	M6 – Desenvolup.Aplicacions Web + M08 – Despleg. Aplicacions	
	Práctica Conjunta M08 + M06	
Gómez Villoldo, Alejandro		
Práctica N°: --	Raspberry + APIRest (Springboot) + Angular	

Resumen de construcción de la práctica utilizando Raspberry (APIRest + MariaDB) con Front en Angular

Primero, creamos la APIRest tiene que tener un controlador como el que enseño a continuación con todas las opciones CRUD. En mi caso utilizo Java en Netbeans construido con Springboot.

```

UserController.java x application.properties x
[
    public List<User> listUsers() {
        return userService.listUsers();
    }

    @GetMapping("/user/{id}")
    public ResponseEntity<User> obtainUser(@PathVariable Integer id) {
        try {
            User product = userService.obtainUserId(id);
            return ResponseEntity.ok(body: product);
        } catch (Exception e) {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping("/user")
    public void newUser (@RequestBody User user) {
        userService.saveUser(user);
    }

    @PutMapping("/user/{id}")
    public ResponseEntity<?> editUser (@RequestBody User user, @PathVariable Integer id) {
        try {
            User userExists = userService.obtainUserId(id);
            userExists.setNombre(nombre: user.getNombre());
            userExists.setEmail(email: user.getEmail());
            userExists.setSubject(subject: user.getSubject());
            userExists.setDescription(description: user.getDescription());
            userExists.setNumorder(numorder: user.getNumorder());

            userService.saveUser(user: userExists);

            return new ResponseEntity<User>(body: user, status: HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<User>(status: HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/user/{id}")
    public void deleteProduct (@PathVariable Integer id) {
        userService.deleteUser(id);
    }

    @DeleteMapping("/user")
    public String message() {
        return ("Para eliminar un User tienes que introducir su {id}");
    }
}

```

También creamos el modelo o interface con los campos de nuestra clase “User”.

```

@Entity
@Data @NoArgsConstructor @AllArgsConstructor

public class User {
    @Id @GeneratedValue
    private Integer id;
    private String nombre;
    private String email;
    private String subject;
    private String description;
    private String numorder;
}

```

Esta APIRest deberá estar conectada con nuestra base de datos.

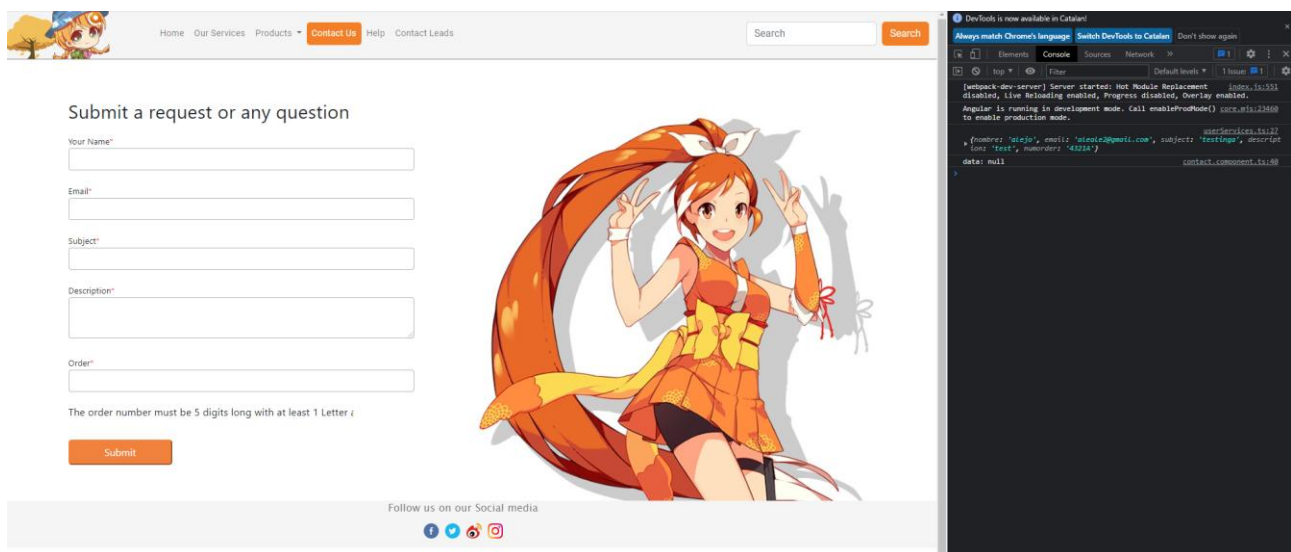
En MariaDB tenemos creada una DataBase con el nombre “usersdb2” y en ella tenemos una tabla llamada “user”. En la siguiente imagen podemos ver dichos elementos con valores insertados (mientras hacía pruebas con los campos).

```
MariaDB [usersdb2]> SELECT * FROM user;
+-----+-----+-----+-----+-----+-----+
| id | description | email | nombre | numorder | subject |
+-----+-----+-----+-----+-----+-----+
| 1 | prueba1 | prueba1@gmail.com | prueba1 | 1234A | prueba localhost |
| 2 | test | aleale2@gmail.com | alejo | 4321A | testinga |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0,001 sec)

MariaDB [usersdb2]> SELECT * FROM user;
+-----+-----+-----+-----+-----+-----+
| id | description | email | nombre | numorder | subject |
+-----+-----+-----+-----+-----+-----+
| 3 | testing2 | prueba3@gmail.com | prueba3 | 2134S | testinga2 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0,001 sec)

MariaDB [usersdb2]>
```

Desde el nuestra página de contacto de nuestra web montada con Angular deberemos hacer el submit, llamar a la API desde la IP ubicada en nuestra Raspberry.




Se pasaran los datos tal y como se muestra.

```
[webpack-dev-server] Server started: Hot Module Replacement index.js:551
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.

Angular is running in development mode. Call enableProdMode() core.mjs:23460
to enable production mode.

userServices.ts:27
{nombre: 'alejo', email: 'aleale2@gmail.com', subject: 'testinga', descript
ion: 'test', numorder: '4321A'}
data: null contact.component.ts:40
```

Una vez pasamos la información con el form de Contacto y se almacena en nuestra database deberemos poder mostrarlos para gestionar los servicios para usuarios Admin.

Home	Our Services	Products ▾	Contact Us	Help	Contact Leads	Search
Form Information Leads						
Name	Email	Order	Subject	Description	ID	Action
prueba3	prueba3@gmail.com	testinga2	testing2	2134S	3	

Este sería el TS con las funciones para mostrar los usuarios recuperándolos desde la Database.

```
import { Component } from '@angular/core';
import { userService } from '../servicios/userServices';
import { TemplateRef } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';

@Component({
  selector: 'app-form-view',
  templateUrl: './form-view.component.html',
  styleUrls: ['./form-view.component.css']
})

export class FormViewComponent {
  user: User[] = [];
  userId: any = "";
  userNombre: any = "";
  userEmail: any = "";
  userSubject: any = "";
  userDescription: any = "";
  userNumorder: any = "";

  constructor(private usersService: userService) { }

  ngOnInit(): void {
    /*this.userService.sendInformation(this.userNombre, this.userEmail, this.userSubject, this.userDescription, this
    | this.user = user;
    | }); */
    this.userService.getUsers().subscribe((user: User[]) => { this.user = user; this.userService.getUsers() });
  }

  applyFilter(event: Event): void {
    const filterValue = (event.target as HTMLInputElement).value.trim().toLowerCase();

    if (filterValue != "") {
      this.user = this.user.filter(user => user.nombre.toLowerCase().includes(filterValue));
    } else {
      this.userService.getUsers().subscribe((user: User[]) => { this.user = user; this.userService.getUsers() });
    }
  }

  deleteUser(id: BigInteger) {
    this.userService.deleteUser(id).subscribe(id => console.log("USER " + id + " DELETED CORRECTLY"));
    this.userService.getUsers().subscribe((user: User[]) => { this.user = user; this.userService.getUsers() });
  }
}

export interface User {
  id: BigInteger;
  nombre: string;
  email: string;
  subject: string;
  description: string;
  numorder: string;
}
```

El anterior TS se comunica con otro TS que corresponde y gestiona los servicios de conexión de Angular. A través de una variable con la IP de la Raspberry paso los métodos CRUD.

```
TS userServices.ts X
AngularFormularios > src > app > servicios > TS userServices.ts > User > numorder
1  import { Injectable } from "@angular/core";
2  import { BehaviorSubject, of } from "rxjs";
3  import { Observable } from 'rxjs/internal/Observable';
4  import { HttpClient } from "@angular/common/http";
5
6
7  @Injectable()
8  export class userService {
9      constructor(private http: HttpClient) { }
10     users: User[] = [];
11     //private urlService = 'http://172.16.22.99:8080/user';
12     private urlService = 'http://192.168.1.91:8080/user';
13
14     /*public sendInformation(nombre:string, email:string, subject:string,
15         if(nombre != "" && email != "" && subject != "" && order != ""){
16             this.users.push({nombre:nombre, email:email, subject:subject,
17         }
18         return of(this.users);
19     }
20     */
21
22     getUsers(): Observable<User[]> {
23         return this.http.get<User[]>(this.urlService);
24     }
25
26     addUser(user: User): Observable<User> {
27         console.log(user);
28         return this.http.post<User>(this.urlService, user);
29     }
30
31     updateUser(id: string, user: User): Observable<User> {
32         return this.http.put<User>(`${this.urlService}/${id}`, user);
33     }
34
35     deleteUser(id: string): Observable<any> {
36         return this.http.delete(`${this.urlService}/${id}`);
37     }
38 }
39
40 export interface User {
41     nombre: string;
42     email: string;
43     subject: string;
44     description: string;
45     numorder: string;
46 }
```