Generalitat de Catalunya
Consorci per a la Formació Contínua de Catalunya

pime Coneixement

MINISTERIO DE TRABAJO, MIGRACIONES Y SEGURIDAD SOCIAL

SERVICIO PÚBLICO DE EMPLEO ESTATAL

Angular
IFCD65

Formador: Borja Mulleras Vinzia

Observables

Los OBSERVABLES sirven para trabajar con procesos asíncronos, es decir, con operaciones cuya respuesta no es inmediata.

Los procesos asíncronos no son solo peticiones http o consumo de API's

También podemos crear observables para estar pendientes de ciertas acciones de la aplicación y que se ejecuten procesos cuando pase algo. (eventos)

Trabajan con un flujo continuo de datos, es decir, pueden estar escuchando y esperando multiples respuestas.

Si falla, se puede reintentar la operación.

Se pueden encadenar operaciones adicionales a partir de una respuesta. (map, foreach, reduce, filter…)

Observables

PASO 1
ng new observables

PASO 2
Creamos un componente para examinar los observables
ng g c observables --skip-tests --inline-style

PASO 3
Borramos app.component.html y ponemos
<app-observables></app-observables>

Observables

PASO 4
Modificamos el contenido de observables.component.ts

```
import { Component } from '@angular/core';
import { Observable} from 'rxjs';
@Component({
  selector: 'app-observables',
  templateUrl: './observables.component.html',
  styles: [
  ]
})
export class ObservablesComponent {
  private obs:Observable<string>;
  constructor() {
    this.obs = new Observable;
  }
}
```

<string> señala el tipo del observable que estamos esperando

Observables

Los observables tienen 2 partes:

DEFINICIÓN
Se instancia el observable y se define su comportamiento.
¿Qué es lo que quiero observar?

SUBSCRIPCIÓN
Se invoca el método subscribe() y se indica el comportamiento que se hará con ello
¿Qué quiero hacer con ello?

Observables

PASO 4Bis

Modificamos el constructor de la clase ObservablesComponent de observables.component.ts

```
constructor() {
    this.obs = new Observable(
        observer => {
          //Aqui definimos aquello que queremos observar. En este caso es una
          //funcion que se ejecuta cada segundo e incrementa un contador.
            let contador=0;
          //Creamos una función que se incrementa una unidad cada segundo
            let intervalo=setInterval( () => {
                //incrementar el contador en una unidad.
                contador++;
                //Indicamos en que momento cambia el contador. Permite que los
                 //elementos que esten observando reciban la información.
                 //Le pasamos como parametro el valor que queremos enviar
                observer.next(contador);
            },1000)
        }
    );
```

Observables

PASO 5

Añadimos la subscripción y lectura al constructor

```
//Aqui nos subscribimos al observable para detectar cuando cambia el contador.
  this.obs.subscribe(numero => console.log('Subs',numero));
 }
```

ANGULAR 15

Ver el contador por pantalla

Tenemos que crear una propiedad en el objeto ObservablesComponent

Inicializar en el constructor este valor

Y en subscriber, tenemos que asignar el contador a el numero.

Observables

Dentro del setInterval podemos hacer un clear de la función

```
if (contador==10) {
    clearInterval(intervalo);
//Cerramos el observable
observer.complete();
}
```

Los observables pueden tener 3 estados:
Next: se ejecuta el observable
Error: falla el observable
Complete: finaliza el observable, independiente que acabe bien o mal

Cambiamos esto el subscribe del observable

Observables

PASO 6

Modificamos observables.components.ts en el constructor, en la parte de subscripción

```
this.obs.subscribe({
                //Acciones a realizar si el observable termina correctamente
                  next: numero => {
                    console.log('Subs',numero);
                    this.segundos=numero;
                  },
                //Acciones a realizar si el observable falla
                  error: error => {
                    console.log('El observable falló');
                  },
                //Acciones a realizar cuando se completa el observable
                //tanto si hace lo que tiene que hacer como si falla
                  complete: () => {
                    console.log('El observable se completó');
                  }
                  });
  }
```

Observables

PASO 7
Comprobamos el error en observables.components.ts

```
….
if (contador==10) {
                clearInterval(intervalo);
                observer.error('Hay un error en el observable');
                observer.complete();
              }
….

//Acciones a realizar si el observable falla
                    error: error => {
                      console.log(error);
                    },
```

Observables

PASO 8

Podemos hacer que el observable se lance varias veces en caso de fallo.
Modificamos la llamada observables.components.ts

Añadir en el import:
import { Observable,retry} from 'rxjs';

Y la instrucción del observable es:
this.obs.pipe(retry(2)).subscribe({

IMPORTANTE: hay que quitar el obs.complete() sino no funciona el retry.

Servicios

Un servicio de Angular es un elemento del framework que encapsula cierta funcionalidad, requerida, normalmente, por más de un componente.

Comunicación con base de datos (CRUD)
Gestionar logs de la aplicación
Calcular operaciones complejas que se utilizan en varios componentes
Manejar el estado de la aplicación

Los servicios se usan para implementar el patrón Singleton (clase para cual solo se quiere instanciar un único objeto)

Un servicio puede depender de otros servicios.

En un servicio se puede marcar el ámbito del componente:
Global, Module o Component

Servicios

¿Cómo creamos un servicio?
ng g s servicios/mi-servicio

Una vez creado el servicio, en el módulo que queramos usarlo habrá que importarlo
Modificamos *.module.ts
Import {MiServicioService} from '.app/servicios/mi-servicio.service'

Y en el decorador, sección PROVIDERS hay que poner el servicio.

Servicios

Para probar servicios continuamos con el proyecto erp o generamos un nuevo

Queremos crear un listado de clientes en clientes

PASO 0
Debemos crear un interface de clientes.
ng g i modelos/cliente

Y creamos un interface con los siguientes campos:
Modelos/clientes.ts

```
export interface Cliente {
        id: number,
        nombre:string,
        direccion:string,
        obs?:string
}
```

Servicios

PASO 1
Creamos el servicio
`ng g s servicios/clientes`

Y ponemos:

```
export class ClientesService {
  private clientes:Cliente[];
 constructor() {
    this.clientes = [
        { id: 1, nombre:'Borja Mulleras',direccion:'Paseo Sant Joan Bosco',obs:'Factura a 30 dias'},
        { id: 2, nombre:'Jordi Diaz',direccion:'Plaza España s/n',obs:'Factura a 60 dias'},
        { id: 3, nombre:'Raul Martinez',direccion:'Cardenal Reig s/n', obs:'Factura a 90 dias'}
    ]
   }
}
```

Servicios

PASO 2
Debemos crear 2 métodos en el servicio:
listadoClientes()
detalleCliente(id:number)

```
//como definimos la propiedad privada, necesitamos este método.
 public listadoClientes():Cliente[] {
   return this.clientes;
 }
 //método que nos filtar a través del id el cliente correcto
 public detalleCliente(id:number):Cliente {
   return this.clientes.filter(cliente => cliente.id==id)[0];
 }
```

Servicios

PASO 3

Como este servicio lo tenemos que usar en el módulo, debemos editar el módulo correspondiente y añadirlo. En este caso es el app.module.ts en los providers

```
…
import { ClientesService } from './servicios/clientes.service';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [ClientesService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Servicios

PASO 4

Vamos a crear 3 componentes, uno de clientes y otros dos de lista de clientes y detalle de cliente

ng g c componentes/cliente

ng g c componentes/listaClientes

ng g c componentes/detalleCliente

PASO 5

Ahora configuramos el componente de clientes para que llame a estos dos componentes como hemos hecho en el ejemlo de modulos.

- Modificamos cliente.component.html
- app-routing.module.ts

Servicios

cliente.component.html

```
<h2>Módulo de clientes</h2>
<div class="links">
        <a routerLink="listaClientes" routerLinkActive="activo">
                <button>Lista de clientes</button>
        </a>
         <a routerLink="detalleCliente" routerLinkActive="activo">
                <button>Detalle de clientes</button>
        </a>
</div>
<router-outlet></router-outlet>
```

Servicios

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { ClienteComponent } from './componentes/cliente/cliente.component';
import { ListaClientesComponent } from './componentes/lista-clientes/lista-
clientes.component';
import { DetalleClienteComponent } from './componentes/detalle-cliente/detalle-
cliente.component';
const routes: Routes = [
    {path:'',component:ClienteComponent,children:[
        {path:'listaClientes',component:ListaClientesComponent},
        {path:'detalleCliente',component:DetalleClienteComponent},
    ]}
    ];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Servicios

PASO 6

Queremos que en listaClientes se muestre el listado de clientes.
Tenemos que editar lista-clientes.component.ts para dar visibilidad al servicio

```
import { Component } from '@angular/core';
import { Cliente } from 'src/app/modelos/cliente';
import { ClientesService } from 'src/app/servicios/clientes.service';

@Component({
  selector: 'app-lista-clientes',
  templateUrl: './lista-clientes.component.html',
  styleUrls: ['./lista-clientes.component.css']
})
export class ListaClientesComponent {
  public clientes:Cliente[];
  //clienteServicio es una inyección de dependencia.
  constructor(private clienteServicio:ClientesService) {
    this.clientes = clienteServicio.listadoClientes();
    console.log(this.clientes);
  }
}
```

Servicios

PASO 7

Hay que mostrar el listado en lista-clientes.component.html

```
<h1>Lista de clientes</h1>
<div *ngFor="let cliente of clientes">
        <span class="id">{{cliente.id}} </span>
        <span class="nombre">{{cliente.nombre}} </span>
</div>
```

Servicios

PASO 7

Hay que mostrar el listado en lista-clientes.component.css

```
div span {
        display: inline-block;
}
div span.id {
        width:  20px;
}
div span.nombre {
        width:  200px;
}
```

Servicios

PASO 8

Ahora querremos hacer que cuando piquemos en el nombre se nos cargue el detalle del cliente. Tenemos que poner el link

```
<h1>Lista de clientes</h1>
<div *ngFor="let cliente of clientes">
        <span class="id">{{cliente.id}} </span>
        <span class="nombre">
                <a [routerLink]="['../detalleCliente',cliente.id]" >{{cliente.nombre}}</a>
         </span>
</div>
```

Servicios

PASO 9

Hay que tocar el enrutado para que haga caso al id

Hay que cambiar el array de rutas en app-routing.module.ts

```
const routes: Routes = [
   {path:'',component:ClienteComponent,children:[
      {path:'listaClientes',component:ListaClientesComponent},
      /*{path:'detalleCliente',component:DetalleClienteComponent},*/
      {path:'detalleCliente/:id',component:DetalleClienteComponent},
   ]}
   ];
```

AHORA SE CARGA EL COMPONENTE DE DETALLE CLIENTE

Hay que conseguir que se carguen los datos del detalle, por lo tanto,

Hay que recuperar el id en el componente de detalleCliente y

cargar los datos del cliente

Servicios

PASO 10

Modificamos detalle-cliente.module.ts

```
import { Component } from '@angular/core';
import { Cliente } from 'src/app/modelos/cliente';
import { ClientesService } from 'src/app/servicios/clientes.service';
@Component({
  selector: 'app-detalle-cliente',
  templateUrl: './detalle-cliente.component.html',
  styleUrls: ['./detalle-cliente.component.css']
})
export class DetalleClienteComponent {
  public cliente:Cliente;
  constructor(private clienteServicio:ClientesService) {
    this.cliente = this.clienteServicio.detalleCliente(1);
    console.log(this.cliente)
  }
}
```

Servicios

PASO 10

Modificamos detalle-cliente.module.ts para añadir la captura del parámetro id

No hace falta hacerlo con un observable

```
export class DetalleClienteComponent {
  public cliente:Cliente;
  constructor(private clienteServicio:ClientesService,private rutaActiva:ActivatedRoute)
{
    let id = rutaActiva.snapshot.params['id'];
    this.cliente = clienteServicio.detalleCliente(id);
    console.log(this.cliente);
  }
}
```

Servicios

PASO 11

Modificamos detalle-cliente.component.html

```
<h2>Detalle del cliente {{cliente.id}}</h2>
Nombre: {{cliente.nombre}}<br>
Dirección: {{cliente.direccion}}<br>
Observaciones: {{cliente.obs}}<br>
<a routerLink="/listaClientes">Volver listado</a>
```

¿Qué pasa si el id no existe?

Servicios

Tenemos que modificar detalle-cliente.component.ts y detalle-cliente.component.html

```
export class DetalleClienteComponent {
  public cliente:Cliente;
  public existe:boolean;
  constructor(private clienteServicio:ClientesService,private rutaActiva:ActivatedRoute) {
    this.existe=false;
    let id = this.rutaActiva.snapshot.params['id'];
     if (this.cliente = this.clienteServicio.detalleCliente(id)) {
        this.existe=true;
        console.log(this.existe);
     }
  }
}
```

Servicios

Tenemos que modificar detalle-cliente.component.ts y detalle-cliente.component.html

```
<span *ngIf="existe==true; then datos else error"></span>

<ng-template #datos>
<h2>Detalle del cliente {{cliente.id}}</h2>
Nombre: {{cliente.nombre}}<br>
Dirección: {{cliente.direccion}}<br>
Observaciones: {{cliente.obs}}<br>
</ng-template>

<ng-template #error>
        <h3>No es correcto el ID</h3>
</ng-template>


<a routerLink="/listaClientes">Volver listado</a>
```

ANGULAR 16

Hacer los mismo para proveedores adaptando los datos de clientes a proveedores.

Servicios HTTPClient

La ventaja de utilizar HTTPCLIENT para realizar conexiones AJAX respecto a lo visto anteriormente es que te permite subscribir a Observables.

¿Qué es una API Rest?
Es una aplicación de backend que ofrece servicios a través de unos endpoints

REST este montado sobre HTTP extiendo las capacidades de este.

REST esta basado en una serie de acciones:

GET: /peliculas consultar un listado
GET: /películas/:id consultar una película en concreto
POST: /película crear una película
PUT: /película modificar una película
DELETE: /película borrar una película

Vamos a utilizar un servicio que ya esta creado:
JSONPlaceHolder

Servicios HTTPClient

PASO 1

Generamos el servicio

```
ng g s servicios/todo
```

Generamos un modelo interface

```
ng g i modelos/todo
```

todo.ts

```
export interface Todo {
        userId: number,
        //lo ponemos como no obligatorio pq en el
        //alta no lo informaremos
        id?: number,
        title: string,
        completed: boolean
}
```

Servicios HTTPClient

PASO 2
Generamos un componente todo
ng g c componentes/todo

PASO 3
Modificamos cliente.component.html

```
<h2>Módulo de clientes</h2>
<div class="links">
            <a routerLink="listaClientes" routerLinkActive="activo">
                        <button>Lista de clientes</button>
            </a>
             <a routerLink="detalleCliente" routerLinkActive="activo">
                        <button>Detalle de clientes</button>
            </a>
             <a routerLink="listaTareas" routerLinkActive="activo">
                        <button>Lista de tareas</button>
            </a>
</div>
<router-outlet></router-outlet>
```

Servicios HTTPClient

PASO 4

Modificamos app-routing.module.ts para añadirle la ruta nueva

```
import { TodoComponent } from './componentes/todo/todo.component';

const routes: Routes = [
    {path:'',component:ClienteComponent,children:[
        {path:'listaClientes',component:ListaClientesComponent},
        {path:'detalleCliente/:id',component:DetalleClienteComponent},
        {path:'listaTareas',component:TodoComponent},
        ]}
    ];
```

Servicios HTTPClient

PASO 5

Añadimos el servicio TodoService y HttpClientModule en app.module.ts

```
…
import { ClienteComponent } from './componentes/cliente/cliente.component';
import { TodoService } from './servicios/todo.service';
import { TodoComponent } from './componentes/todo/todo.component';
import { HttpClientModule } from '@angular/common/http';
…
@NgModule({
imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [ClientesService,TodoService],
…
```

Servicios HTTPClient

PASO 6

Modificamos todo.service.ts

```
import { Injectable } from '@angular/core';
import { Todo } from 'src/app/modelos/todo';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class TodoService {
  constructor(private http:HttpClient) { }
  //método para consultar todas las tareas
  public listaTareas():Observable<Todo[]> {
      //Recurso a utilizar
      let api = 'https://jsonplaceholder.typicode.com/todos/';
      //peticion asincrona al servicio
      return this.http.get<Todo[]>(api);
    }
}
```

Servicios HTTPClient

PASO 7

Modificamos todo.component.ts

```
import { Todo } from 'src/app/modelos/todo';
import { TodoService } from 'src/app/servicios/todo.service';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';
…
export class TodoComponent {
    constructor (private servicio:TodoService) {
      let tareas = this.servicio.listaTareas().subscribe({
        //función a ejecutar si la petición ha ido bien
        next: (todos) => console.log(todos),
        //función a ejecutar si hay un error
        error: (error) => console.log('Algo fue mal',error),
        //funcion a ejecutar si la tarea se completa
        complete: () => console.log('Tarea completada')
        })
    }
}
```

Servicios HTTPClient

PASO 7 bis

Volvemos a modificar todo.component.ts para poder devolver una variable y hacer data bind

```
export class TodoComponent {
    public listaTareas:any;
    constructor (private servicio:TodoService) {
        this.consultaTareas();
    }
    public consultaTareas() {
        let tareas = this.servicio.listaTareas().subscribe({
        //función a ejecutar si la petición ha ido bien
        next: (todos) => { console.log(todos);
                           this.listaTareas=todos;},
        //función a ejecutar si hay un error
        error: (error) => console.log('Algo fue mal',error),
        //funcion a ejecutar si la tarea se completa
        complete: () => console.log('Tarea completada')
        })
    }
}
```

Servicios HTTPClient

PASO 8

En todo.component.html

```
<h1>Lista de tareas</h1>
<div *ngFor="let tarea of listaTareas">
        <span class="id" >{{tarea.id}} </span>
        <span class="tarea">{{tarea.title}} </span>
</div>
```

Angular 17

Completar este ejemplo con la consulta del detalle de la tarea

Angular 18

Podéis realizar el mismo ejemplo utilizando cuanquiera de los recursos que hay en https://jsonplaceholder.typicode.com/

Generalitat de Catalunya
Consorci per a la Formació Contínua de Catalunya

pime Coneixement

MINISTERIO DE TRABAJO, MIGRACIONES Y SEGURIDAD SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL

Angular
IFCD65

Formador: Borja Mulleras Vinzia