Angular

Kontrola prava pristupa

Autori:
Milan Segedinac
Goran Savić

1. Kontrola prava pristupa u klijentskim aplikacijama

Kontrola prava pristupa omogućuje da korisnik aplikacije može da pristupi samo funkcionalnostima koje su mu dopuštene. Na serverskoj strani aplikacije kontrola prava pristupa realizuje se tako što se korisnicima dodele korisničke uloge, a izvršavanje REST servisa se dopusti samo određenim ulogama. Prilikom poziva svakog servisa proverava se koju ulogu ima korisnik i, ukoliko toj ulozi nije dozvoljeno da izvrši servis, odbije se izvršavanje na primer uz slanje odgovora sa statusom 403 – Forbidden. Ukoliko neautorizovani korisnik (korisnik koji se nije prijavio na sistem) pokuša da izvrši servis, poslao bi mu se odgovor 401 – Unauthorized.

Pored kontrole prava pristupa na serverskoj strani, gde se onemogućuje korisnicima da izvrše servise za koje nemaju, u veb aplikacijama se obzebeđuje i kontrola prava pristupa u klijentskoj aplikaciji, čime se postiše da se od korisnika sakrivaju nedozvoljene funkcionalnosti iz korisničkog interfejsa. To se može uraditi na dva načina:

1. Nedozvoljavanjem otvaranja rute stranice
2. Sakrivanjem elemenata korisničkog interfejsa, na primer dugmeta

Pri tome je od ključne važnosti obezbediti sinhronizovanost prava pristupa serverske i klijentske strane. Za tu namenu mi ćemo koristiti JSON Web Token (JWT).

U Angular aplikacijama kontrola prava pristupa se najčešće enkpsulira u zasebne objekte koje je potrebno koristiti u čitavoj aplikaciji, pa je zato zgodno omogućiti injekciju zavisnosti ovih objekata. Za tu namenu koriste se Angular *servisi*.

Servisi u Angular aplikacijama

Termin servis koristi se u različitim značenjima u računarstvu. U Angular aplikacijama, servis je objekat koji se može pribaviti injekcijom zavisnosti. Mi smo već koristili servise, na primer Http servis i videli smo da se injekcija zavisnosti ostvaruje tako što se parametar tipa Http samo navede u konstruktoru komponente u kojoj ćemo ga koristiti. Pored korišćenja gotovih servisa, u Angularu imamo mogućnost i da pravimo sami svoje servise. Servis se zadaje klasom, a injektovani objekat biće instanca klase. Pri tome moramo da uradimo nekoliko stvari:

1. Klasu servisa moramo da anotiramo dekoratorom `@Injectable()` kao što smo klase komponente anotirali dekoratorom `@Component`.
2. Kreiranu klasu servisa moramo registrovati u `AppModule`. Registrovanje se vrši tako što klasu servisa navedemo u listi providers u `@NgModule`, kao što smo klasu komponente naveli u listi `declarations`.

Ukoliko koristimo angular-cli za scaffolding aplikacije, novi servis ćemo inicijalno generisati sa `ng generate service <name-of-the-service>` i navedena dva koraka

će biti obavljena automatski. Tako kreirani servis moguće je injektovati u komponente, *ali i u druge servise*.

Česta namena servisa u Angular aplikacijama je enkapsulacija sloja za komunikaciju sa REST servisima. Servis za preuzimanje, postavljanje, izmenu i brisanje resursa `recrod` prikazan je listingom ispod.

```
@Injectable()

export class RecordService {

    private readonly path = '/api/records';

    constructor(private http: HttpClient) { }

    getRecords(): Observable<Record[]> {
        return this.http.get<Record[]>(this.path, {params})
            .catch((error: any) =>
                Observable.throw(error.message || 'Server error')
            );
    }

    getRecord(id: string): Observable<Record> {
        return this.http.get<Record>(`/api/records/${id}`)
            .catch((error: any) => Observable.throw(error.message
|| 'Server error'));
    }

    saveRecord(newRecord: Record): Observable<Record> {
        let headers = new HttpHeaders({ 'Content-Type':
'application/json' });
        return this.http.
            post(this.path, JSON.stringify(newRecord),
{ headers }).
            catch((error: any) => Observable.throw(error.message
|| 'Server error'));
    }

    deleteRecord(id: string): Observable<Record> {
        return this.http.delete(`${this.path}/${id}`).
            catch((error: any) =>
                Observable.throw(error.message || 'Server error')
            );
    }
}
```

Vidimo da `RecordService` ima postavljenu putanju REST servisa sa kojima će komunicirati (`'/api/records'`) i da implementira metode za preuzimanje, dodavanje i brisanje objekata tipa `Record`. U navedeni servis injektovan je servis `http` kroz koji će se komunikacija obavljati. Sve metode klase `RecordService` vraćaju obzervable za objekte tipa `Record`.

Kako bismo kreirani servis koristili? Prvo bismo morali da ga injektujemo, tako što bismo ga naveli u konstruktoru komponente u kojoj ćemo ga koristiti. Zatim bismo pozivali metode servisa kada nam treba komunikacija sa REST apijem. Primer korišćenja dat je listingom ispod.

```
@Component({

 …
})
export class MainComponent implements OnInit {
  public records: Record[];
  …
  constructor(private recordService: RecordService) {
  …
  }

  private loadData(){
    this.recordService.
      getRecords().
        subscribe((records: Record[]) => {this.records =
records;});
  }

}
```

Obratimo pažnju ta činjenicu da je metoda `recordService.getRecords` vratila obzervablu, što znači da je potrebno obraditi je kroz `subscribe` metodu.

Autnetifikacija u Angular aplikaciji

Autentifikacija u Angular aplikacijama se tipično enkapsulira u servis. Ovaj servis treba da obezbedi prijavu na sistem, odjavu sa sistema i preuzimanje podataka o ulogovanom korisniku. U situaciji u kojoj se koristi JWT za autentifikaciju, prijava na sistem obuhvata slanje zahteva sa korisničkim imenom i lozinkom i (ukoliko je uspešna) preuzimanje JWT tokena iz odgovora. U zaglavlju svakog sledećeg zahteva slaće se i JWT token, tako da ga je potrebno sačuvati prilikom prijave na sistem. Za čuvanje JWT tokena može se koristiti lokalno skladište pregledača (eng. *local storage*) - mehanizam čuvanja podataka u pregledaču u kom se podaci čuvaju kao parovi (ključ,string vrednost). Lokalno skladište je vezano za origin (kombinacija protokola, hosta i porta, na primer http://localhost:8080) i samo stranice koje imaju taj origin moćiće da pristupe zadatom lokalno skladištu. Lokalno

skladište je perzistentno, odnosno, podaci će ostati u lokalnom skladištu i nakon reseta pregledača.

Prijava na sistem obezbeđuje snimanje JWT tokena prijavljenog korisnika u lokalno skladište. Odjava sa sistema će se onda svesti na brisanje JWT tokena iz lokalnog skladišta. Implementacija servisa za autentifikaciju data je listingom ispod.

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


// import { Http, Headers, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import { HttpClient } from '@angular/common/http';
import { HttpHeaders } from '@angular/common/http';
import { JwtUtilsService } from 'app/security/jwt-utils.service';

@Injectable()
export class AuthenticationService {

  private readonly loginPath = '/api/login'

  constructor(private http: HttpClient, private jwtUtilsService:
JwtUtilsService) { }

  login(username: string, password: string): Observable<boolean> {
    var headers: HttpHeaders = new HttpHeaders({ 'Content-Type':
'application/json' });
    return this.http.post(this.loginPath,
JSON.stringify({ username, password }), { headers })
      .map((res: any) => {
        let token = res && res['token'];
        if (token) {
          localStorage.setItem('currentUser', JSON.stringify({
                                    username: username,

roles:this.jwtUtilsService.getRoles(token),
                                    token: token
                                  }));
          return true;
        }
        else {
          return false;
        }
      })
      .catch((error: any) => {
        if (error.status === 400) {
          return Observable.throw('Ilegal login');
        }
        else {
          return Observable.throw(error.json().error || 'Server
error');
        }
      });
  }

  getToken(): String {
```

Manipulacija JWT tokenima izmeštena je u zaseban servis, `JWTUtilsService`. Ovaj servis nam omogućuje da preuzmemo uloge korisnika i prikazan je listingom ispod.

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


@Injectable()
export class JwtUtilsService {

  constructor() { }

  getRoles(token:string){
    let jwtData = token.split('.')[1]
    let decodedJwtJsonData = window.atob(jwtData)
    let decodedJwtData = JSON.parse(decodedJwtJsonData)
    return decodedJwtData.roles.map(x => x.authority)||[];
  }
}
```

Preuzimanje uloga korisnika svodi se na uzimanje sekcije JWT tokena u kom se uloge nalaze, dekodiranje tog dela JWT tokena i vraćanje sadržaja dekodiranog dela tokena.

Korišćenje servisa za autentifikaciju

Komponenta LoginComponent namenjena je za prijavu na sistem. Templejt te komponente omogućuje unos korisničkog imena i lozinke i prikazan je listingom ispod.

```
<form class="form-signin" (ngSubmit)="login()">

  <h2 class="form-signin-heading">Please sign in</h2>
  <label for="username" class="sr-only">Username</label>
  <input type="text" id="username" class="form-control"
name="username" [(ngModel)]="user.username" placeholder="Username"
required autofocus>
  <label for="inputPassword" class="sr-only">Password</label>
  <input type="password" id="inputPassword" class="form-control"
name="username" [(ngModel)]="user.password" placeholder="Password"
required>
  <button class="btn btn-primary btn-block" type="submit">Sign
in</button>
</form>
<div *ngIf=wrongUsernameOrPass class="alert alert-warning box-msg"
role="alert">
  Wrong username or password.
</div>
```

Klasa ove komponente omogućuje prijavu na sistem. U njoj se injektuje AuthenticationService, kroz koji će se korisničko ime i lozinka poslati na server i prihvatiti token, kao što je opisano u prethodnoj sekciji. Kod klase dat je listingom ispod.

```
import { Component, OnInit, ViewEncapsulation } from '@angular/
core';


import { AuthenticationService } from '../security/
authentication.service'
import { Observable } from 'rxjs/Observable';

import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class LoginComponent implements OnInit {

  public user;

  public wrongUsernameOrPass:boolean;

  constructor(private authenticationService:AuthenticationService,
              private router: Router) {
    this.user = {};
    this.wrongUsernameOrPass = false;
   }

  ngOnInit() {
  }

  login():void{
    this.authenticationService.login(this.user.username,
this.user.password).subscribe(
      (loggedIn:boolean) => {
        if(loggedIn){
          this.router.navigate(['/main']);
        }
      }
    ,
    (err:Error) => {
      if(err.toString()==='Ilegal login'){
        this.wrongUsernameOrPass = true;
        console.log(err);
      }
      else{
        Observable.throw(err);
      }
    });
```

Osnovna metoda ove klase je `login()`. Ova metoda korisničko ime i lozinku unete u formu templejta komponente šalje na server koristeći `AuthenticationService`. U slučaju da je prijava na sistem uspela, pomoću routera se radi redirekcija na stranicu na komponentu na ruti 'main'. U slučaju neuspešne prijave, ostaje se na stranici za prijavu.

Guard i zabrana ruta

Kada su podaci o prijavljenom korisniku preuzeti sa servera i kada su smešteni u lokalno skladište, treba neprijavljenim korisnicima onemogućiti otvaranje svih ruta osim one namnjene prijavi na sistem. Za tu namenu u Angular aplikacijama se koriste gardovi (eng. *guards*). Gard je posebna vrsta servisa što znači da je klasa antorana dekoratorom `@Injectable()` odnosno da se pribavi injekcijom zavisnosti. Namena garda je da u određenim slučajevima zabrani otvaranje rute (u naše slučaju ukoliko ne postoje podaci o prijavljenom korisniku u lokalnom skladištu).

Postoje četiri vrste gardova definisanih sledećim interfejsima:

- `CanActivate` - Određuje da li ruta može da se aktivira
- `CanActivateChild` - Određuje da li potomačke rute mogu da se aktiviraju
- `CanDeactivate` - Određuje da li ruta može da se deaktivira
- `CanLoad` - Određuje da li modul može da se učita

Da bi se servis učinio gardom potrebno je da implementira odgovarajući interfejs. U našem slučaju želimo da zabranimo neprijavljenim korisnicima aktiviranje svih ruta osim rute komponente za prijavu na sistem, pa ćemo stoga koristiti `CanActivate` gard. U implementaciji ovog interfejsa metoda `canActivate` vratiće `true` ukoliko je korisnik prijavljen na sistem, a `false` ukoliko nije. Kod garda dat je listingom ispod.

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

import { CanActivate, ActivatedRouteSnapshot,
RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import { AuthenticationService } from 'app/security/
authentication.service';
import { Router } from '@angular/router';

@Injectable()
export class CanActivateAuthGuard implements CanActivate {

  constructor(private authenticationService:
AuthenticationService, private router: Router){}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> |
Promise<boolean> | boolean {
    if(this.authenticationService.isLoggedIn()){
      return true;
    }
    else{
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

Orbatimo pažnju na način na koji se u gardu proverava da li je korisnik prijavljen: za tu namenu koristimo `AuthenticationService`, koji je u gardu pribavljen injekcijom zavisnosti.

Kada je kreiran gard, potrebnoje postaviti ga na rute koje želimo da „štiti“. Prilikom definisanja ruta u `app.module` za svaku je moguće postaviti atribut `canActivate` sa gardovima. Kod postavljanja gardova dat je listingom ispod.

```
const appRoutes: Routes = [

  { path: 'record/:id', component: RecordDetailsComponent,
canActivate:[CanActivateAuthGuard] },
  { path: 'main', component: MainComponent, canActivate:
[CanActivateAuthGuard] },
  { path: 'login', component: LoginComponent},
  { path: '', redirectTo: 'main', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

Na listingu vidimo da je neprijavljenim korisnicima zabranjen pristup rutama `'main'` i `'record/:id'`.

Presretanje zahteva

Kada je obezbeđena prijava na sistem i kada su delovi klijentske aplikacije zaštićeni gardovima, potrebno je obezbediti da se iz klijentske aplikacije mogu pozivati REST servisi u skladu sa pravima korisnika. Da bismo to postigli uz svaki zahtev trebamo da pošaljemo i zaglavlje (`'X-Auth-Token'`) koje će sadržati JWT korisnika. To bismo mogli da uradimo postavljanjem zaglavlja pri svakom pojedinačnom zahtevu. Ovaj pristup je nezgodan jer moramo uvodi još jedan aspekt o kom treba razmišljati pri svakom HTTP zahtevu. Drugi pristup je da se napiše kod koji će presretati HTTP zahteve i u svaki umetati odgovarajuće zaglavlje. Za tu namenu koriste se interseptori (eng. *interceptor*). Servis `Http` ne dozvoljava korišćenje interseptora, ali servis `HttpClient` koji takođe služi za HTTP komunikaciju dozvoljava, pa ćemo ga zbog toga koristit u ovom primeru.

HTTP interseptor je Angular servis čija klasa implementira interfejs `HttpInterceptor` odnosno mora da ima metodu `intercept`. Ova metoda prima dva parametra: `HttpRequest` koji predstavlja presretnuti zahtev i `HttpHandler` koji omogućuje da se pristupa događajima HTTP zahteva. Implementacija HTTP interseptora za JWT autentifikaciju data je listingom ispod.

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

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent,
HttpHeaders } from '@angular/common/http';
import { AuthenticationService } from 'app/security/
authentication.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class TokenInterceptorService implements HttpInterceptor {

  constructor(private inj: Injector) { }

  intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    let authenticationService:AuthenticationService =
this.inj.get(AuthenticationService);
    request = request.clone({
      setHeaders: {
        'X-Auth-Token': `${authenticationService.getToken()}`
      }
    });

    return next.handle(request);
  }

}
```

Napravljena je kopija zahteva u koju je za zaglavlje ’X-Auth-Token’ postavljen token preuzet iz AuthenticationService. Zatim se pozivom next.handle(request) prelazi na dalje procesiranje zahteva, uz napomenu da zahtevi ima dodato navedeno zaglavlje.

Nakon što je kreiran interseptor treba ga uključiti u konfiguraciju aplikacije. Interseptor je dodat u providers listu u app.module, kao što je prikazano listingom ispod.

```
providers: [

  {
    provide: HTTP_INTERCEPTORS,
    useClass: TokenInterceptorService,
    multi: true
  },
  RecordService,
  AuthenticationService,
  CanActivateAuthGuard,
  JwtUtilsService
],
```