**Guards in Angular**, a core feature of Angular Router used to **control navigation and protect routes**. Guards are interfaces that allow you to **intercept and handle route navigation** based on conditions—like authentication, unsaved changes, data preloading, etc.

**🔐 Angular Route Guards — In Depth**

**🧱 What Are Guards?**

Guards are **Angular services** that implement special interfaces provided by @angular/router. They let you decide **if navigation should be allowed or not** in different scenarios:

| **Guard Interface** | **Triggered When...** | **Use Case Example** |
| --- | --- | --- |
| CanActivate | Before accessing a route | Auth check before entering a route |
| CanActivateChild | Before accessing a child route | Auth check on child routes |
| CanDeactivate | Before leaving the current route/component | Warn user about unsaved changes |
| CanLoad | Before loading a lazy-loaded module | Block loading of large features |
| Resolve<T> | Before route activation (to fetch data) | Fetch product/user data in advance |

**✅ 1. CanActivate – Gatekeeper for a Route**

**✅ Purpose:**

Checks if a route **can be accessed**.

**✅ Example:**

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService, private router: Router) {}

canActivate(): boolean {

if (this.authService.isLoggedIn()) {

return true;

} else {

this.router.navigate(['/login']);

return false;

}

}

}

**✅ In Routes:**

ts

CopyEdit

{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }

**✅ 2. CanActivateChild – Gatekeeper for Child Routes**

**✅ Purpose:**

Blocks access to **nested routes** if unauthorized.

**✅ Example:**

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class ChildGuard implements CanActivateChild {

constructor(private authService: AuthService, private router: Router) {}

canActivateChild(): boolean {

return this.authService.isLoggedIn();

}

}

**✅ In Routes:**

{

path: 'admin',

component: AdminComponent,

canActivateChild: [ChildGuard],

children: [

{ path: 'users', component: UserListComponent },

{ path: 'settings', component: AdminSettingsComponent }

]

}

**✅ 3. CanDeactivate – Ask Before Leaving**

**✅ Purpose:**

Warn users before navigating away from a component (e.g., unsaved form).

**✅ Interface:**

ts

CopyEdit

export interface CanComponentDeactivate {

canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;

}

**✅ Guard:**

@Injectable({ providedIn: 'root' })

export class DeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

canDeactivate(component: CanComponentDeactivate): boolean {

return component.canDeactivate ? component.canDeactivate() : true;

}

}

**✅ Component:**

export class EditFormComponent implements CanComponentDeactivate {

canDeactivate(): boolean {

return confirm('You have unsaved changes. Are you sure you want to leave?');

}

}

**✅ In Routes:**

{ path: 'edit', component: EditFormComponent, canDeactivate: [DeactivateGuard] }

**✅ 4. CanLoad – Prevent Loading a Lazy Module**

**✅ Purpose:**

Blocks the **loading** (not just access) of a lazy-loaded module.

@Injectable({ providedIn: 'root' })

export class LoadGuard implements CanLoad {

constructor(private authService: AuthService, private router: Router) {}

canLoad(): boolean {

return this.authService.hasRole('admin');

}

}

**✅ In Lazy Route:**

{

path: 'admin',

loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule),

canLoad: [LoadGuard]

}

**✅ 5. Resolve<T> – Fetch Data Before Loading**

**✅ Purpose:**

Preload data **before** the route activates, so the component gets it immediately.

@Injectable({ providedIn: 'root' })

export class ProductResolver implements Resolve<Product> {

constructor(private productService: ProductService) {}

resolve(route: ActivatedRouteSnapshot): Observable<Product> {

return this.productService.getProduct(route.paramMap.get('id')!);

}

}

**✅ In Routes:**

{

path: 'product/:id',

component: ProductComponent,

resolve: { product: ProductResolver }

}

**✅ In Component:**

ngOnInit() {

this.route.data.subscribe(data => {

this.product = data['product'];

});

}

**💡 Summary: When to Use Each Guard**

| **Scenario** | **Use This Guard** |
| --- | --- |
| User must be logged in | CanActivate |
| Prevent users from leaving form | CanDeactivate |
| Protect child routes | CanActivateChild |
| Stop loading a large lazy module | CanLoad |
| Fetch data before showing component | Resolve<T> |

**📦 Best Practices**

* Keep guard logic **lightweight**.
* Use guards to enforce **security, roles, and data integrity**.
* Prefer Resolve<T> for loading critical data before route loads.
* Always return a boolean, Observable<boolean>, or Promise<boolean>.