

Laboratorio 3 — Comunicación y control en Angular 18.2 (zoneless-ready)

Introducción técnica

Inputs y Outputs: Nueva filosofía con Signals

Antes (Angular 2–17):

- `@Input()` y `@Output()` usaban `EventEmitter` y `ChangeDetection` por `zone.js`.
- Requerían detección manual con `ChangeDetectorRef` si se usaban fuera de zona.

Ahora (Angular 18 zoneless):

- Inputs y Outputs se integran con **Signals**.
- Los Inputs son reactivos automáticamente → se pueden usar dentro de `computed()` o `effect()`.
- Los Outputs siguen usando `EventEmitter`, pero ahora funcionan sin Zone gracias a los triggers reactivos.

Ejemplo base:

```
@Component({
  selector: 'app-child',
  standalone: true,
  template: `<button (click)="notifyParent()">Notify</button>`
})
export class ChildComponent {
  @Input({ required: true }) message!: string;
  @Output() send = new EventEmitter<string>();

  notifyParent() {
    this.send.emit(`Child says: ${this.message}`);
  }
}
```

Guards: Nuevos enfoques reactivos

Los **route guards** siguen usando la interfaz (**CanActivate**, **CanDeactivate**, etc.), pero desde Angular 16+ se pueden escribir como **funciones puras**.

Ejemplo moderno:

```
export const authGuard: CanActivateFn = (route, state) => {  
  const isLoggedIn = inject(AuthService).isLoggedIn();  
  return isLoggedIn ? true : inject(Router).createUrlTree(['/login']);  
};
```

Ventajas:

- Sin clases → más rápido, menos boilerplate.
- Compatible con **route providers zoneless**.
- Se puede combinar con **Signals o Effects**.

Interceptors: Encadenamiento reactivo con HttpClient y Signals

En Angular 18, los interceptores siguen el mismo patrón, pero pueden coordinarse con **effects** para manipular peticiones.

Ejemplo moderno:

```
@Injectable()  
export class AuthInterceptor implements HttpInterceptor {  
  auth = inject(AuthService);  
  
  intercept(req: HttpRequest<any>, next: HttpHandler) {  
    const token = this.auth.token();  
    const cloned = req.clone({  
      headers: { Authorization: `Bearer ${token}` }  
    });  
    return next.handle(cloned);  
  }  
}
```

HttpClient + Signals + Effects

La gestión de peticiones puede hacerse ahora sin [AsyncPipe](#), usando Signals:

```
@Injectable({ providedIn: 'root' })
export class UserService {
  private http = inject(HttpClient);

  users = signal<any[]>([]);
  loading = signal(false);

  loadUsers() {
    this.loading.set(true);
    this.http.get<any[]>('https://jsonplaceholder.typicode.com/users')
      .subscribe({
        next: res => this.users.set(res),
        complete: () => this.loading.set(false)
      });
  }
}
```

Proyecto Angular 18.2

Estructura

```
src/
├── app/
│   ├── app.routes.ts
│   ├── app.component.ts
│   ├── services/
│   │   ├── auth.service.ts
│   │   ├── user.service.ts
│   │   └── auth.interceptor.ts
│   ├── guards/
│   │   └── auth.guard.ts
│   ├── features/
│   │   ├── login.component.ts
│   │   └── users.component.ts
│   └── shared/
│       └── child.component.ts
```

app.routes.ts

```
import { Routes } from '@angular/router';
import { authGuard } from '../guards/auth.guard';

export const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'login', loadComponent: () => import('../features/login.component').then(m => m.LoginComponent) },
  { path: 'users', canActivate: [authGuard], loadComponent: () => import('../features/users.component').then(m => m.UsersComponent) }
];
```

auth.service.ts

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

```
@Injectable({ providedIn: 'root' })
export class AuthService {
  private _token = signal<string | null>(null);
  private _logged = signal(false);

  login(user: string, pass: string) {
    if (user === 'admin' && pass === 'admin') {
      this._token.set('fake-token-123');
      this._logged.set(true);
    }
  }

  logout() {
    this._token.set(null);
    this._logged.set(false);
  }

  token = this._token.asReadonly();
  isLoggedIn = this._logged.asReadonly();
}
```

auth.guard.ts

```
import { CanActivateFn, Router } from '@angular/router';
import { inject } from '@angular/core';
import { AuthService } from '../services/auth.service';

export const authGuard: CanActivateFn = () => {
  const auth = inject(AuthService);
  const router = inject(Router);
  return auth.isLoggedIn() ? true : router.createUrlTree(['/login']);
};
```

auth.interceptor.ts

```
import { Injectable, inject } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/http';
import { AuthService } from '../auth.service';
```

```
@Injectable()
```

```
export class AuthInterceptor implements HttpInterceptor {
  auth = inject(AuthService);
```

```
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const token = this.auth.token();
    const authReq = token
      ? req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })
      : req;
    return next.handle(authReq);
  }
}
```

login.component.ts

```
import { Component, inject } from '@angular/core';
import { Router } from '@angular/router';
import { FormsModule } from '@angular/forms';
import { AuthService } from '../services/auth.service';

@Component({
  standalone: true,
  selector: 'app-login',
  template: `
    <h2>Login</h2>
    <form (ngSubmit)="login()">
      <input [(ngModel)]="user" name="user" placeholder="User">
      <input [(ngModel)]="pass" name="pass" type="password" placeholder="Password">
      <button type="submit">Login</button>
    </form>
  `,
  imports: [FormsModule]
})
export class LoginComponent {
  private auth = inject(AuthService);
  private router = inject(Router);

  user = "";
  pass = "";

  login() {
    this.auth.login(this.user, this.pass);
    if (this.auth.isLoggedIn()) this.router.navigateByUrl('/users');
  }
}
```

users.component.ts

```
import { Component, inject, effect } from '@angular/core';
import { UserService } from '../services/user.service';
import { ChildComponent } from '../shared/child.component';

@Component({
  standalone: true,
  selector: 'app-users',
  template: `
    <h2>Users</h2>
    <button (click)="service.loadUsers()">Reload</button>

    @if (service.loading()) {
      <p>Loading users...</p>
    } @else {
      @for (user of service.users; track user.id) {
        <app-child [message]="user.name" (send)="onChildMsg($event)"></app-child>
      }
    }
  `,
  imports: [ChildComponent]
})
export class UsersComponent {
  service = inject(UserService);

  onChildMsg(msg: string) {
    console.log('Received from child:', msg);
  }
}
```


child.component.ts

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

```
@Component({
  standalone: true,
  selector: 'app-child',
  template: `<p (click)="notifyParent()"> 👤 {{ message }}</p>`
})
export class ChildComponent {
  @Input() message = "";
  @Output() send = new EventEmitter<string>();

  notifyParent() {
    this.send.emit(`Clicked on: ${this.message}`);
  }
}
```

user.service.ts

```
import { Injectable, signal } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({ providedIn: 'root' })
export class UserService {
  private http = inject(HttpClient);

  users = signal<any[]>([]);
  loading = signal(false);

  loadUsers() {
    this.loading.set(true);
    this.http.get<any[]>('https://jsonplaceholder.typicode.com/users')
      .subscribe({
        next: res => this.users.set(res),
        complete: () => this.loading.set(false)
      });
  }
}
```

app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  template: `<router-outlet></router-outlet>`
})
export class AppComponent {}
```

Resultados del laboratorio

- Comunicación padre-hijo reactiva (@Input + @Output + Signals)
- Autenticación protegida con Guards
- Interceptor de autenticación funcional
- Peticiones HTTP con HttpClient + Signals
- Control Flow nativo @if y @for
- Lazy loading de componentes y zoneless-ready

Glosario Técnico — Laboratorio 3

Angular 18.2

Comunicación, seguridad y control en Angular moderno (zoneless-ready)

@Input()

Definición: Decorador que marca una propiedad como entrada desde un componente padre.

Angular 18+: ahora puede reaccionar directamente mediante *Signals* sin necesitar `ChangeDetectorRef`.

Ejemplo:

```
@Input({ required: true }) message!: string;
```

Permite recibir datos desde el padre y se integra con el nuevo motor reactivo.

@Output() y EventEmitter

Definición: Decorador que expone un evento al componente padre.

Uso moderno:

Sigue utilizando `EventEmitter`, pero ahora los eventos pueden triggerearse sin depender de `zone.js`.

Ejemplo:

```
@Output() send = new EventEmitter<string>();
```

Comunicación padre-hijo

Concepto: Mecanismo de Angular para conectar componentes:

- El padre pasa datos con `@Input`.
 - El hijo emite eventos con `@Output`.
Ahora es **completamente reactivo** gracias a Signals y control flow (`@if`, `@for`).
-

Signals

Definición: Nueva API reactiva de Angular.

Uso: Crea variables que notifican automáticamente sus cambios.

Ejemplo:

```
const count = signal(0);  
count.set(count() + 1);
```

Sustituye a `BehaviorSubject` o `NgRx` para estado local y reduce dependencias externas.

`effect()`

Definición: Bloque reactivo que se ejecuta cuando cambian las señales que usa.

Ejemplo:

```
effect(() => console.log(userSignal()));
```

Angular 18+: se puede escribir dentro de `services` y `components`.

Si dentro se modifican señales, debe añadirse `{ allowSignalWrites: true }`.

`computed()`

Definición: Señal derivada de otras señales.

Ejemplo:

```
const fullName = computed(() => first() + ' ' + last());
```

Solo recalcula cuando cambian las señales dependientes.

HttpClient

Definición: Servicio HTTP de Angular para peticiones REST.

Angular 18+: funciona zoneless y se integra con Signals.

Ejemplo:

```
http.get<User[]>('/api/users').subscribe(data => users.set(data));
```

Interceptor HTTP

Definición: Clase o servicio que intercepta y transforma las peticiones/respuestas HTTP.

Ejemplo:

```
intercept(req, next) {  
  const clone = req.clone({ setHeaders: { Authorization: 'Bearer token' } });  
  return next.handle(clone);  
}
```

Uso: Añadir tokens, manejar errores, logging o cacheo.

Guard (Route Guard)

Definición: Función o clase que determina si una ruta puede activarse o no.

Ejemplo moderno:

```
export const authGuard: CanActivateFn = () => {  
  const auth = inject(AuthService);  
  return auth.isLoggedIn() ? true : inject(Router).createUrlTree(['/login']);  
};
```

Ventajas: versión funcional más ligera y rápida, ideal para Angular 18 zoneless.

inject()

Definición: Nueva forma de obtener dependencias sin usar constructores.

Ejemplo:

```
const router = inject(Router);  
const http = inject(HttpClient);
```

Reemplaza `constructor(private service: Service) {}` y facilita uso en funciones puras.

Control Flow: @if, @for, @switch

Definición: Nuevo sistema de estructuras de control nativas en plantillas Angular 18+.

- **@if:** Condicional reactivo.
- **@for:** Iteración reactiva y eficiente.

Ejemplo:

```
@if (loading()) { <p>Loading...</p> } @else { <p>Data ready</p> }  
@for (user of users; track user.id) { <li>{{ user.name }}</li> }
```

Más rápido que ***ngIf** y ***ngFor**, y completamente zoneless.

Lazy Loading

Definición: Carga diferida de módulos o componentes solo cuando se navega a ellos.

Ejemplo:

```
{ path: 'users', loadComponent: () => import('./users').then(m => m.UsersComponent) }
```

Genera *chunks* (**users.chunk.js**) para reducir el tamaño inicial del bundle.

Chunk

Definición: Fragmento independiente del bundle JS generado en la compilación.

Función: Angular los carga dinámicamente al visitar rutas lazy.

Beneficio: Mejora el rendimiento inicial (LCP, FCP).

Bundle

Definición: Conjunto de todos los ficheros JavaScript, CSS y assets compilados por Angular.

El *bundle inicial* se divide en *chunks* gracias al *code splitting*.

LCP (Largest Contentful Paint)

Definición: Métrica de rendimiento que mide el tiempo hasta que el contenido principal se muestra.

Relación: Lazy loading y signals mejoran el LCP reduciendo render innecesario.

Zone.js y Zoneless

Zone.js: librería que detecta automáticamente los cambios para actualizar la vista.

Zoneless (Angular 18): ahora se puede usar Angular **sin Zone.js**, delegando la reactividad a Signals.

```
bootstrapApplication(AppComponent, { zone: 'noop' });
```

Aporta mejor rendimiento y control fino del rendering.

Router

Definición: Mecanismo de Angular para navegación SPA.

Angular 18: soporta `loadComponent` (sin módulos), guards funcionales y route-scoped providers.

DI (Dependency Injection)

Definición: Patrón base de Angular que inyecta dependencias (servicios, interceptores, etc.) en componentes.

Angular 18: `inject()` simplifica su uso en funciones y servicios standalone.