEMOS QUE ESTABLECER UNAS CABECERAS DONDE SE ENVIA
                LA INFORMACION SET () ES PARA REEMPLAZAR LOS DATOS.

return this._http.post(this.url+"save-project",params,{headers:headers}); =>
                EL METODO NOS DEVOLVERA UNA PETICION POR POST, EN LA
                CUAL VAMOS A DAR DE ALTA AL API DEL BACK END
                "02-PROJECTO-NODEJS", EN EL CUA ANADIREMOS SOBRE LA
                URL, LA VINCULAMOS AL PARAMETRO GUARDADO, MODIFICAMOS
                PARAMS, INCLUIMOS CABECERAS.
     "THIS.URL" =>ES EL URL TOMADO DEL API, EN ESTE CASO NOSOTROS
                  LO TENEMOS EXTRADIO DE GLOBAL
  "SAVE-PROYECT" =>DENTRO DEL BACKEND02-PROJECT-NODEJS/ROUTES/ PROJECT.TS
               TOMAMOS LA RUTA YA CREADA POR EL API
           router.post("/save-project",ProjectController.SaveProject)
      "PARAMS" => SON LOS PARAMETROS MODIFICADOS EN LA PAGINA WEB
{HEADERS:HEADERS} => ES LA CABECERA QUE SE VA A MODIFICAR ES UN JSON
```

b. UTILIZAMOS EL SERVICIO EN CREATE.COMPONENT.TS

CREATE.COMPONENT.TS

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

import { Project } from "../../models/project"; // INVOCAMOS AL MODELO PROJECT
import { ProjectService } from "../../services/project.service"; // INVOCAMOS AL SERVICIO

import { UploadService } from "../../services/upload.service";
import { Global } from "../../services/global";

@Component({
  selector: 'app-create',
  templateUrl: './create.component.html',
  styleUrls: ['./create.component.css'],
  providers: [ProjectService, UploadService] // PROJECTSERVICE ES UN SERVICIO SE INVOCA
MEDIANTE PROVIDERS
})
export class CreateComponent implements OnInit {

  public project: Project; // CREAMOS UN MÉTODO A TRAVES DEL MODELO
  public title: string;
  public status: string;
  public filesToUpload: Array<File>;

  constructor(
    private _projectService: ProjectService, //CREAMOS UN MÉTODO PRIVADO A TRAVES DEL
SERVICIO
    private _uploadService: UploadService
  ) {
    this.title = "Crear Proyecto";
    this.project = new Project("", "", "", "", 2019, "", ""); //DEBEN SER LOS MISMOS
CONTENIDOS QUE EN EL MODELO
  }
```

```
  ngOnInit() {
  }
  /*CREAMOS EL EVENTO ONSUBMIT*/
  onSubmit(form: any) {

    //GUARDAR DATOS
    this._projectService.saveProject(this.project).subscribe(
      response => {
        if (response) {
          console.log(response);
          //SUBIR IMAGEN
          this._uploadService.makeFileRequest(Global.url + "upload-image/" +
this.project._id, [], this.filesToUpload, 'image').then((result: any) => {
            this.status = "success";
            console.log(result);
            form.reset(); //SE VACIA EL FORMULARIO
          });

        } else {
          this.status = "failed";
        }
      }, error => {
        console.log(<any>error);
      }
    );
  }

  fileChangeEvent(fileInput: any) {
    console.log(fileInput);
    this.filesToUpload = <Array<File>>fileInput.target.files;
  }

}
```

*** METODO PARA STRING ***

import { UploadService } from "../../services/upload.service"; => Importamos el servicio para cargar archivos, lo veremos después
@Component({providers: [UploadService]}) => Incluimos el servicio de cargar archivos
public status: string; => Creamos el estado para validación de formulario
public fileToUpload: Array<File> => Creamos el parámetro para añadir el archivo
constructor(private _uploadService: UploadServe) => Creamos una variable de tipo UploadService

this._projectService.saveProject(this.project).suscribe(); => Creamos el método para guardar el formulario importando desde project.service.ts el apartado

this._projectService => Llamamos al parámetro invocado anterior
.saveProject(this.project) => Invocamos al método (lo tenemos en project.service.ts) y el parámetro que añadimos es this.project
.suscribe(response=>{},error=>{console.log(<any>error);} ➔ permite reemplazar los archivos internos mediante la librería observable

this._uploadService.makeFileRequest(Global.url + "upload-image/" + this.project._id, [], this.filesToUpload, 'image').then((result: any) => {}); =>

```
Añadimos el método para subir archivos del formulario (es un método diferente) desde upload.service.ts
this._uploadService => Llamamos al método para cargar archivo
.makeFileRequest(Parametros) => Añadimos el método desde upload.service.ts
Global.url => Es la URL que tenemos
"upload-image/" => Es como queremos que se quede guardada, sebe de ser acorde con el backend (02-proyecto-nodejs)
this.project._id => añadimos el id, recordemos que en el backend (02-proyecto-nodejs) es obligatorio (:id)
[] => Es un archivo vacio, no se utiliza de momento es para información secundaria
this.filesToUpload => Es el archivo que vamos a subir
"image" => Es la terminación como se va a cargar la imagen
.then((result:any,reject:any)=>{}  ==> Es una promesa que va a dar un resultado cuando da resultado o rechazo
this.status = "success" => Mediante la variable status definimos que el formulario se reinicie y se ejecute la operacion
form.reset(); => Genera que se borre el formulario

fileChangeEvent(fileInput:any){} => Este metodo genera que añadamos del la imagen el archivo con el contenido solamente de la imagen
this.filesToUpload = <Array<File>>fileInput.target.files => convertimos la imagen en un array de tipo file
```

10) GENERAR VALIDACION DE FORMULARIO EN HTML Y CSS

CREATE.COMPONENT.HTML

```
    <div class="message success" *ngIf="status == 'success'">
        El proyecto se ha creado correctamente puedes <a href="#">verlo aquí</a>
    </div>
    <div class="message failed" *ngIf="status == 'failed'">
        El proyecto no se ha creado, vuelva a intentarlo
    </div>



<p>
   <label for="image">Imagen del proyecto</label>
   <input type="file" name="image" placeholder="Subir Imagen" required
    (change)="fileChangeEvent($event)"/>
 </p>
                        *** METODO CHANGE PARA VALIDAR FORMULARIO ***

(change)="fileChangeEvent($event)"/> => Genera que cuando haya cualquier cambio el método FileChangeEvent tome el evento generado
<*ngIf="status == 'success'"> => Generamos el evento si sucede que sea success
```

CREATE.COMPONENT.CSS

```
.container{
    padding: 50px;
    padding-top: 10px;
}
.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"]{
    display: block;
    font-size: 17px;
    padding: 5px;
    margin-top: 15px;
    color: white;
    background-color:rgba(53,103,164,1);
    border: 1px solid rgba(53,103,164,1);
    cursor: pointer;
}

form button:hover, form input[type="submit"]:hover{
    background-color:rgba(53,103,164,0.8);
    border: 1px solid rgba(53,103,164,1);
}

.form_error{
    padding: 3px;
    font-size: 13px;
    color: white;
    background-color: red;
    margin-left: 5px;
}

form input[disabled]{
    opacity: 0.5;
    cursor:not-allowed;
}


.message{
    width: 50%;
    padding: 5px;
    border: 1px solid #eee;
```

```
    border-radius: 15px;
}

.success{
    background-color: green;
    color: white;
}

.message a{
    color: white;
}

.failed{
    background-color: red;
    color: white;
}
```

11) SUBIR IMAGEN DESDE PROYECTO

a. CREAMOS EL METODO MIKEFILEREQUEST EN UPLOAD.SERVICE.TS

UPLOAD.SERVICE.TS

```
import { Injectable } from "@angular/core";
import { Global } from "./global";

@Injectable()
export class UploadService {
    public url: string;

    constructor() {
        this.url = Global.url;
    }
    makeFileRequest(url: string, params: Array<string>, files: Array<File>, name: string){
        return new Promise(function(resolve, reject){
            var formData:any = new FormData();
            var xhr = new XMLHttpRequest();

            for(var i = 0; i < files.length; i++){
                formData.append(name, files[i], files[i].name);
            }

            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4){
                    if(xhr.status == 200){
                        resolve(JSON.parse(xhr.response));
                    }else{
                        reject(xhr.response);
                    }
                }
            }

            xhr.open('POST', url, true);
            xhr.send(formData);
        });
    }
```

```
    makeFileRequest(url: string, params: Array<string>, files: Array<File>, name: string){
        return new Promise(function(resolve, reject){
            var formData:any = new FormData();
            var xhr = new XMLHttpRequest();

            for(var i = 0; i < files.length; i++){
                formData.append(name, files[i], files[i].name);
            }

            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4){
                    if(xhr.status == 200){
                        resolve(JSON.parse(xhr.response));
                    }else{
                        reject(xhr.response);
                    }
                }
            }
            xhr.open('POST', url, true);
            xhr.send(formData);
        });
    }
}
```

*** METODO PARA SUBIR ARCHIVOS ***

```
import { Injectable } => LLAMAMOS AL INJECTABLE
import { Global } from => LLAMAMOS A LA URL GLOBAL
@Injectable() => INVOCAMOS AL DECORADOR PARA EL EJECUTABLE
export class UploadService {} => GENERAMOS LA CLASE QUE VA A EXPORTAR EL METODO
public url: string; => DEFINIMOS EL PARAMETRO URL
constructor() {} => LLAMAMOS AL CONSTRUCTOR PARA QUE SE EJECUTE EL METODO
this.url = Global.url => LE VAMOS VALOR A LA URL POR EL GLOBAL.URL

makeFileRequest(url: string, params: Array<string>, files: File[],name:string){} =>
 GENERAMOS EL METODO PARA SUBIR ARCHIVOS, MEDIANTE PETICION AJAX CLASICA, DEBE DE TENER UNOS
 PARAMETROS OBLIGATORIOS
url: string => PARA LA PETICION AJAX
params: Array<string>=> DEFINIMOS EL TIPO DE DATO QUE VAMOS A RECIBIR
files: File[] => DEFINIMOS EL FORMATO DEL ARCHIVOS QUE VAMOS A RECIBIR
name:string => DEFINIMOS EL FORMATO CON EL QUE VAMOS A DARLE EL NOMBRE

return new Promise(function(resolve, reject) {} => GENERAMOS COMO RESPUESTA UNA PROMESA CON
                                                    VALOR RESOLVE(RESPUESTA) O
REJECT(RECHAZO)
var formData:any = new FormData() => GENERAMOS UNA VARIABLE QUE GENERE EL FORMATO DE
FORMULARIO
 EL FORMULARIO YA LO TENEMOS DEFINIDO EN OTRO METODO
var xhr = new XMLHttpRequest() => GENERAMOS VAR PARA PETICION ASINCRONA AJAX

for (var i = 0;i<files.length;i++){formData.append(name, files[i], files[i].name);} =>
CREAMOS UN FOR PARA RECORRER LOS ARCHIVOS Y QUE ANADA EL ARCHIVO Y EL NOMBRE, SIEMPRE ASI
 var i = 0; → DEFINE LA VARIABLE I QUE EMPIECE EN 0
 i < files.length → DEFINE QUE SI LA VAR I ES INFERIOR A EL NO ARCHIVOS
 i++ → SIGNIFICA QUE LA VAR I AUMENTE +1
 formData.append → SE ANADE EN LA VAR FORMDATA NOMBRE (NAME), ANADE EL FICHERO A MEDIDA QUE
LLEGAN FILE[I], Y AGREGA EL NOMBRE DE LA ITERACION DEL FICHERO FILE[I].NAME



xhr.onreadystatechange = function () {} => CREAMOS LA PETICION AJAX MEDIANTE FUNCION ANONIMA

if (xhr.readyState == 4) {} => ES PARA COMPROBAR LA CONECTIVIDAD SIEMPRE ASI

if (xhr.status == 200) {} => LA PETICION HTTP DEBE ESTAR OK

resolve(JSON.parse(xhr.response)) => LA RESPUESTA DE PETICION AJAX LA CONVIERTE EN FORMATO
JSON

}else{reject(xhr.response);} => SI HAY RECHAZO MUESTRA LA PETICION EN CONSOLA


xhr.open("POST", url, true) => CREAMOS LA PETICION POR POST CON LA URL SI ESTA TRUE

xhr.send(formData) => ENVIAMOS EL FORMULARIO EMULADO
```

12) LISTADO DE PROYECOS DEL PORTAFOLIO:

a. EN PROJECT.SERVICE.TS:

CREAMOS EL MÉTODO PARA SACAR DE LA BASE DE DATOS Y CONSUMIR LA API TODOS NUESTROS PROYECTOS, UN DOCUMENTO JSON

CREATE.COMPONENT.HTML

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Project } from "../models/project";
import { Global } from "./global";


@Injectable()
export class ProjectService {
    public url: string;

    constructor(
        private _http: HttpClient
    ) {
        this.url = Global.url;
    }
    testService() {
        return "Probando el servicio de angular";
    }
    saveProject(project: Project) { /* --- GUARDAR FORMULARIO --- */
        let params = JSON.stringify(project);
        let headers = new HttpHeaders().set('Content-Type', 'application/json');
        return this._http.post(this.url + "save-project", params, { headers: headers });
    }
    getProjects(): Observable<any> { /* -- EXTRAER PROYECTO DE BBDD--- */
        let headers = new HttpHeaders().set('Content-Type', 'application/json');
        return this._http.get(this.url + 'projects', { headers: headers });
    }
}
```

*** METODO PARA EXTRAER PROYECTO DE LA BASE DE DATOS ***

```
getProjects(): Observable<any>{} => Generamos el método
let headers = new HttpHeaders().set('Content-Type', 'application/json') => Generamos la cabecera de la petición ajax
return this._http.get(this.url + 'projects', { headers: headers }) => Generamos la petición ajax por GET
```

b. EN PROJECT.COMPONENT.TS:

CREAMOS EL MÉTODO GETPROYECT (IGUAL QUE EL NOMBRE ANTERIOR PARA EVITAR CONFUNSION).

PROJECTS.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { Project } from '../../models/project';
import { ProjectService } from '../../services/project.service';
import { Global } from '../../services/global';
@Component({
  selector: 'app-projects',
  templateUrl: './projects.component.html',
  styleUrls: ['./projects.component.css'],
  providers: [ProjectService]
})
export class ProjectsComponent implements OnInit {
  public projects: Project[];
  public url: string;
  constructor(
    private _projectService: ProjectService → INJECTAMOS EL SERVICIO EN EL CONSTRUCTOR
  ) {
    this.url = Global.url;
  }
  ngOnInit() {
    this.getProjects(); → llamamos al método GETPROJECT para que se ejecute inmediato
  }
  getProjects() {
    this._projectService.getProjects().subscribe(
      response => {
        console.log(response);
        if (response.projects) {
          this.projects = response.projects;
        }
      }, error => {
        console.log(<any>error);
      });
  }
}
                                    *** METODO GETPROJECT ***
import { Project } from '../../models/project' => Extraemos el models Project
import { ProjectService } from '../../services/project.service' => Importamos el servicio, lo
tenemos que añadir providers:[ProjectService]
import { Global } from '../../services/global' => Extraemos el URL global del api
export class ProjectsComponent implements OnInit {} => Generamos el proceso de exportación
public projects: Project[] => Generamos un parámetro de project de tipo array
public url: string => Definimos el tipo url de string
constructor(private_projectService:ProjectService){} => Definimos en el constructor el servicio
privado
this.url = Global.url => Definimos el URL como tipo Global
ngOnInit(){this.getProject();} => Ejecuta el servicio getProject() al ejecutar
```

c. EN PROJECT.COMPONENT.HTML

CREAMOS UNA LISTA CON UN *NGFOR PARA QUE ARROJE TODAS LAS LISTAS QUE LLAMAMOS DE LA BBDD

PROJECTS.COMPONENT.HTML

```
<div class="container">
    <h2>Projectos</h2>
    <ul>
        <li *ngFor="let project of projects" class="project">
            <a [routerLink]="['/proyecto', project._id]">
                <div class="image">
                    <img src="{{url+'get-image/'+project.image}}" *ngIf="project.image"/>
                </div>
                <h3>{{project.name}}</h3>
                <p>{{project.category}}</p>
            </a>
        </li>
    </ul>
</div>
```

*** GENERAMOS LA LISTA DE LOS PROYECTOS ***

<li *ngFor =””> → Creamos un condicional ForEeach desde angular

<“let Name of NameFormData”> → Creamos que se recorra en función de una variable

13) LISTADO DE PROYECTOS DEL PORTAFOLIO:

USAMOS UNA NUEVA URL DEL API DE BACKEND PARA DEVOLVER UNA IMAGEN DESDE EL BACK END

a. CREAMOS EL CONTROLADOR: EN BACKEND → CONTROLLERS → PROJECT.JS

EN CONTROLLERS – PROJECT.JS

```
var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS
var fs = require('fs');
var path = require('path');

var controller = {
    getImageFile: function (req, res) {
        var file = req.params.image;
        var path_file = './uploads/' + file;

        fs.exists(path_file, (exists) => {
            if (exists) {
                return res.sendFile(path.resolve(path_file));
            } else {
                return res.status(200).send({
                    message: "No existe la imagen..."
                });
            }
        });
    }
};

module.exports = controller;
```

getImageFile: function (req,res){}	Es el método que elegimos para poder extraer una imagen del backend
Var file = req.params.image;	Estamos extrayendo la imagen de los parámetros que tenemos por parte del formulario (pintados en la página)

Var path_file = "./uploads/"+file;	Creamos una var con la ubicación del archivo más
Fs.exist(path_file(err,exist)=>{});	Utilizamos la librería File System, con el cual comprobamos si existe ruta, se realice callback
If(exist){}	Comprueba con condicional IF si existe
Return res.sendFile(path.resolve(path_file));	var path = require("path"); Es un módulo dentro de NodeJs encargado de poder extraer las rutas dentro del equipo path.resolve("ubicacion"); Se encarga de devolver el archivo mediante la ubicación seleccionada
Return res.status(200).send({});	Creamos la función else en caso de error

b. CREAMOS LA RUTA EN EL BACKEND → ROUTES → PROJECT.TS

EN ROUTES – PROJECT.JS

```
'use strict'

var express = require('express');
var ProjectController = require('../controllers/project');
var router = express.Router();
var multipart = require('connect-multiparty');
var multipartMiddleware = multipart({ uploadDir: './uploads' });

router.get('/get-image/:image', ProjectController.getImageFile);

module.exports = router;
```

14) MAQUETAMOS EL LISTADO DEL PROYECTO DEL PORTAFOLIO

RECORDAMOS QUE TODOS LOS ESTILOS CSS LOS TENEMOS EN ASSETS → CSS- STYLE.CSS

a. NOS DIRIGIMOS A ASSETS → CSS → STYLE.CSS

```
.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;
}
```

15) CREAMOS EL COMPONENTE DETAIL PARA ARROJAR INFORMACION DE COMPONENT PROJECT

EL COMPONENT/DETAIL VA A SERVIR PARA HACER UN HIPERVINCULO Y PODER ARROJAR OTRA PAGINA DONDE SE VERA TODO CON MAS DETALLE, SE HACE DESDE CONSOLA

a. CREAMOS EL COMPONENTE → COMPONENT/DETAIL

```
Ng g component component/detail
```

b. CREAMOS LA RUTA EN APPROUTING → PARA CREAR UNA RUTA NUEVA AL COMPONENTE

EN APPROUTING

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

import { DetailComponent } from './components/detail/detail.component';

const appRoutes: Routes = [
    {path: 'proyecto/:id', component: DetailComponent},
→ CREAMOS LA RUTA DE DETAIL.COMPONENT, LA CUAL TENDRA LA QUERY PROYECTO

export const appRoutingProviders: any[] = [];
export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

c. CREAMOS EL ENLACE DENTRO DE COMPONENT/PROJECT → EN HTML

EN APPROUTING

```
<div class="container">
    <h2>Projectos</h2>

    <ul>
        <li *ngFor="let project of projects" class="project">
            <a [routerLink]="['/proyecto', project._id]">
              → INTEGRAMOS EN CADA PROYECTO UN HIPERVINCULO PARA QUE NOS LLEVE AL
                COMPONENTE DETAILS JUNTO CON SU ID
                <div class="image">
                    <img src="{{url+'get-image/'+project.image}}" *ngIf="project.image"/>
                </div>
                <h3>{{project.name}}</h3>
                <p>{{project.category}}</p>
            </a>
        </li>
    </ul>
</div>
```

<a [routerLink]=””>	Creamos una ruta en angular
“[‘/proyecto’,Project_id]”	Damos seguimiento al link al cual llamaremos el router Link

d. MODIFICAMOS LA MAQUETACION EN PROJECT.CSS

ASSETS → CSS → STYLE.CSS

```css
.project .image{
    width: 100%;
    height: 150px;
    overflow: hidden;
}

.project img{
    width: 100%;
}

.project a{
    text-decoration: none;
    color: black;
}
```

e. CREAMOS EL METODO → GETPROJECT PARA PODER RECOGER LA INFORMACIÓN DE LA BASE DE DATOS

SERVICE → PROJECT.SERVICE.TS

```typescript
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { Project } from '../models/project';
import { Global } from './global';

@Injectable()
export class ProjectService{
    public url:string;

    constructor(
        private _http: HttpClient
    ){
        this.url = Global.url;
    }
    getProject(id): Observable<any>{
        let headers = new HttpHeaders().set('Content-Type', 'application/json');
        return this._http.get(this.url+'project/'+id, {headers: headers});
    }
```

getProject(id):	Creamos un método que necesita una variable que será el ID
Observable<any>	Es para llamadas HTTP, de cualquier tipo o segun una interface/modelo
Let headers = new HttpHeaders().set('Content-type', 'application/json');	Creamos una cabecera la cual tendrá la estructura de las respuesta
return this._http.get(this.url+'project/'+id, {headers: headers});	Devolverá una respuesta .get con el url y debe de ser tipo cabecero.

f. AÑADIMOS ROUTERLINK AL COMPONENT COMPONENT/CREATE.TS

CREATE.COMPONENT.HTML

```html
<div class="container" *ngIf="project">
    <h2>{{title}}</h2>

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

```
        El proyecto se ha creado correctamente, puedes <a [routerLink]="['/proyecto',
save_project._id]">verlo aquí</a> → AÑADIMOS EL ROUTERLINK AL PROYECTO CREADO
    </div>

    <div class="message failed" *ngIf="status == 'failed'">
        El proyecto NO SE HA PODIDO CREAR
    </div>
```

g. VAMOS A DAR LE VALOR A LA PROPIEDAD PUBLIC SAVE_PROJECT EN CREATE.COMPONENT.TS

CREATE.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { Project } from "../../models/project";
import { ProjectService } from "../../services/project.service";
import { UploadService } from '../../services/upload.service';
import { Global } from '../../services/global';

@Component({
  selector: 'app-create',
  templateUrl: './create.component.html',
  styleUrls: ['./create.component.css'],
  providers: [ProjectService, UploadService]
})
export class CreateComponent implements OnInit {

    public title: string;
    public project: Project;
    public save_project;  → CREAMOS LA VARIABLE
    public status: string;
    public filesToUpload: Array<File>;

    constructor(
        private _projectService: ProjectService,
        private _uploadService: UploadService
    ){
        this.title = "Crear proyecto";
        this.project = new Project('','','','',2019,'','');
    }

    ngOnInit() {
    }

    onSubmit(form: any){

        // Guardar datos básicos
        this._projectService.saveProject(this.project).subscribe(
            response => {
                if(response.project){

                    // Subir la imagen
                    if(this.filesToUpload){
                        this._uploadService.makeFileRequest(Global.url+"upload-image/"
                        +response.project._id, [], this.filesToUpload, 'image')
                        .then((result:any) => {
                            this.save_project = result.project;
                             → ANADIMOS EL PROYECTO CREADO A LA VAR, PARA PODERLA
```

```
                            LLAMAR EN COMPONENT/DETAILS
                        this.status = 'success';
                        form.reset();
                    });
                }else{
                    this.save_project = response.project;
                    this.status = 'success';
                    form.reset(); //SE VACIA EL FORMULARIO
                }

            }else{
                this.status = 'failed';
            }
        },
        error => {
            console.log(<any>error);
        }
    );
  }
```

Creamos la variable **save_project** la cual se utilizará para obtener en una VAR el proyecto subido.

public save_project;	Creamos la variable encargada de obtener el proyecto creado

```
this._projectService.saveProject(this.project).subscribe(
        response => {
            if(response.project){
                if(this.filesToUpload){
                    this._uploadService.makeFileRequest(Global.url+
"upload-image/"+response.project._id, [],this.filesToUpload, 'image')
                        .then((result:any) => {
                      this.save_project = result.project;
                      this.status = 'success';
                      form.reset();
                      });
                }else{
                 this.save_project =response.project;
                    this.status = 'success';
                    form.reset();
                }
            }else{
                this.status = 'failed';
            }
        },
        error => {
            console.log(<any>error);
        }
    );
  }
```

This.save_project = result.project;

Vamos a estar llenando la variable save_project con el resultado creado del formulario

16) COMPLETAMOS TODA LA LOGICA DE DETAIL.COMPONENT.TS

a. MOSTRAR INFORMACIÓN AL DETALLE

DETAIL.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { Project } from '../../models/project';
import { ProjectService } from '../../services/project.service';
import { Global } from '../../services/global';
import { Router, ActivatedRoute, Params } from '@angular/router';
→ SON MODULOS NECESARIOS PARA PODER ACCEDER AL ROUTER

@Component({
  selector: 'app-detail',
  templateUrl: './detail.component.html',
  styleUrls: ['./detail.component.css'],
  providers: [ProjectService]  → ANADIMOS EL SERVICIO
})
export class DetailComponent implements OnInit {
  public url: string;          → DEFINIMOS LA RUTA
  public project: Project;     → DEFINIMOS EL PROYECTO
  public confirm: boolean;

  constructor(
    private _projectService: ProjectService,
    private _router: Router,       → DEFINIMOS LOS OBJETOS DEL TIPO
    private _route: ActivatedRoute → DEFINIMOS LOS OBJETOS DEL TIPO
  ){
    this.url = Global.url;
    this.confirm = false;
  }

  ngOnInit(){
    this._route.params.subscribe(params => {
        → CON ESTO RECOGEMOS EL PARAMETRO QUE VIENE DE LA URL
        let id = params.id;

        this.getProject(id); → CON ESTO LLAMAMOS AL METODO Y LE INCLUIMOS ID
    });
  }

  getProject(id){
    this._projectService.getProject(id).subscribe(
        → LLAMAMOS AL METODO GETPROJECT DENTRO DEL SERVICIO PROJECTSERVICE
        response => {
            this.project = response.project;
        → CON ESTO RELLENAMOS LA VAR PROYECT CON EL PROTECYECTO
        },
        error => {
            console.log(<any>error);
        }
    )
  }

  setConfirm(confirm){
    this.confirm = confirm;
```

```
  }

  deleteProject(id){
    this._projectService.deleteProject(id).subscribe(
        response => {
            if(response.project){
                this._router.navigate(['/proyectos']);
            }
        },
        error => {
            console.log(<any>error);
        }
    );
  }

}
```

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

Importamos a Component (encargado de crear los compoentes de Angular)
Importamos a OnInit (Encargado de ejecutar métodos cuando se abra la página)

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

Es el encargado de llamar al modelo schema que creamos para el formulario

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

Llamamos al servicio encargado de los métodos del backend, como arrojar información y modificarla

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

Llamamos a la ruta, define como global.service

b. CREAR EN DETALLES TODO EL FORMULARIO

DETAIL.COMPONENT.HTML

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

    <div class="image" *ngIf="project.image">
        <img src="{{url+'get-image/'+project.image}}"/>
    </div>
    <div class="data">
        <h1>{{project.name}}</h1>
        <h3>{{project.description}}</h3>
        <p>{{project.category}}</p>
        <p>{{project.langs}}</p>
        <p *ngIf="confirm == false">
            <a [routerLink]="['/editar-proyecto', project._id]" class="button-edit">Editar</a>
            <a (click)="setConfirm(true)" class="button-delete">Borrar</a>
        </p>

        <p *ngIf="confirm == true">
            <span class="confirm">¿Estas seguro de eliminar este proyecto?</span>
            <a (click)="setConfirm(false)" class="button-edit">Cancelar</a>
            <a (click)="deleteProject(project._id)" class="button-delete">Eliminar
definitivamente</a>
        </p>
    </div>
</div>
```

```
<div class="container" *ngIf="project"></div>
```

Creamos el div contenedor para el diseño de CSS,
Añadimos *ngIf para que se valide SOLO si existe proyecto

```
<div class="image" *ngIf="project.image">
        <img src="{{url+'get-image/'+project.image}}"/>
    </div>
```

Creamos una clase “image” y si tiene imagen *ngIf arroje la imagen desde la url

```
<div class="data">
   <h1>{{project.name}}</h1>
   <h3>{{project.description}}</h3>
   <p>{{project.category}}</p>
   <p>{{project.langs}}</p>
</div>
```

Interpolamos los datos de la base de datos “backend”

```
<p *ngIf="confirm == false">
  <a [routerLink]="['/editar-proyecto', project._id]" class="button-edit">Editar</a>
  <a (click)="setConfirm(true)" class="button-delete">Borrar</a>
</p>
```

Creamos un subproceso que en caso de que le demos click a borrar el botón luego nos genera otra pregunta en la que se cancela.

```
<p *ngIf="confirm == true">
 <span class="confirm">¿Estas seguro de eliminar este proyecto?</span>
 <a (click)="setConfirm(false)" class="button-edit">Cancelar</a>
 <a (click)="deleteProject(project._id)" class="button-delete">El definitivamente</a>
</p>
```

Creamos un botón que generará nuevamente otra pregunta si se desea eliminar, el cual llamara a la función deleteProject(Project._id), siendo el id que se va a eliminar.

c. CREAMOS LOS ESTILOS CSS DE LA IMAGEN Y LA DATA

ASSETS → CSS → STYLE.CSS

```
.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;
```

```
}
```

d. CREAMOS LOS ESTILOS DEL BOTON DE EDITAR Y BORRAR

ASSETS → CSS → STYLE.CSS

```
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;
}
```

17) CREAMOS LA LOGICA DEL BOTON DE "BORRAR PROYECTO"

a. VAMOS A CREAR UN SERVICIO DENTRO DE PROJECT.SERVICE.TS, EL CUAL VA A UTILIZAR EL METODO HTTP DELETE

SERVICES / PROJECT.SERVICE.TS

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { Project } from '../models/project';
import { Global } from './global';

@Injectable()
export class ProjectService{
    public url:string;

    constructor(
        private _http: HttpClient
    ){
        this.url = Global.url;
    }

    deleteProject(id): Observable<any>{
        let headers = new HttpHeaders().set('Content-Type', 'application/json');
        return this._http.delete(this.url+'project/'+id, {headers: headers});
    }
}
```

deleteProject(id):	Definimos el método que se encargara de borrar un proyecto según el id
Observable<any>	Seteamos para que nos devuelva en formato observable, es un formato de tipo HTTP, pero también podemos hacer petición según interfaz/modelo
let headers = new HttpHeaders() .set('Content-Type', 'application/json');	Es necesario para recuperar una petición de cabecero, todas las peticiones lo necesitan
return this._http.delete(this.url+'project/'+id, {headers: headers});	El método nos devolverá eliminar según el url y teniendo el cuenta la estructura de la cabecera

b. EN DETAIL.COMPONENT.TS, CREAMOS UN METODO QUE LLAMARÁ AL METODO DEL SERVICIO, TENDRÁ EL MISMO NOMBRE

DETAIL.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { Project } from '../../models/project';
import { ProjectService } from '../../services/project.service';
import { Global } from '../../services/global';
import { Router, ActivatedRoute, Params } from '@angular/router';

@Component({
  selector: 'app-detail',
  templateUrl: './detail.component.html',
  styleUrls: ['./detail.component.css'],
  providers: [ProjectService]
})
export class DetailComponent implements OnInit {
  public url: string;
  public project: Project;
  public confirm: boolean;

  constructor(
    private _projectService: ProjectService,
```

```
    private _router: Router,
    private _route: ActivatedRoute
  ){
    this.url = Global.url;
    this.confirm = false;
  }

  ngOnInit(){
    this._route.params.subscribe(params => {
        let id = params.id;
        this.getProject(id);
    });
  }
  deleteProject(id){
    this._projectService.deleteProject(id).subscribe(
        response => {
            if(response.project){
                this._router.navigate(['/proyectos']);
            }
        },
        error => {
            console.log(<any>error);
        }
    );
  }
}
```

deleteProject(id){}	Es un método que llamara al servicio para borrar la documentación
This._projectService.deleteProject(id)	Llama al servicio para que ejecute la función de borrar con el id
.suscribe(response, error)	Crea una promesa que devolverá una de las dos respuestas
if(response.project){ this._router.navigate(['/proyectos']); }	Una vez se efectué el método llamado al servicio, haremos que nos devuelva a la página principal.

c. **DENTRO DE DETAIL.COMPONENT.HTML VAMOS A MODIFICAR EL BOTON DE BORRAR PARA QUE LLAME AL METODO DENTRO DE DETAIL.COMPONENT.TS/DELETEPROJECT**

DETAIL.COMPONENT.HTML

```
<div class="container" *ngIf="project">
    <div class="data">
        <p *ngIf="confirm == true">
            <span class="confirm">¿Estas seguro de eliminar este proyecto?</span>
            <a (click)="setConfirm(false)" class="button-edit">Cancelar</a>
            <a (click)="deleteProject(project._id)"
                class="button-delete">Eliminar definitivamente</a>
        </p>
    </div>
</div>
```

(click)=”deleteProject(Project._id)”	Llamamos mediante un evento click la función que borre el proyecto pidiendo el id

18) ACTUALIZANDO PROYECTOS MEDIANTE EL COMPONENTE EDIT.COMPONENT.TS:

a. CREAMOS UNA PAGINA DE EDICION MEDIANTE

```
Ng g component component/edit
```

b. AÑADIMOS EL COMPONENTE Y LA RUTA EN APP.ROUTING:

APP.ROUTING.TS

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

import { EditComponent } from './components/edit/edit.component';

const appRoutes: Routes = [
    {path: 'editar-proyecto/:id', component: EditComponent},
];

export const appRoutingProviders: any[] = [];
export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

c. DENTRO DE DETAIL.COMPONENT EN EL BOTON DE EDITAR, EL CUAL TENDRA LA RUTA DEL EDIT.COMPONENT.HTML

DETAIL.COMPONENT.HTML

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

    <div class="data">
        <p *ngIf="confirm == false">
            <a [routerLink]="['/editar-proyecto', project._id]" class="button-edit">Editar</a>
            <a (click)="setConfirm(true)" class="button-delete">Borrar</a>
        </p>

        <p *ngIf="confirm == true">
            <span class="confirm">¿Estas seguro de eliminar este proyecto?</span>
            <a (click)="setConfirm(false)" class="button-edit">Cancelar</a>
            <a (click)="deleteProject(project._id)" class="button-delete">
             Eliminar definitivamente</a>
        </p>
    </div>
</div>
```

<a [routerLink]= "['/editar-proyecto', project._id]" class="button-edit">Editar	Desde el botón de editar direccionaremos a la ruta donde se encontrara el componente edit.component.html

d. CREAMOS EN EL SERIVCIO EL METODO UPDATESERVICE, EL CUAL SE ENCARGA DE REEMPLAZAR LOS DATOS

SERVICE / PROJECT.SERVICE.TS

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { Project } from '../models/project';
import { Global } from './global';

@Injectable()
export class ProjectService{
    public url:string;

    constructor(
        private _http: HttpClient
    ){
        this.url = Global.url;
    }
    updateProject(project): Observable<any>{
        let params = JSON.stringify(project);
        let headers = new HttpHeaders().set('Content-Type', 'application/json');
        return this._http.put(this.url+'project/'+project._id, params, {headers: headers});
    }
}
```

UpdateProject(Project):Observable<any>	Creamos el método que va a reemplazar/editar la información
Let params = Json.stringify(Project);	Creamos una variable, la cual será el proyecto que lo recibimos como un objeto y lo transformamos en un tipo string
let headers = new HttpHeaders() .set('Content-Type', 'application/json');	Creamos cabeceras, fundamental para las peticiones
return this._http.put (this.url+'project/'+project._id, params, {headers: headers});	Devolvera una petición put (reemplazar), la cual será con el url e id y con los cambios en params

e. CARGAMOS EL COMPONENTE Y CREAMOS LA RUTA EN APP.ROUTING.TS

APP.ROUTING.TS

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

import { EditComponent } from './components/edit/edit.component';

const appRoutes: Routes = [
    {path: 'editar-proyecto/:id', component: EditComponent},

];

export const appRoutingProviders: any[] = [];
export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

f. CARGAMOS LA INFORMACION EN EDIT.COMPONENT.TS

EDIT.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { Project } from '../../models/project';
import { ProjectService } from '../../services/project.service';
import { UploadService } from '../../services/upload.service';
import { Global } from '../../services/global';
import { Router, ActivatedRoute, Params } from '@angular/router';

@Component({
  selector: 'app-edit',
  templateUrl: '../create/create.component.html',
  → UTILIZAREMOS EL MISMO FORMATO HTML EN DETAIL.COMPONENT.HTML EN EDIT.COMPONENT.HTML
  styleUrls: ['./edit.component.css'],
  providers: [ProjectService, UploadService]
 → VOLVEMOS A LLAMAR A LOS SERVICIOS EJECUTAMOS EN DETAIL.COMPONENT.TS
})
export class EditComponent implements OnInit {

    public title: string;
    public project: Project;
    public save_project;
    public status: string;
    public filesToUpload: Array<File>;
    public url: string;

    constructor(
        private _projectService: ProjectService,
        private _uploadService: UploadService,
        private _route: ActivatedRoute,
        private _router: Router
    ){
        this.title = "Editar proyecto";
        this.url = Global.url;
    }

  ngOnInit(){
```

```
    this._route.params.subscribe(params => {
        let id = params.id;

        this.getProject(id);
    });
  }

  getProject(id){
    this._projectService.getProject(id).subscribe(
        response => {
            this.project = response.project;
        },
        error => {
            console.log(<any>error);
        }
    )
  }

  onSubmit(){
    this._projectService.updateProject(this.project).subscribe(
        response => {
            if(response.project){

                // Subir la imagen
                if(this.filesToUpload){
                    this._uploadService.makeFileRequest(Global.url+"upload-
image/"+response.project._id, [], this.filesToUpload, 'image')
                    .then((result:any) => {
                        this.save_project = result.project;
                        this.status = 'success';
                    });
                }else{
                    this.save_project = response.project;
                    this.status = 'success';
                }

            }else{
                this.status = 'failed';
            }
        },
        error => {
            console.log(<any>error);
        }
    );
  }

    fileChangeEvent(fileInput: any){
        this.filesToUpload = <Array<File>>fileInput.target.files;
    }
}
```

g. EN DETAIL.COMPONENT.HTML CREAMOS LOS CONDICIONALES *NGIF PORQUE LAS PETICIONES AJAX SON MAS LENTAS QUE LAS PETICIONES JAVASCRIPT

DETAIL.COMPONENT.HTML

```
<div class="container" *ngIf="project">
→ *NGIF IMPORTANTE PARA QUE NO TENGAMOS PROBLEMAS EN LAS PETICIONES AJAX
    <div class="image" *ngIf="project.image">
        <img src="{{url+'get-image/'+project.image}}"/>
    </div>
```

INTEGRAR LIBRERIAS EXTERNAS EN ANGULAR:

1. PODEMOS INTEGRAR VARIAS LIBRERIAS EN CUALQUIER PROYECTO DE ANGULAR MEDIANTE EL GESTOR NPM, PODEMOS BUSCAR EN LA PAGINA WEB, VAMOS A PONER EJEMPLO CON LA LIBRERÍA DE JQUERY:
2. INSTALAMOS LA LIBRERÍA DE JQUERY MEDIANTE
3. NPM I JQUERY - - SAVE (SAVE PARA PROYECTO LOCAL)
4. LO VEREMOS CARGADO EN PACKAGE.JSON/DEPENDIENCIES
5. PARA PODER UTILIZARLO EN EL PROYECTO HAY QUE INVOCARLO MEDIANTE ANGULAR.JSON/SCRIPTS

ANGULAR.JSON:

```
"scripts": [
            "node_modules/jquery/dist/jquery.min.js",
            "src/assets/bxslider/dist/jquery.bxslider.min.js"
          ]
```

6. COMO EJEMPLO PODEMOS HACER QUE AL HACER CLICK, EL NAVBAR CAMBIE DE TAMAÑO

CONTACT.COMPONENT.TS:

```
ngOnInit() {
    $('logo').click((e)=>{
     e.preventDefault();
  → IMPORTANTE → ES PARA EVITAR QUE UN LINK REDIRECCIONE

     $('header').css('background','green').css('height','50px');
   });
 }
```

CREAR UN SLIDER CON JQUERY Y UTILIZAR @INPUT Y @OUTPUT, VIEWCHILD Y DIRECTIVAS PERSONALIZADAS:

1. CREAR UN SLIDER CON JQUERY:

a. Podemos crear un Slider con jQuery al igual que con Bootstrap, para hacerlo primero tendremos que hacer un div que incluya las imágenes:

SLIDER.COMPONENT.HTML:

```
Slider: <strong>{{anchura}}</strong>, Captions: <strong>{{captions}}</strong>
→ CREAMOS UN COMPONENTE QUE ESTARA INTEGRANDO DENTRO DE OTRO MEDIANTE @INPUT, EL CUAL  NOS INFORMARA DE LA ANCHURA

<div class="galeria">
  <div><img src="assets/img/coffee1.jpg" title="Funky roots"></div>
  <div><img src="assets/img/coffee2.jpg" title="The long and winding road"></div>
  <div><img src="assets/img/coffee3.jpg" title="Happy trees"></div>
</div>

<button (click)="lanzar($event)">Conseguir autor</button>
→ CREAMOS OTRO BOTON CONDICIONAL EL CUAL LANZARA EL EVENTO QUE LLAMARA A TRAVES DEL NOMBRE DEL AUTOR EN SLIDER.COMPONENT.TS
```

SLIDER.COMPONENT.TS:

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
declare var $:any;

@Component({
  selector: 'slider',
  templateUrl: './slider.component.html',
  styleUrls: ['./slider.component.css']
})
export class SliderComponent implements OnInit {

  @Input() anchura: number;
  @Input('etiquetas') captions: boolean;
→ LOS INPUT LOS LLAMAMOS DESDE EL PADRE CONTACT.COMPONENT HACIA SLIDER.COMPONENT
  @Output() conseguirAutor = new EventEmitter();
→ EL OUTPUT LO EMITIMOS NOSOTROS DESDE SLIDER.COMPONENT A CONTACT.COMPONENT, EN
EL CUAL MEDIANTE UN BOTON SE HARA LA PETICION
→ NEW EVENTEMITTER GENERA UN EVENTO CAPTURADO EN ESTE CASO POR EL BOTON

  public autor: any;

  constructor(){

    this.autor = {
      nombre: "Víctor Robles",
      website: "victorroblesweb.es",
      youtube: "Victor Robles WEB"
    };

  }

  ngOnInit() {
    $("#logo").click(function(e){
      e.preventDefault();

      $("header").css("background","green")
             .css("height","50px");
    });

    $('.galeria').bxSlider({
      mode: 'fade',
      captions: this.captions,
      slideWidth: this.anchura
    });

    // Lanzar evento
     this.conseguirAutor.emit(this.autor);
  }

  lanzar(event){
    this.conseguirAutor.emit(this.autor);
  }
```

```
→ ES UN METODO EL CUAL EMITIRA UN OUTPUT CON TODA LA INFORMACION DEL AUTOR, ESTE LUEGO LO TRANSFORMAREMOS EN CONTACT.COMPONENT.HTML A UNA FUNCION CON EL EVENTO RECOGIDO → CONSEGUIRAUTOR()= “GETAUTOR($EVENT)”
}
```

.bxSlider({});	Genera el slider a través de las imágenes integradas dentro
Mode	Genera el tipo de transición entre las imágenes
Captions:	Es un booleano y genera el atributo entre las imágenes
sliderWidth	Genera el tamaño

2. UTILIZAR @INPUT (LLEVAR UN METODO DEL PADRE AL HIJO):

a. Dentro del componente padre contact.component.ts definimos el tipo de variable y en component.html lo utilizamos para importar en el hijo

CONTACT.COMPONENT.TS:

```
import { Component, OnInit, ViewChild } from '@angular/core';
declare var $:any;

@Component({
  selector: 'app-contact',
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.css']
})
export class ContactComponent implements OnInit {
  public widthSlider: number;
  public anchuraToSlider: any;
  public captions: boolean;
  public autor: any;

→ CREAMOS LAS PROPIEDADES QUE IMPORTAREMOS DEL PADRE CONTACT.COMPONENT AL HIJO
SLIDER.COMPONENT

  @ViewChild('textos') textos;
→ VIEWCHILD ES UN METODO MÁS RAPIDO DE LLAMAR DESDE .HTML CON #NOMBRE-DIRECTIVA

  constructor() {
    this.captions = true;
   }

  ngOnInit() {
  }

  cargarSlider(){
    this.anchuraToSlider = this.widthSlider;
→ CREAMOS UN BOTON QUE VA A CAMBIAR MEDIANTE @INPUT EL TAMAÑO
  }
  resetearSlider(){
    this.anchuraToSlider = false;
  }
  getAutor(event){
    this.autor = event;
  }
}
```

CONTACT.COMPONENT.HTML:

```
<div class="container">
  <h1>Contacto</h1>

  <div id="texto" #textos>
    <p appResaltado>
      Esta es la pagina de contacto, puedes enviarnos cualquier duda
    </p>
  </div>

  <div>Creación de Slider mediante Jquery:</div>
```

```
  Tamaño del slider:
  <input type="number" [(ngModel)]="widthSlider" />
  <button (click)="cargarSlider()">Cargar slider</button>
  <button (click)="resetearSlider()">Resetear slider</button>

  <div *ngIf="anchuraToSlider">
    <slider
      [anchura]="anchuraToSlider"
      [etiquetas]="captions"
      (conseguirAutor)="getAutor($event)"
    ></slider>
  </div>

  {{ autor.nombre + " " + autor.website }}
</div>
```

DIRECTIVAS:

1. Las directivas es un componente dentro de angular que permite crear un formato o diseño que se podrá utilizar en cualquier otro componente, este puede modificar el html/css o realizar diversas funciones es útil para modificar varios componentes.
2. Crear directivas: `ng g directive resultado`
3. Creará un resultado.directive.ts

RESULTADO.DIRECTIVE.TS:

```
import { Directive, ElementRef } from '@angular/core';
→ IMPORTAREMOS ELEMENTREF PARA PODER ACCEDER A LA INFORMACION RAIZ DEL .TEXTCONTENT Y .INNERHTML

@Directive({
  selector: '[appResaltado]'
})
export class ResaltadoDirective {

  constructor(public el: ElementRef){
 → DEFINIMOS DIRECTAMENTE UNA VARIABLE DE TIPO ELEMENTO REFERENCIADO DENTRO DEL
    CONSTRUCTOR PARA HACERLO MAS RAPIDO

  }

  ngOnInit(){
        var element = this.el.nativeElement;
        element.style.background = "red";
        element.style.padding = "20px";
        element.style.marginTop = "15px";
        element.style.color = "white";
           → ACCEDEMOS AL ELEMENTO Y REALIZAMOS CAMBIOS EN EL CSS
  }
}
```

RESULTADO.DIRECTIVE.TS:

```
<div class="container">
  <h1>Contacto</h1>

  <div id="texto" #textos>
    <p appResaltado> → INVOCAMOS A LA DIRECTIVA PARA HACER CAMBIOS EN CSS
      Esta es la pagina de contacto, puedes enviarnos cualquier duda
    </p>
  </div>
```