CanDeactivate 23/25

RECURSOS APUNTES

Así como los Guards pueden permitir o denegar el acceso a una ruta, también pueden permitir (o no) el caso contrario, que un usuario salga de una página.

Cómo proteger la salida de una ruta

¿Tienes en una página un formulario muy largo, o un proceso que demora algunos minutos, y el usuario cierra el navegador o se redirecciona a otra ruta?

Puedes alertarle de que el progreso del formulario se perderá o que un proceso aún no ha finalizado utilizando los Guards del tipo **CanDeactivate**.

1. Creando el guard

Crea un Guard utilizando el comando `ng g g <nombre-guard>` y esta vez selecciona que implemente la interfaz **CanDeactivate**.

El aspecto del nuevo Guard será como el siguiente.

```
// modules/shared/guards/exit.guard.ts
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanDeactivate, RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
```

```
export class ExitGuard implements CanDeactivate<unknown> {

  canDeactivate(component: unknown, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
    return true;
  }

}
```

La función canDeactivate() devuelve un booleano (o una promesa u observable con un booleano) para permitir o no la salida del usuario de la página dependiendo la lógica que tu quieras.

2. Importando el guard

Importa el Guard en la ruta que necesites, esta vez en la propiedad **canDeactivate** de cada regla.

```
// modules/cms/cms-routing.module.ts
import { ExitGuard } from '../shared/guards/exit.guard';

const routes: Routes = [
  {
    path: '',
    component: LayoutComponent,
    children: [
      {
        path: '',
        redirectTo: 'admin',
        pathMatch: 'full'
      },
      {
        path: 'admin',
        component: AdminComponent,
```

```
        canDeactivate: [ ExitGuard ],
      },
    ]
  }
];
```

Lógica dinámica del Guard

En los Guards del tipo **CanDeactivate** puedes implementar la lógica que necesites para permitir o no la salida del usuario de una página. En este caso, la lógica está atada al Guard y no tiene relación con el estado del componente, dificultando la programación de esa lógica.

Puedes relacionar el Guard con el Componente, para que sea el mismo quién determine la lógica y valide si permitir o no la salida del usuario.

1. Interfaz personalizada

Crea una interfaz para tipar tu componente y obligarlo a implementar una función que contendrá la lógica necesaria.

```
// modules/shared/guards/exit.guard.ts
export interface OnExit {
  onExit: () => Observable<boolean> | Promise<boolean> | boolean;
}

@Injectable({
  providedIn: 'root'
})
export class ExitGuard implements CanDeactivate<unknown> {

  canDeactivate(component: OnExit, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

    return component.onExit ? component.onExit() : true;
```

```
  }
}
```

La interfaz **OnExit** tiene una función llamada `onExit()` que los componentes tendrán que implementar para permitir la salida. En el return del Guard, has el llamado a dicha función para validar o no la lógica del componente.

2. Implementando la interfaz

Implementa la interfaz en el componente cuya lógica dependerá de que el usuario pueda o no salir de la página.

```
// modules/cms/components/admin/admin.component.ts
import { Component } from '@angular/core';
import { OnExit } from '../../../shared/guards/exit.guard';

@Component({
  selector: 'app-admin',
  templateUrl: './admin.component.html',
  styleUrls: ['./admin.component.scss']
})
export class AdminComponent implements OnExit {

  onExit() {
    const exit = confirm('¿Seguro desea salir?');
    return exit;
  }
}
```

De esta manera, solo necesitará un Guard del tipo **CanDeactivate**, cada componente que lo necesite, implementará la interfaz y aplicará la lógica necesaria para su funcionamiento.
