**Que es?**

Es un frame work de javascript que nos permite crear aplicaciones conocidas como SPA(Single Page Aplication). Quiere decir que crearemos aplicaciones que tendran unicamente HTML, aunque podemos crear los llamados componentes que hace que el diseño sea mas complejo

**Instalacion:**

Primero necesitamos nodejs

instalamos angular con: npm install -g @angular/cli

cli(command line interface)

**plugins para visual studio core:**

Angular v7 snippedts

Angular languaje servide

emmet

bootstrap 4

**Primera aplicacion:**

comando: ng new mi-primer-app

Se instalan varios modulos mas de 100 pero solo son para el proyecto en desarrollo

para levantar el servidor y poder visualizar el navegador usamos el

comando: ng serve –o

**Archivos basicos:**

app.component.ts

import { Component } from '@angular/core';//Se importa el componente core de angular

@Component({//Decorador con los datos del componente

selector: 'app-root', //nombre del selector con el que se llamara el control

templateUrl: './app.component.html',//template html osea la pagina que interactuara con el codigo

styleUrls: ['./app.component.css']//Estilos de la pagina

})

export class AppComponent {//Se exporta la clase

title = 'Hola mundo desde Angular';//Propiedad declarada

}

app.component.html

<p>{{title}}</p> //Aqui usamos la interpolacion que ponemos entre llaves la propiedad y se imp´rime en el navegador

app.module.ts

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

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

import { AppComponent } from './app.component';

import { PersonasComponent } from './personas/personas.component';

import { PersonaComponent } from './persona/persona.component';

@NgModule({

declarations: [

AppComponent, PersonasComponent, PersonaComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**Flujo de una aplicacion:**

El archvio main.ts levanta el flujo del modulo osea el archivo app.module.ts

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

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

import { AppComponent } from './app.component';

import { PersonasComponent } from './personas/personas.component';

import { PersonaComponent } from './persona/persona.component';

@NgModule({//Se declara el decorador del core

declarations: [

AppComponent, PersonasComponent, PersonaComponent

],

imports: [

BrowserModule //Se importaran otros modulos

],

providers: [],

bootstrap: [AppComponent]//Indica que archivo se va a levantar osea el pricipal que en este caso seria: app.components.ts

})

export class AppModule { }

y todo se muestra en el archivo index.html

**Creacion de un componente manual:**

en src/app se crean los componentes

creamos un nuevo folder con el nombre de personas

creamos un archivo personas.component.ts

codigo:

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

@Component({

selector: 'app-personas',//Nombre que usaremos en html se recomienda usar app- devemos poner atencion a que el nombre no se repita

templateUrl: './personas.component.html',//archivo html

export class PersonasComponent{

}

creamos un archivo que se llame personas.component.html

registramos en el app.module.ts el nuevo componente

import {PersonasComponent} from './personas/personas.component'

@NgModule({

declarations: [

AppComponent, PersonasComponent

],

y por ultimo en el archivo app.component.html llamamos al componente

<app-personas></app-personas>

**cli para crear componentes:**

comando: ng generate component presona ó ng g c persona

**Avitar tener un archivo component.html**

@component({

template: `<app-persona></app-persona>

{{titulo}}`

})

Se cambia templateUrl por template y entre stiks se coloca el código

**instalar bootstrap:**

comando:

npm install bootstrap --save

npm install jquery --save

npm install popper.js --save

ahora tenemos que modificar el archivo angular.json

en la parte de

"styles":[

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

"scripts": [

"node\_modules/jquery/dist/jquery.slim.min.js",

"node\_modules/popper.js/dist/umd/popper.min.js",

"node\_modules/bootstrap/dist/js/bootstrap.min.js"

]

**Concepto de interpolación:**

app.component.ts

import { Component } from '@angular/core';//Se importa el componente core de angular

@Component({//Decorador con los datos del componente

selector: 'app-root', //nombre del selector con el que se llamara el control

templateUrl: './app.component.html',//template html osea la pagina que interactuara con el codigo

styleUrls: ['./app.component.css']//Estilos de la pagina

})

export class AppComponent {//Se exporta la clase

title = 'Hola mundo desde Angular';//Propiedad declarada

}

app.component.html

<p>{{title}}</p> //Aqui usamos la interpolacion que ponemos entre llaves la propiedad y se imp´rime en el navegador

**Property binding:**

App.component.html

<div class="container">

<div class="row">

<div class="col">

<button [disabled]="agregarPersona"

>Agregar persona</button>

</div>

</div>

</div>

En este archivo agregamos la propiedad disabled del botón entre corchetes para indicarle a angular que el valor va a cambial y en el resultado de la variable entre comillas dobles

App.component.ts

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

@Component({

selector: 'app-personas',

templateUrl: './personas.component.html'

})

export class PersonasComponent{

agregarPersona=true;// Agregamos en la declaración de la clase la variable que usamos para relacionar el boroton

}

**Event binding:**

Esto sirve para asociar un componente con un evento

App-component.html

Para asociar el evento click solo escribimos la palabra click entre paréntesis y entre comillas el nombre del metodo

<div class="container">

<div class="row">

<div class="col">

<button [disabled]="!agregarPersona"

(click)="onClickBoton()"

>Agregar persona</button>

<p>{{mensaje}}</p>

</div>

</div>

</div>

app-component-ts:

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

@Component({

selector: 'app-personas',

templateUrl: './personas.component.html'

})

export class PersonasComponent{

agregarPersona=false;

mensaje="No se ha dado click";

onClickBoton (){

this. mensaje ="Diste click";

}

}

**Event binding para un input:**

Para un input debemos de agregar algo de codigo extra, casting, etc.

app-component.html

<div class="container">

<div class="row">

<div class="col">

<input type="text" (input)="onEscribirInput($event)">

<p>{{inputText}}</p>

</div>

</div>

</div>

// Usamos el evento input y mandamos como parámetro el mismo control usando $event

app-component-ts:

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

@Component({

selector: 'app-personas',

templateUrl: './personas.component.html'

})

export class PersonasComponent{

inputText:string="";

onEscribirInput(event: Event){

this.inputText=(<HTMLInputElement>event.target).value;

}

}

**Two way binding:**

Con el event binding podemos hacer que el control mande información al ts pero no lo contrario con two way binding podemos hacer que estén sincronizada una variable con el contenido del input

Primero tenemos que modificar el archivo app.module.ts

app.module.ts:

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

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

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

import { AppComponent } from './app.component';

import { PersonasComponent } from './personas/personas.component';

import { PersonaComponent } from './persona/persona.component';

@NgModule({

declarations: [

AppComponent, PersonasComponent, PersonaComponent

],

imports: [

BrowserModule,FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

app-component.html

<div class="container">

<div class="row">

<div class="col">

<input type="text" [(ngModel)]="inputAsociado">

<p>{{inputAsociado}}</p>

</div>

</div>

</div>

app.module.ts:

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

@Component({

selector: 'app-personas',

templateUrl: './personas.component.html'

})

export class PersonasComponent{

inputAsociado="Hola mundo";

}

**Directiva \*ngIf:**

<p \*ngIf="disteClick">{{mensajeAgregarPersona}}</p>

Si el valor es false el componente no se mostrara y si el valor es true el componente se mostrara

Asi se aplicaría el else

<p \*ngIf="disteClick; else sinPersona">{{mensajeAgregarPersona}}</p>

<ng-template #sinPersona>

<p>No se ha agregado ninguna persona</p>

</ng-template>

En este caso se mostraría el control ng-template mientras el valor de disteClick sea false

**Directiva \*ngFor**

Este sirve para crear varios controles por ejemplo todos los datos que hay en un array como veremos en el ejemplo

Primero creamos un archivo con el nombre persona.model.ts para crear una clase y ahí añadirle los elementos a iterar

persona.model.ts:

export class persona{

constructor(public nombre:string,public apellido:string){

}

}

app.component.ts

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

import {persona} from './persona.model';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

titulo = 'Lista de personas';

personas:persona[] = [new persona("Juan","Camaney"), new persona("Fabian", "Gonzales")];

nombrePersona:string;

apellidoPersona:string;

onAgregarPersona(){

this.personas.push(new persona(this.nombrePersona,this.apellidoPersona));

}

}

app.component.html

<form>

<input type="text" name="nombre" [(ngModel)]="nombrePersona" placeholder="Nombre">

<input type="text" name="apellido" [(ngModel)]="apellidoPersona" placeholder="Apellido">

<button (click)="onAgregarPersona()">Agregar</button>

<div \*ngFor="let persona of personas; let i = index">

{{i+1}}.- {{persona.nombre}} {{persona.apellido}}

</div>

</form>

En este caso como estamos poniendo los elementos input dentro de un form es necesario ponerle un name si no no funciona

**Property binding entre componentes:**

Esto sirve para mandar datos entre componente padre a hijo

app.component.html

<form>

<input type="text" name="nombre" [(ngModel)]="nombrePersona" placeholder="Nombre">

<input type="text" name="apellido" [(ngModel)]="apellidoPersona" placeholder="Apellido">

<button (click)="onAgregarPersona()">Agregar</button>

<div >

<app-persona \*ngFor="let personaArray of personas; let i = index"

[persona]="personaArray"

[index]="i"

></app-persona>

</div>

</form>

Agregamos el \*ngFor en el componente hijo y mandamos por directivas los valores [persona]

persona.component.ts

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

import { persona } from '../persona.model';

@Component({

selector: 'app-persona',

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

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

})

export class PersonaComponent {

@Input()persona:persona;

@Input()index:number;

}

Agregamos el control Input de angular/core y hacemos las directivas @Input()persona:persona;

persona.component.html

<p>{{index+1}}.- {{persona.nombre}} {{persona.apellido}}</p>

Colocamos el contenido del control con las variables declaradas

**Event binding entre componentes:**

Para mandar funciones entre componentes utilizamos este método primero tenemos que crear un nuevo componentes llamado formulario y mover ahí el formulario

formulario.component.html:

<form>

<input type="text" name="nombre" [(ngModel)]="nombrePersona" placeholder="Nombre">

<input type="text" name="apellido" [(ngModel)]="apellidoPersona" placeholder="Apellido">

<button (click)="onAgregarPersona()">Agregar</button>

</form>

formulario.component.ts

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

import { persona } from '../persona.model';

@Component({

selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

@Output()personaCreada = new EventEmitter<persona>();

nombrePersona:string;

apellidoPersona:string;

constructor() { }

ngOnInit() {

}

onAgregarPersona(){

this.personaCreada.emit(new persona(this.nombrePersona,this.apellidoPersona))

}

}

Se traslada del componente app.component.ts las variables string que se utilizan como ngModel en los controles input. Creamos un decorador de salida para poder mandar el objeto persona creado del componente hijo al padre y lo emitimos en el método onAgregarPersona(no olvides importar el componente EventEmitter de angular/core)

app.component.html

<app-formulario (personaCreada)="onPersonaAgregada($event)"></app-formulario>

<div >

<app-persona \*ngFor="let personaArray of personas; let i = index"

[persona]="personaArray"

[index]="i"

></app-persona>

</div>

Se agrega en el componente principal el control del formulario y declaramos la función que se estará monorizando enlazando la con otro método que se creara en el ts del componente principal

app.component.ts

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

import {persona} from './persona.model';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

titulo = 'Lista de personas';

personas:persona[] = [new persona("Veronica","Hernandez"), new persona("Gisela", "Perez")];

onPersonaAgregada(event:persona){

this.personas.push(event);

}

}

**Referencias locales angular:**

Las referencias locales son muy similares al two way binding

Primero ponemos una referencia al control empezando por el signo de # y lo mandamos por parámetros a la funcion

formulario.component.html

<form>

<input type="text" name="nombre" #nombrePresona placeholder="Nombre">

<input type="text" name="apellido" [(ngModel)]="apellidoPersona" placeholder="Apellido">

<button (click)="onAgregarPersona(nombrePersona)">Agregar</button>

</form>

formulario.component.ts

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

import { persona } from '../persona.model';

@Component({

selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

@Output()personaCreada = new EventEmitter<persona>();

nombrePersona:string;

apellidoPersona:string;

constructor() { }

ngOnInit() {

}

onAgregarPersona(nombrePersona:HTMLInputElement){

this.personaCreada.emit(new persona(nombrePersona.value,this.apellidoPersona))

}

}

En los parametros de la función agregamos el nombre del parámetro y el tipo de parámetro y reemplazamos donde usábamos la variable nombrePersona por el parámetro nombrePersona y como manda el control completo accedemos a el atributo value

**Servicios:**

Los servicios son como una especie de modulo global donde se comunicaran los componentes atravez de el. Para crear uno primero tenemos que crear un nuevo componente LoggingService.service.ts

loggingService.service.ts

export class LoggingService{

imprimirMensaje(mensaje:string){

console.log(mensaje);

}

}

formulario.component.ts

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

import { persona } from '../persona.model';

import { LoggingService } from '../LoggingService.service';

@Component({

selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

@Output()personaCreada = new EventEmitter<persona>();

/\* nombrePersona:string;

apellidoPersona:string;\*/

constructor(private LoggingService:LoggingService) { }

ngOnInit() {

}

onAgregarPersona(nombrePresona:HTMLInputElement, apellidoPersona:HTMLInputElement){

this.LoggingService.imprimirMensaje("Se agrego una persona");

this.personaCreada.emit(new persona(nombrePresona.value,apellidoPersona.value))

}

}

Despues inyectamos el servicio en el contructor y lo utilizamos en la función on agregar persona

**Data Services:**

Para poder usar un data services hay que hacer varios cambios en el código de nuestro poryecto pero al final nos estará ahorrando código y será mucho mas simple

Primero creamos un nuevo archivo de servicio:

PersonasService.service.ts

import { persona } from './persona.model';

export class PersonasService{

personas:persona[] = [new persona("Veronica","Hernandez"), new persona("Gisela", "Perez")];

onPersonaAgregada(event:persona){

this.personas.push(event);

}

}

En este nuevo archivo movemos la creación de los nombres y el evento de empujar una persona al array

app.component.ts

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

import { persona } from './persona.model';

import { PersonasService } from './PersonasService.service';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

titulo = 'Lista de personas';

personas:persona[]=[];

constructor(private PërsonasService:PersonasService){}

ngOnInit(): void {

this.personas = this.PërsonasService.personas;

}

}

En este archivo tenemos que inyectar el servicio PersonasService en el constructor y en el método ngOnInit agregamos el array con los datos en la variable personas que es la que se manda al componente app.component.html en el \*ngFor

formulario.component.ts

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

import { persona } from '../persona.model';

import { LoggingService } from '../LoggingService.service';

import { PersonasService } from '../PersonasService.service';

@Component({

selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

/\* nombrePersona:string;

apellidoPersona:string;\*/

constructor(private LoggingService:LoggingService,private PersonasService:PersonasService) { }

ngOnInit() {

}

onAgregarPersona(nombrePresona:HTMLInputElement, apellidoPersona:HTMLInputElement){

this.LoggingService.imprimirMensaje("Se agrego una persona");

this.PersonasService.onPersonaAgregada(new persona(nombrePresona.value,apellidoPersona.value))

}

}

En este archivo vinculamos el evento onAgregarPersona con onPersonaAgregada del servicio PersonasServices

**Leer servicios desde otros servicios:**

**PersonasService.service.ts**

import { persona } from './persona.model';

import { LoggingService } from './LoggingService.service';

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

@Injectable()

export class PersonasService{

personas:persona[] = [new persona("Veronica","Hernandez"), new persona("Gisela", "Perez")];

constructor (private LoggingService:LoggingService){}

onPersonaAgregada(event:persona){

this.LoggingService.imprimirMensaje("Persona agregada " + event.nombre);

this.personas.push(event);

}

}

Para hacer esto solo es necesario inyectar el servicio en el contructor, importar el servicio y agregar al inicio de la clase el decorador @Injectable()

**Comunicación entre componentes utilizando un servicio:**

Podemos comunicarnos entre componentes utilizando un servicio y asi nos ahorramos código

Primero agregamos un botón en el div del nombre para al darle click saber que índice es

persona.component.html

<div>{{index+1}}.- {{persona.nombre}} {{persona.apellido}} <button type="submit" (click)="onSaberIndice()">Indice</button></div>

persona.component.ts

Agregamos la función en el ts y la asociamos con un objeto de tipo EventEmitter

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

import { persona } from '../persona.model';

import { PersonasService } from '../PersonasService.service';

@Component({

selector: 'app-persona',

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

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

})

export class PersonaComponent{

@Input()persona:persona;

@Input()index:number;

constructor(private PersonasService:PersonasService){}

onSaberIndice(){

this.PersonasService.personaIndice.emit(this.index);

}

}

PersonasService.service.ts

Creamos el evento de tipo EventEmitter

import { persona } from './persona.model';

import { LoggingService } from './LoggingService.service';

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

@Injectable()

export class PersonasService{

personas:persona[] = [new persona("Veronica","Hernandez"), new persona("Briseida", "Perez")];

personaIndice = new EventEmitter<number>();

constructor (private LoggingService:LoggingService){}

onPersonaAgregada(event:persona){

this.LoggingService.imprimirMensaje("Persona agregada " + event.nombre);

this.personas.push(event);

}

}

formulario.component.ts

Nos suscribimos al evento en el constructor para poder acceder cada que se de click

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

import { persona } from '../persona.model';

import { LoggingService } from '../LoggingService.service';

import { PersonasService } from '../PersonasService.service';

@Component({

selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

/\* nombrePersona:string;

apellidoPersona:string;\*/

constructor(private LoggingService:LoggingService,private PersonasService:PersonasService) {

this.PersonasService.personaIndice.subscribe((indice:number) => alert("El indice es: " + indice));

}

ngOnInit() {

}

onAgregarPersona(nombrePresona:HTMLInputElement, apellidoPersona:HTMLInputElement){

this.LoggingService.imprimirMensaje("Se agrego una persona");

this.PersonasService.onPersonaAgregada(new persona(nombrePresona.value,apellidoPersona.value))

}

}

**Pipes:**

Los pipes sirven para dar formato a la salida del navegador por ejemplo formato de tipo moneda y se utiliza en la interpolación:

<h2 class="alert alert-success">Total: {{totalPresupuesto | currency:'MXN':'$':'1.2-2'}} </h2>

«»

**ngClass:**

podemos condicionar el uso de una clase según el valor de una variable booleana

<input [ngClass] =”{nombre-clase: varialbe ===’varialbe\_convalor\_booleano ’}”

**Generar un modulo con ng:**

Para generar un modulo con ng tecleamos el comando ng g module nombre-modulo

Despues de crearlo hay que agregar en los imports de

app.module.ts

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

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

import {FormsModule} from '@angular/forms'

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { PersonaComponent } from './persona/persona.component';

import { FormularioComponent } from './formulario/formulario.component';

import { LoggingService } from './LoggingService.service';

import { PersonasService } from './PersonasService.service';

@NgModule({

  declarations: [

    AppComponent,

    PersonaComponent,

    FormularioComponent

  ],

  imports: [

    BrowserModule, FormsModule, AppRoutingModule

  ],

  providers: [LoggingService,PersonasService],

  bootstrap: [AppComponent]

})

export class AppModule { }

Configurar el archivo de rutas (routing)

Vamos a crear una pagina con varias rutas para esto debemos de configurar el modulo routing que creamos

app-routing.module.ts

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

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

import { FormularioComponent } from './personas/formulario/formulario.component';

import { PersonasComponent } from './personas/personas.component';

const routes: Routes = [

  { path: '', component: PersonasComponent },

  { path: 'personas', component: PersonasComponent },

  { path: 'agregar', component: FormularioComponent },

  { path: 'personas/:id', component: FormularioComponent}

]

@NgModule({

  imports: [RouterModule.forRoot(

    routes

  )

  ]

})

export class AppRoutingModule { }

después configuramos el el app.module.ts para importar el routing

app.module.ts

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

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

import {FormsModule} from '@angular/forms'

import { AppRoutingModule } from './app-routing.module';

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

import { AppComponent } from './app.component';

import { PersonaComponent } from './personas/persona/persona.component';

import { FormularioComponent } from './personas/formulario/formulario.component';

import { LoggingService } from './LoggingService.service';

import { PersonasService } from './PersonasService.service';

import { PersonasComponent } from './personas/personas.component';

@NgModule({

  declarations: [

    AppComponent,

    PersonaComponent,

    FormularioComponent,

    PersonasComponent

  ],

  imports: [

    BrowserModule, FormsModule, AppRoutingModule,RouterModule

  ],

  providers: [LoggingService,PersonasService],

  bootstrap: [AppComponent]

})

export class AppModule { }

y en el app.component.html vamos a cambiar el componente que muestra por el routin que dara paso a los componente4s según la ruta

app.component.html

<router-outlet></router-outlet>

Este funciona gracias al RouterModule  que agregamos en el archvio app.module.ts

**Redireccionamiento con router:**

Primero debemos declarar en el constructor el route

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

import { persona } from '../persona.model';

import { PersonasService } from '../PersonasService.service';

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

@Component({

  selector: 'app-personas',

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

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

})

export class PersonasComponent implements OnInit {

  personas:persona[]=[];

  constructor(private PersonasService:PersonasService,

    private router:Router){}

  ngOnInit(): void {

   this.personas =  this.PersonasService.personas;

  }

  onAgregar(){

    this.router.navigate(['personas/agregar']);

  }

}

Y después lo usamos con la función navigate y le pasamos la dirección

**Utilizar los parámetros que se mandad por la url:**

Vamos a crear un ancla con los parámetros:

Personas.component.html

<a [routerLink]="['/personas',index]">elemento 1</a>

Ahora vamos a utilizar ese parámetro que se recibió

Formulario.component.ts

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

import { persona } from '../../persona.model';

import { PersonasService } from '../../PersonasService.service';

import { Router, ActivatedRoute } from '@angular/router';

@Component({

  selector: 'app-formulario',

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

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

})

export class FormularioComponent implements OnInit {

  nombrePersona:string;

  apellidoPersona:string;

  index:number;

  constructor(private PersonasService:PersonasService,

    private router: Router,

    private route: ActivatedRoute) {

    this.PersonasService.personaIndice.subscribe((indice:number) => alert("El indice es: " + indice));

  }

  ngOnInit() {

    this.index = this.route.snapshot.params['id'];

    }

Recordar que el nombre de id se asigna en el archivo de rauting y aunque se declare como number asigna un valor string hay que usar Number.isNaN

app-routing.module.ts

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

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

import { FormularioComponent } from './personas/formulario/formulario.component';

import { PersonasComponent } from './personas/personas.component';

const routes: Routes = [

  { path: '', component: PersonasComponent },

  { path: 'personas', component: PersonasComponent },

  { path: 'agregar', component: FormularioComponent },

  { path: 'personas/:id', component: FormularioComponent}

]

@NgModule({

  imports: [RouterModule.forRoot(

    routes

  )

  ]

})

export class AppRoutingModule { }

**query params:**

para pasar parámetros por la url debemos declararlos en el <a>

<a

[routerLink]="['/personas',index]"

[queryParams]="{modoEdicion: '1'}">Link</a>

Y para acceder a ellos

this.modoEdicion = +this.route.snapshot.queryParams['modoEdicion'];

aquí le asigna a la variable modoEdicion el valor del parámetro modoEdicion y al agregarle el símbolo + la convierte en number

**child route o rutas hijas:**

Para poder trabajar con rutas hijas debemos modificar el archivo app-routing.module.ts

app-routing.module.ts

const routes: Routes = [

  { path: '', component: PersonasComponent },

  { path: 'personas', component: PersonasComponent, children: [

    { path: 'agregar', component: FormularioComponent },

    { path: ':id', component: FormularioComponent},

  ]}

]

De este modo agregamos a las rutas como hijas y se visualizaran en la misma pagina pero en este caso hay que agregar en el personas.component.html el router-outlet

personas.component.html

<button (click)="onAgregar()">Agregar</button>

  <div >

    <app-persona \*ngFor="let personaArray of personas; let i = index"

    [persona]="personaArray"

    [index]="i"

    ></app-persona>

  </div>

  <router-outlet></router-outlet>

**Guardar los datos en una base de datos de firebise:**

Primero debemos crear la bases de datos de firebise en modo realtimedatabase y habilitar las funciones de escritura y lectura si sale el error 401 de no autorizado para escribir

Luego creamos un nuevo servicio para que se conecte con la base de datos

data.service.ts

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

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

import { persona } from './persona.model';

@Injectable()

export class DataServices{

    constructor(private HttpCliente: HttpClient){}

    guardarPersona(personas: persona[]){

        this.HttpCliente.put('https://listado-personas-ad186.firebaseio.com/datos.json', personas)

        .subscribe(

            response => console.log("Persona agregada " + response),

            error => console.log("Error al guardar: " + error)

        );

    }

}

La coneccion debe terminar siempre con .json

app.module.ts

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

@NgModule({

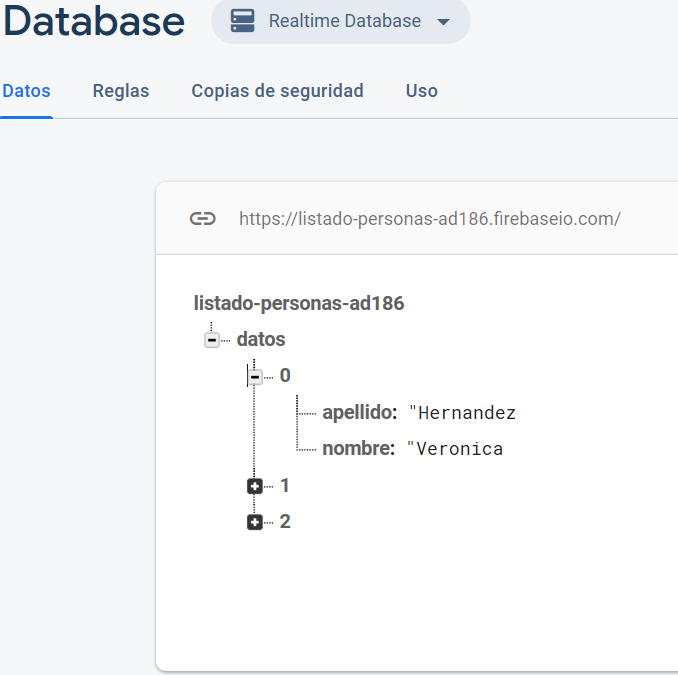
  imports: [

    BrowserModule, FormsModule, AppRoutingModule,RouterModule,HttpClientModule

export class AppModule { }

En este archivo agregamos el httpclientmodule y lo agregamos a los imports

Base de datos:



**Actualizar datos de la base:**

Es muy sencillo solo agregamos en el archivo dataservice la función:

data.service.ts

editarPersona(index:number,persona: persona){

        let url: string = 'https://listado-personas-ad186.firebaseio.com/datos/' + index + '.json';

        this.HttpCliente.put(url,persona)

        .subscribe(

            response => console.log("Se modifico persona: " + response),

            error => console.log("Error al modificar persona: " + error)

        );

    }

Nota que concatenamos la url con el índice del dato que queremos actualizar

**Borrar un registro:**

Borar un registro es relativamente fácil, solo cambiamos elmetodo put por delete

eliminarPersona(index:number,personas:persona[]){

        let url: string = 'https://listado-personas-ad186.firebaseio.com/datos/' + index + '.json';

        this.HttpCliente.delete(url)

        .subscribe(

            response => {console.log("Se elimino persona: " + response);

        },

            error => console.log("Error al eliminar persona: " + error)

        );

    }

**Otra forma de trabajr con fomularios:**

Vamos a trabajar con ngModel pero esta vez asociamos en ngModel con la propiedad name y asi no tenemos que poner el nombre de una variable y aparte tenemos acceso a este valor al enviar el formulario por el evento login(f)

login.component.html

<form #f="ngForm" (ngSubmit)="f.form.valid && login(f)">

    <input name="email" type="email" ngModel required size="20" />

    <input name="password" type="password" ngModel required size="15" />

    <button type="submit">Login</button>

</form>

login.component.ts

login(form:NgForm){

    const email = form.value.email;

    const passwor = form.value.password;

  }

**Configurar firebase para pedir autenticación:**

Primro cambiamos las reglas para que pida autenticación:

Reglas

{

"rules": {

".read": "auth != null",

".write": "auth != null"

}

}

Y en autenticación cabíamos a correo y habilitamos

**Trabajar con la solicitud de autenticación:**

Primero creamos un nuevo servicio que pedirá la autenticación y nos regresara el numero de acceso:

login.service.ts

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

import \* as firebase from 'firebase';

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

@Injectable()

export class LoginService{

 token: string;

 constructor(private router:Router){}

 login(email:string,password:string){

    firebase.auth().signInWithEmailAndPassword(email,password).then(

        response => {

            firebase.auth().currentUser.getIdToken().then(

                token => {

                    this.token = token;

                    this.router.navigate(['/']);

                }

            )

        }

    )

 }

 getIdToken(){

     return this.token;

 }

}

data.service.ts

constructor(private HttpCliente: HttpClient, private LoginService: LoginService){}

    cargarPersonas(){

        return this.HttpCliente.get('https://listado-personas-ad186.firebaseio.com/datos.json?auth=' + this.LoginService.getIdToken());

    }

**Guardianes:**

Los guardianes sirve para evitar que un usuario ingrese a una dirección sin estar autenticado

Primero creamos un nuevo servicio:

login-guardian.ts

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';

import { LoginService } from './login.service';

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

@Injectable()

export class LoginGuardian implements CanActivate{

    constructor(private LoginService: LoginService, private router: Router){}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot){

      if(this.LoginService.isAutenticado()){

          return true;

      }else{

          this.router.navigate(['login']);

          return false;

      }

    }

}

Y modificamos el archivo de routing

app-routing.module.ts

const routes: Routes = [

  { path: '', component: PersonasComponent, canActivate:[LoginGuardian] },

  { path: 'personas', component: PersonasComponent, canActivate:[LoginGuardian] ,children: [

    { path: 'agregar', component: FormularioComponent },

    { path: ':id', component: FormularioComponent },

  ]},

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

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

]

**Compilar un proyecto para subirlo a un servidor:**

Solo escribimos en una terminal el comando ng build –prod –aot

**Usar el hosting de firebase para subir nuestra aplicación:**

Primero instalamos las herramientas de firebase con el comando:

npm install -g firebase-tools

después tecleamos los comandos:

firebase login

firebase init

firebase deploy

Para subir la pagina al host de firebase