PROGRAMACIÓN ANGULAR

Reutilización de Componentes

Como podemos observar en el código desarrollado hasta este momento tanto el componente **buscador** como el componente **platos** tienen el mismo código html:

```
<div class="card-deck">
    <div class="card" *ngFor="let platoAux of platosBusqueda; let i = index" st
yle="max-width: 25%; padding: 10px;">
      <img class="card-img-
top" src="assets/img/{{platoAux.imagenPath}}" alt="Card image cap" style="max-
height: 150px;">
      <div class="card-body">
        <h5 class="card-title">{{platoAux.nombre}}</h5>
        <p class="card-text">$ {{platoAux.precio}}</p>
      </div>
      <div class="card-footer" style="text-align: center;">
        <button (click)="verPlato(platoAux.id)" class="btn btn-
primary" style="width: 90%;">Detalle</button>
      </div>
    </div>
  </div>
```

Por lo tanto lo que haremos será crear un nuevo componente para que podamos evitar esta redundancia de código.

Creamos el nuevo componente

ng g c components/itemPlato

Modifico el componente platos eliminado el código del card:

```
<h1>Nuestros Platos</h1>

<div class="card-deck">
</div>
```

El código eliminado lo asigno en el nuevo componente itemPlato, eliminando únicamente el *ngFor.

```
    <div class="card" style="max-width: 25%; padding: 10px;">
```

```
      <img class="card-img-
top" src="assets/img/{{platoAux.imagenPath}}" alt="Card image cap" style="max-
height: 150px;">
      <div class="card-body">
        <h5 class="card-title">{{platoAux.nombre}}</h5>
        <p class="card-text">$ {{platoAux.precio}}</p>
      </div>
      <div class="card-footer" style="text-align: center;">
        <button (click)="verPlato(platoAux.id)" class="btn btn-
primary" style="width: 90%;">Detalle</button>
      </div>
    </div>
```

Además de esto vamos a crear una nueva carpeta dentro de src, carpeta entidades y crearemos dentro de esta la clase TypeScript Plato:

```
export class Plato {
    id: string;
    nombre: string;
    precio: string;
    rubro: string;
    imagenPath: string;
    ingredientes: string[];
}
```

Modifico ítem-plato.component.ts agregando una variable

```
platoAux:Plato;
```

```
import { Component, OnInit } from '@angular/core';
import { Plato } from 'src/app/entidades/Plato';

@Component({
  selector: 'app-item-plato',
  templateUrl: './item-plato.component.html',
  styleUrls: ['./item-plato.component.css']
})
export class ItemPlatoComponent implements OnInit {

  platoAux:Plato;
```

```
  constructor() { }

  ngOnInit(): void {
  }

}
```

Reutilizo el nuevo componente en platos.component.ts

```
<h1>Nuestros Platos</h1>

<div class="card-deck">
  <app-item-plato *ngFor="let platoAux of platosArr; let i = index"></app-item-
plato>
</div>
```

Ahora nuestro objetivo es poder pasar un parámetro desde nuestro componente padre a un componente hijo (app-item-plato) para lograr esto usaremos los @Input los cuales indican que una propiedad será recepcionada desde fuera del componente actual.

@Input

```
@Input() platoAux:Plato;
```

Modificamos la definición de la variable agregando el @Input, con lo cual le indicamos a angular que el valor de la variable puede venir desde afuera. Por lo tanto ahora podemos hacer lo siguiente.

```
<div class="card-deck">
  <app-item-
plato *ngFor="let platoLocal of platosArr; let i = index" [platoAux]="platoLoca
l"></app-item-plato>
</div>
```

Siendo [platoAux] el nombre de la variable declarada en el componente **app-item-plato** y platoLocal la variable declarada en el *ngFor.

Si presionamos en el Botón “Ver Mas…” de la tarjeta veremos que no funciona y emite error, debemos hacer algunas modificaciones para subsanar el error:

Crear el Método verPlato() en el componente ItemPlato y agregar un nuevo Input para manejar el id de cada plato:

```
@Component({
  selector: 'app-item-plato',
  templateUrl: './item-plato.component.html',
  styleUrls: ['./item-plato.component.css']
})
export class ItemPlatoComponent implements OnInit {

  @Input() platoAux:Plato;
  @Input() index:number;

  constructor() { }

  ngOnInit(): void {
  }

  public verPlato(){
    console.log(this.index);
  }

}
```

Modificamos el evento click de nuestro html

```
(click)="verPlato()"
```

Agregamos la definición de nuestro nuevo input **index** en el componente

```
<app-item-
plato *ngFor="let platoLocal of platosArr; let i = index" [platoAux]="platoLoca
l" [index]="platoLocal.id">
  </app-item-plato>
```

Finalmente agrego el ruteo en mi componente ítem plato

```
  constructor(private router:Router) { }

  public verPlato(){
    console.log(this.index);
    this.router.navigate(['/detallePlato', this.index])
```

```
  }
```

En este momento hemos logrado que todo funcione, pero haciendo uso de ítem plato como componente.

Componente Buscador

Los mismos pasos que hicimos para el componente plato debemos aplicarlo al componente Buscador.

```
<h1>Buscando: {{termino}}  </h1>

<div class="card-deck">
  <app-item-
plato *ngFor="let platoLocal of platosBusqueda; let i = index" [platoAux]="plat
oLocal" [index]="platoLocal.id">
  </app-item-plato>
</div>
```

Emitir un Evento del Hijo hacia el Padre @Output

Vamos a ejecutar el método verPlato que existe en el componente padre platos.component

```
 public verPlato(idx:string){
    console.log("ID PLATO " + idx);
    this.router.navigate(['/detallePlato', idx])
  }
```

Ejecutando una llamada desde el componente hijo itemPlato

Para lograr este objetivo haremos uso del decorador @Output

```
@Output() platoSeleccionado:EventEmitter<number>;
```

Nuestro componente itemPlato toma la siguiente forma

```
export class ItemPlatoComponent implements OnInit {

  @Input() platoAux:Plato;
  @Input() index:number;

  //entre <> tiene que asignarse el tipo de dato a emitir
  //puede ser un object, string, number, etc
```

```
  @Output() platoSeleccionado:EventEmitter<number>;//number es el index


  constructor(private router:Router) {
    this.platoSeleccionado = new EventEmitter();
  }

  ngOnInit(): void {
  }

  public verPlato(){
    console.log(this.index);
    //this.router.navigate(['/detallePlato', this.index])
    this.platoSeleccionado.emit(this.index);
  }


}
```

Como vemos ahora el método verPlato() emite el evento y envía el parámetro index

```
this.platoSeleccionado.emit(this.index);
```

Y nuestro platos component queda:

```
<h1>Nuestros Platos</h1>

<div class="card-deck">
  <app-item-plato (platoSeleccionado)="verPlato($event)"
  *ngFor="let platoLocal of platosArr; let i = index" [platoAux]="platoLocal"
  [index]="platoLocal.id">
  </app-item-plato>
</div>
```

Note que hemos agregado el evento

```
(platoSeleccionado)="verPlato($event)"
```

$event es una palabra reservada de angular que indica que el evento **platoSeleccionado** ejecuta un evento y emite un valor

Para el componente buscador.component repito lo realizado anteriormente

```
<h1>Buscando: {{termino}}  </h1>

<div class="card-deck">
  <app-item-plato (platoSeleccionado)="verPlato($event)"
  *ngFor="let platoLocal of platosBusqueda; let i = index" [platoAux]="platoLoc
al"
  [index]="platoLocal.id">
  </app-item-plato>
</div>
```

PIPES

Transforman el dato solo a nivel visual.

Variable Inicial	Pipe	Resultado Final	
nombre = “Carlos”;	{{nombre | uppercase}}	CARLOS	Todo en mayusculas
nombre = “Carlos”;	{{nombre | lowercase}}	carlos	Todo en minúsculas
nombre = “Carlos”; nombre = “Carlos”; array=[0,1,2,3,4]	{{nombre | slice:3}} {{nombre | slice:0:3}} {{ array | slice:1:3}}	Los Car 1,2,3	Modifica una cadena o array según la cantidad de caracteres indicados en value (slice:value)
numero=3,14459265 numero=3,14459265 numero=3,14459265	{{ numero | number}} {{ numero | number:’3.0-0’}} {{ numero | number:’1.0-2’}}	3,142	Formatea un numero asignando cantidad de enteros y de decimales
numero=0,234	{{ numero | percent}}	23.4%	Pasa a porcentaje
salario=1234.5	{{ salario | currency}}	U$D 1,234.50	Números a formato Moneda.

heroe={
nombre=”Batman”,
ciudad=”Gotica”,
poder=”millonario”
}
	{{ heroe | json}}	heroe={
		nombre=”Batman”,
		ciudad=”Gotica”,
		poder=”millonario”
		}
			Muestre el objeto en formato JSON

fecha = new Date()	{{fecha | date}} {{fecha | date:’medium’}} {{fecha | date:’dd-MM’}}	Jan 25, 2017 Jan 25 2017 17:33:33 25-01	

ANGULAR - HTTP

Introducción a las peticiones HTTP

Para hacer uso de las peticiones HTTP en angular debemos incluir el modulo correspondiente en nuestro app.module

```
import { HttpClientModule } from "@angular/common/http";
```

```
imports: [
    BrowserModule,
    HttpClientModule
  ],
```

Inyectamos nuestra clase **HttpClient** en el constructor y hacemos uso de la función **get** para obtener el **JSON** de la **URL** especificada.

Ejemplo:

```
  constructor(private http:HttpClient) {
    this.http.get('https://restcountries.eu/rest/v2/lang/es').subscribe((datosURL:any)  => {
      this.datos = datosURL;
      console.log(datosURL);
    });
  }
```

Aplicación Resto

Para convertir nuestra aplicación Angular que lee los datos de un archivo a una que obtenga la información del servidor realizando una conexión a base de datos haremos lo siguiente:

1. Primero agregamos los métodos en nuestro servicio que realizaran la petición de datos a nuestro servidor.

```
//lee todos los platos
  getPlatosFromDataBase(){
    return this.http.get("http://localhost:8081/WebAppServer/RestoServlet?actio
n=listar").pipe(
      map( platosData => platosData));
  }
```

```
  //busca un plato por el id
  getPlatoEnBaseDatosXId(idx:string){
    return this.http.get("http://localhost:8081/WebAppServer/RestoServlet?actio
n=buscar&idPlato=" + idx).pipe(
      map( platoEncontrado => platoEncontrado));
  }

  //busca los platos por un terminode busqueda
  getPlatosBusquedaFromDataBase(termino:string){
    return this.http.get("http://localhost:8081/WebAppServer/RestoServlet?actio
n=busqueda&termino=" + termino).pipe(
      map( platosSearch => platosSearch));
  }
```

2. Modificamos nuestros componentes para que hagan uso de los métodos anteriores

Componente Platos

```
ngOnInit(): void {
    this.servicioDelivery.getPlatosFromDataBase()
    .subscribe(dataPlatos => {
      for(let plato in dataPlatos){
        this.platosArr.push(dataPlatos[plato]);
      }
      this.loading = false;
    });
  }
```

Componente Plato Detalle

```
constructor(private activatedRoute:ActivatedRoute, private servicioDelibery:Del
iveryService) {

    this.activatedRoute.params.subscribe(params =>{
      this.servicioDelibery.getPlatoEnBaseDatosXId(params['id'])
      .subscribe(platoEncontrado => {
        this.plato = platoEncontrado as Plato;
      });
    })
  }
```

Componente Busqueda

```
ngOnInit(): void {
    this.activatedRoute.params.subscribe(params=>{
      this.termino = params['termino'];
      this.servicioDelivery.getPlatosBusquedaFromDataBase(this.termino)
      .subscribe(dataPlatos => {
        for(let plato in dataPlatos){
          this.platosBusqueda.push(dataPlatos[plato]);
        }
      });
    });
  }
```

FORMUARIOS ANGULAR

Para finalizar la aplicación creamos 2 nuevos componentes que vamos a utilizar:

- ng g c components/plato-lista
- ng g c components/plato-admin

Modificamos nuestro navBar para incluir una nueva opción del menú para el componente plato-lista:

```
      <li class="nav-item" routerLinkActive="active">
        <a class="nav-link" [routerLink]="['lista']">Lista Platos</a>
      </li>
```

Modificamos nuestro archivo de RUTAS para contemplar los 2 nuevos componente:

```
    { path: 'lista', component: PlatoListaComponent },
    { path: 'admin/:id', component: PlatoAdminComponent },
```

E incluimos estos componentes en nuestro app.module.ts

```
import { PlatoListaComponent } from './components/plato-lista/plato-
lista.component';
import { PlatoAdminComponent } from './components/plato-admin/plato-
admin.component';
```

```
declarations: [
    …..
    PlatoListaComponent,
    PlatoAdminComponent
  ],
```

Dentro de nuestra clase de servicio implementamos 3 metodos que nos permitan insertar datos de un plato, modificarlo y/o eliminarlo.

```
platoAdminUrl:string = "http://localhost:8081/WebAppServer/RestoServlet";
  newPlato( platoNuevo: Plato) {
    return this.http.post<Plato>( this.platoAdminUrl, null, {params: new HttpPa
rams().set("action", "insertar").set("id", "0")
    .set("nombre", platoNuevo.nombre).set("imagenPath", platoNuevo.imagenPath).
set("precio", platoNuevo.precio).set("rubro", platoNuevo.rubro)
    }).pipe(map( nuevoPlato => {
            console.log(nuevoPlato.nombre);
            return nuevoPlato;
          }));
  }



   updatePlato( platoUpdate: Plato) {
      return this.http.post<Plato>( this.platoAdminUrl, null, {params: new Http
Params().set("action", "actualizar").set("id", platoUpdate.id)
      .set("nombre", platoUpdate.nombre).set("imagenPath", platoUpdate.imagenPa
th).set("precio", platoUpdate.precio).set("rubro", platoUpdate.rubro)
      }).pipe(map( res => {
              console.log(res.nombre);
              return res;
            }));
    }

    deletePlato(idPlato: string){
      return this.http.post( this.platoAdminUrl, null, {params: new HttpParams(
).set("action", "eliminar").set("id", idPlato)})
            .pipe(
            map( res => {
              console.log(res);
              return res;
            }));
    }
```

Para poder trabajar con formularios y hacer uso de las propiedades y eventos de Angular existentes para tal fin debemos importar el módulo FormsModule en nuestro app.module.ts:

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

```
imports: [
      ……
```

```
      FormsModule
      ……
  ],
```

Vamos a modificar primero nuestro componente **platos-lista.component.ts**

Interfaz de la Grilla dePlatos

HTML

```
<h1 >Platos</h1>
<hr>
<div class="row">
  <div class="col-md-12 text-right">
  <!-- Al estar dentro de un router-
outlet hay que añadir un "/" para que salga fuera y acceda
       correctamente a la ruta que queremos ir
  -->
    <button [routerLink]="['/admin', 'nuevo']"
       type="button" class="btn btn-outline-
primary" type="button" name="button">
     Nuevo plato
    </button>
  </div>
</div>
<div class="row animated fadeIn">
  <div class="col-md-12">
    <div class="table-responsive">
      <table width="100%" class="table table-sm table-hover">
        <thead class="thead-dark">
          <tr>
            <th style="width: 10%;">#</th>
```

```
            <th style="width: 20%;">Nombre</th>
            <th style="width: 20%;">Precio</th>
            <th style="width: 20%;">Rubro</th>
            <th style="width: 40%;" class="text-right">Opciones</th>
          </tr>
        </thead>
        <tbody>
            <tr *ngFor="let plato of platos; let i = index">
              <th scope="row">{{ i + 1 }}</th>
              <td>{{ plato.nombre }}</td>
              <td>{{ plato.precio }}</td>
              <td>{{ plato.rubro }}</td>
              <td class="text-right w120">
                  <button [routerLink]="['/detallePlato', plato.id]" type="butt
on"
                  class="btn btn-outline-primary" type="button" name="button">
                  <i class="fas fa-edit"></i> Detalle
                  </button>  
                  <button [routerLink]="['/admin', plato.id]" type="button"
                    class="btn btn-outline-
success" type="button" name="button">
                    <i class="fas fa-edit"></i> Editar
                  </button>  
                  <button (click)="delete(plato.id)" type="button" class="btn b
tn-outline-danger" type="button" name="button">
                    <i class="fas fa-trash-alt"></i> Eliminar
                  </button>
              </td>
            </tr>
        </tbody>
      </table>
      <div *ngIf="(platos | keys).length ===0"
            class="alert alert-info" role="alert">
        <strong>No hay</strong> registros que mostrar
      </div>
    </div>
  </div>
</div>
```

La mayor parte del código ya debería ser conocido para ustedes, solo vamos a analizar los botones que ejecutan las funciones para actualizar o eliminar un plato.

```
<td class="text-right w120">
  <button [routerLink]="['/detallePlato', plato.id]" type="button"
                 class="btn btn-outline-primary" type="button" name="button">
   <i class="fas fa-edit"></i> Detalle
  </button>  
  <button [routerLink]="['/admin', plato.id]" type="button"
                   class="btn btn-outline-
success" type="button" name="button">
  <i class="fas fa-edit"></i> Editar
  </button>  
  <button (click)="delete(plato.id)" type="button" class="btn btn-outline-
danger" type="button" name="button">
  <i class="fas fa-trash-alt"></i> Eliminar
  </button>
</td>
```

En el botón editar vemos que hacemos uso de un **routerLink** para ejecutar la ruta "**'/admin', plato.id**" que nos vincula con el componente PlatoAdmin.
Para el caso del botón Eliminar ejecutamos la función **delete()** a la cual le pasamos como parámetro el ID del plato a eliminar.

Type Script

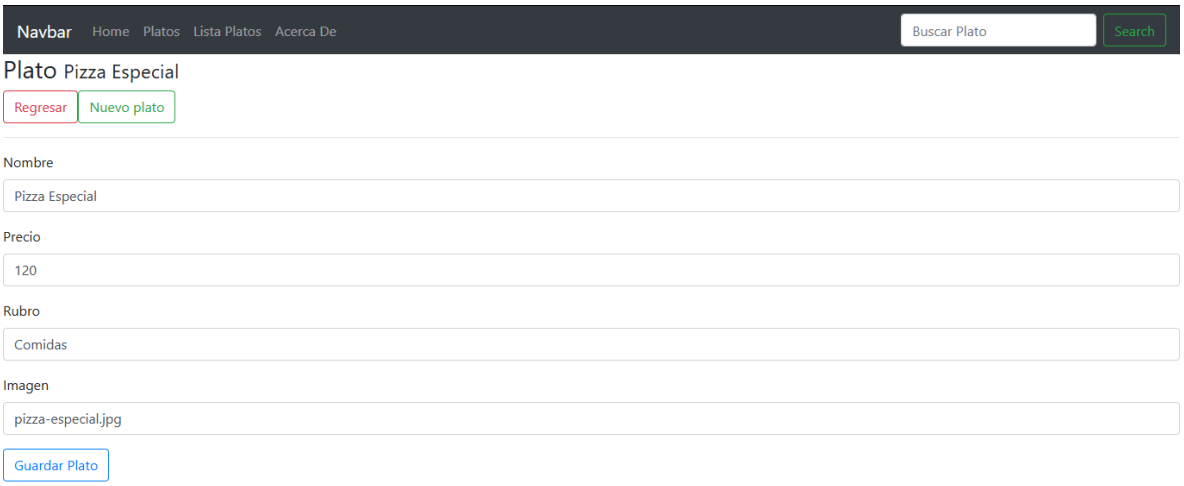
```
export class PlatoListaComponent implements OnInit {

  platos:Plato[] = [];
  loading = true;

  constructor(private servicioDelivery:DeliveryService, private router:Router,
private modalService: NgbModal) {

  }

  ngOnInit() {
    this.servicioDelivery.getPlatosFromDataBase()
    .subscribe(data => {
      console.log(data);
      for(let plato in data){
        console.log(data[plato]);
        this.platos.push(data[plato]);
      }
      this.loading = false;
```

```
    });
  }

  delete(idPlato:string){
    var opcion = confirm("Esta seguro que desea eliminar el plato?");
    if (opcion == true) {
      this.servicioDelivery.deletePlato(idPlato)
      .subscribe(data => {
        console.log(data);
        location.reload();
      });
    }

  }
}
```

Vemos como volvemos a usar el método `getPlatosFromDataBase` usado en otros componentes.

Formulario PlatoAdmin.

Interfaz del Formulario



HTML

```
<h3>Plato <small>{{ plato.nombre }}</small></h3>
<button class="btn btn-outline-
danger" [routerLink]="['/lista']">Regresar</button>
<button class="btn btn-outline-
success" (click)="addNew(forma)">Nuevo plato</button>
```

```
<hr>

<div class="row animated fadeIn fast">
  <div class="col-md-12">
    <form (ngSubmit)="save()" #forma="ngForm">
      <div class="form-group">
        <label for="nombre">Nombre</label>
        <input id="nombre" [(ngModel)]="plato.nombre" name="nombre" class="form
-control" required minlength="4" placeholder="Plato">
      </div>
      <div class="form-group">
          <label for="precio">Precio</label>
          <input id="precio" [(ngModel)]="plato.precio" name="precio" class="fo
rm-control" required minlength="4" placeholder="Precio">
      </div>
      <div class="form-group">
        <label for="rubro">Rubro</label>
        <input id="rubro" [(ngModel)]="plato.rubro" name="rubro" class="form-
control" required minlength="4" placeholder="Rubro">
      </div>
      <div class="form-group">
        <label for="imagenPath">Imagen</label>
        <input id="imagenPath" [(ngModel)]="plato.imagenPath" name="imagenPath"
 class="form-control" required minlength="4" placeholder="Imagen">
      </div>
      <div class="form-group">
        <button type="submit" class="btn btn-outline-
primary">Guardar Plato</button>
        <span style="text-align: right; width: 500; color: blue; font-
weight: bold">    {{ resultadoOperacion }}</span>
      </div>
    </form>
  </div>
</div>
```

El Evento `(ngSubmit)="save()"` ejecuta el submit del formulario llamando a la función save()
Propiedad `ngModel` vincula la etiqueta HTML del formulario con el atributo declarado en la clase TypeScript asociada, por ejemplo `[(ngModel)]="plato.nombre"`. La propiedad ngModel establece un vínculo bidireccional entre el valor del componente html y el atributo declarado en la clase Type Script, si modifico una de las partes se modifica automáticamente su contraparte.

Los símbolos **[]** y**()** que rodean a la propiedad indican en resumen:

- **[] get (obtiene el valor asignado en la variable)**
- **() set (asigna a la variable el valor cargado en la etiqueta)**

TypeScript

```
export class PlatoAdminComponent implements OnInit {


  plato:Plato = {
    id:"0",
    nombre:"",
    precio:"",
    rubro:"",
    imagenPath:"",
    ingredientes:[]
  };
  new = false;
  idplato: string;
  resultadoOperacion = "";

  constructor(private servicioDelivery:DeliveryService, private router:Router,
private activeRoute:ActivatedRoute) {
    this.activeRoute.params
    .subscribe(
      parametros => {
        this.idplato = parametros['id'];
        if(this.idplato != "nuevo"){
          servicioDelivery.getPlatoEnBaseDatosXId(this.idplato)
          .subscribe(platoEncontrado => this.plato = platoEncontrado as Plato);
        }else{
          console.log("ES NUEVO");
        }
      }
    );

  }

  ngOnInit() {
  }

  save() {
    if ( this.idplato === 'nuevo') {
```

```
      console.log('nuevo');
      this.servicioDelivery.newPlato(this.plato)
        .subscribe( data => {
          if(data && data.id){
            this.resultadoOperacion = "Operación finalizada con exito";
            this.router.navigate(['/admin', data.id]);
          }else{
            this.resultadoOperacion = "Error en la operación, verifique los dat
os";
          }
        },
        error => console.error(error));
    } else {
      console.log(`Update ${ this.idplato }`);
      this.servicioDelivery.updatePlato(this.plato)
        .subscribe( data => {
          if(data && data.id){
            this.resultadoOperacion = "Operación finalizada con exito";
            console.log(data);
          }else{
            this.resultadoOperacion = "Error en la operación, verifique los dat
os";
          }
        },
        error => console.error(error));
    }
  }


  addNew(formu: NgForm) {
    this.router.navigate(['/admin', 'nuevo']);
    formu.reset({
      id:"0",
      nombre:"",
      precio:"",
      rubro:"",
      imagenPath:""
    });
  }


}
```

Podemos ver que la función addNew ejecuta la ruta

```
this.router.navigate(['/admin', 'nuevo']);
```

Pasando como parámetro el valor nuevo esto es importante porque nos sirve para saber si en el método **save()** debemos ejecutar un INSERT o un UPDATE del plato cargado en el formulario. Como también este parámetro nos permite saber en el constructor del componente si debemos o no obtener de la base de datos el plato, dado que solo en el caso de querer Editar un plato deberemos hacerlo.

En nuestra clase de servicio analice los métodos **newPlato**, **updatePlato** y **deletePlato** y observe que se hace uso del método **post** para invocar al servidor y del objeto **HttpParams** para enviar la información del plato hacia el servidor por medio de la función **set()**.