GABRIELE ROMANATO

Angular: autenticazione delle route con le guard funzionali

Angular è un framework robusto per lo sviluppo di applicazioni web, e una delle caratteristiche chiave è la gestione delle route. Implementare un sistema di autenticazione efficace è fondamentale per proteggere le risorse sensibili delle nostre applicazioni. Le guardie funzionali sono uno strumento potente in Angular per raggiungere questo obiettivo, consentendo di controllare l'accesso alle route in modo dinamico.

Cos'è una guardia funzionale?

Una guardia funzionale in Angular è una funzione che può essere utilizzata per controllare l'accesso a una route. Quando una route è protetta da una guardia funzionale, la funzione della guardia viene chiamata prima che la route venga attivata. Se la guardia restituisce `true`, la navigazione continua come previsto; altrimenti, se restituisce `false`, la navigazione viene bloccata.

Implementazione dell'autenticazione con guardie funzionali

Inizia creando un servizio di autenticazione che gestisca la logica di autenticazione. Questo servizio dovrebbe contenere metodi per il login, il logout e il controllo dello stato dell'autenticazione.

```
// auth.service.ts

import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
```

```
})
export class AuthService {
  private isAuthenticated: boolean = false;

  login(username: string, password: string): boolean {
    // Implementa la logica di autenticazione qui
    // Restituisci true se l'autenticazione ha successo,
altrimenti false
    // Ad esempio, potresti fare una chiamata a un backend
per verificare le credenziali
    this.isAuthenticated = true;
    return this.isAuthenticated;
  }

  logout(): void {
    this.isAuthenticated = false;
  }

  isAuthenticatedUser(): boolean {
    return this.isAuthenticated;
  }
}
```

Ora, crea una guardia funzionale che utilizzi il servizio di autenticazione per controllare l'accesso alle route.

```
// auth.guard.ts

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

export const authGuard: CanActivateFn = (route, state) => {
  const auth = inject(AuthService);
  if(!auth.isAuthenticatedUser()) {
    return false;
  }
```

```
  return true;
};
```

Per poter utilizzare il servizio di autenticazione all'interno della funzione, dobbiamo far ricorso alla funzione core `inject()` per effettuare la dependency injection al di fuori di una classe.

Ora che hai creato la guardia funzionale, puoi applicarla alle route che desideri proteggere.

```
// app-routing.module.ts

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { ProfileComponent } from
'./profile/profile.component';
import { LoginComponent } from './login/login.component';
import { authGuard } from './auth.guard';

const routes: Routes = [
  { path: 'profile', component: ProfileComponent,
canActivate: [authGuard] },
  { path: 'login', component: LoginComponent },
  // Altre route...
  { path: '', redirectTo: '/', pathMatch: 'full' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Con questi passaggi, hai implementato l'autenticazione sulle route in Angular utilizzando le guardie funzionali. Le route protette saranno

accessibili solo a utenti autenticati, e la navigazione verrà gestita in modo sicuro dal sistema di guardie funzionali.