ANGULAR 9

Peter Gurský, peter.gursky@upjs.sk

Téma: komunikácia komponentov

- **Cieľ:** Vytvoríme si nezávislý komponent s formulárom, v ktorom budeme môcť editovať existujúcu alebo novú skupinu
- V src/modules/groups spustíme
  - ng g c group-edit-child
- Ak ho chceme vidieť v komponente GroupDetailComponent, vložíme do jeho šablóny tag, ktorý sa volá rovnako, ako selector v novom komponente a importujeme ho

```
<app-group-edit-child></app-group-edit-child>
```

group-edit-child pre pridávanie aj editáciu

- Potrebujeme, aby rodičovský komponent informoval o tom, koho editujeme
- V group-edit-child.component.ts si zadefinujeme, ktoré inštančné premenné chce mať na vstupe
  - Rodičovský komponent nevolá konštruktor, to robí framework
  - posielame premennú group (nová alebo editovaná),
- V rodičovi zadefinujeme hodnoty pre tieto premenné

Odoslanie vstupu od rodiča

group-edit-child-component.ts

```
import {Input, OnChanges} from '@angular/core';

export class GroupEditChildComponent implements OnChanges {
  @Input() public group: Group;
   …
   ngOnChanges() {
        //spracovanie group premennej
   }
}
```

rodič - šablóna:

```
<app-group-edit-child [group]="selectedGroup" ></app-group-edit-child>
```

inštančná premenná rodiča

Odoslanie skupiny rodičovi

group-edit-child-component.ts

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

export class GroupEditChildComponent ...{
  @Output() groupChange = new EventEmitter<Group>();
   …
   onSubmit() {
       this.groupChange.emit(this.group);
   }
}
```

rodič - komponent

```
onGroupSave(group:Group) {
    this.groups.push(group);
}
```

rodič - šablóna

```
<app-group-edit-child (groupChange)="onGroupSave($event)"></app-group-edit-child >
```

Kontrola routovania

- Niekedy nechceme dovoliť navigáciu
  - používateľ nie je prihlásený
  - chceme, aby sa používateľ najprv prihlásil
  - je potrebné dodať dáta
  - neboli uložené zmeny vo formulári
  - chceme sa používateľa spýtať či danú zmenu chce uložiť
- Strážca (guard) vráti buď
  - boolean, o tom či, či navigáciu povoliť, alebo nie, alebo
  - UrlTree – sparsovaná URL, kam sa chceme presunúť
  - Observable alebo Promise z dvoch vyššie spomenutých
- Navigácia pokračuje, až keď strážca hodnotu vráti

Typy strážcov

- Strážcovia sú funkcie
  - CanActivateFn – stráži navigáciu na konkrétne pravidlo
  - CanActivateChildFn – stráži rodiča aj deti podľa toho či sa deti môžu zobraziť
  - CanDeactivateFn – stráži odchod z aktuálnej url
  - ResolveFn – poskytuje dáta pred aktiváciou pravidla
  - CanMatchFn – ak strážca nepustí aktuálne pravidlo, nechá router skúšať ďalšie pravidlá

Vytvorenie strážcu

- ng generate guard guards/auth
  - vyberieme CanActivate
  - vytvorí súbor src/app/guards/auth.guard.ts

```
import { ActivatedRouteSnapshot, CanActivateFn, RouterStateSnapshot, UrlTree } from
'@angular/router';
import { Observable } from 'rxjs';


export const authGuard: CanActivateFn = (route : ActivatedRouteSnapshot, state:
RouterStateSnapshot)
  :Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree => {
  return true;
};
```

Vytvorenie strážcu

objekt aktivovaného (budúceho) pravidla routra, ktoré strážime. Máme prístup napr. ku
- spárovanému komponentu,
- vnoreným pravidlám aj rodičovskému pravidlu,
- častiam URL, ktorá bola použitá
- dátam/parametrom, ktoré cez URL prišli

state.url nám povie string-ovú reprezentáciu URL, ktorú práve router spracováva

/auth

guards/auth.guard.ts

```
import { ActivatedRouteSnapshot, CanA      teFn, RouterStateSnapshot, UrlTree    from
'@angular/router';
import { Observable } from 'rxjs';


export const authGuard: CanActivateFn = (route : ActivatedRouteSnapshot, state:
RouterStateSnapshot)
  :Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree => {
  return true;
};
```

Použitie strážcu

- sprážca stráži celý podstrom trás
  - do podtrás sa vnoríme, iba keď strážca pustí rodiča
  - ak chceme chrániť deti z rodiča použijeme CanActivateChildFn

```
const groupsRoutes: Routes = [
  {
    path: 'groups',
    component: GroupsComponent,
    canActivate: [authGuard],
    children: [
      ...
    ]
  }
];
```

Použitie strážcu

- strážca stráži celý podstrom pravidiel
  - do podpravidiel sa vnoríme, iba keď strážca pustí rodiča
  - ak chceme chrániť detské pravidlá z rodiča použijeme CanActivateChildFn
    - Vieme pristupovať k premenným URL určeným pre deti
    - Ak nepustí dieťa, nevytvorí ani rodiča
- poradie overovania:
  - najskôr sa overujú strážcovia CanDeactivateFn a CanActivateChildFn z najhlbšieho vnorenia pravidiel až po najvyššie,
  - potom sa overujú CanActivateFn strážcovia z najvyššieho pravidla po najnižšie
  - ak ktorýkoľvek strážca vráti false/UrlTree, ďalší strážcovia sa neoverujú

Presmerovanie na LoginComponent

```
export const authGuard: CanActivateFn = (route, state) => {
  const router = inject(Router);
  const usersService = inject(UsersService);
  if (usersService.isLoggedIn()) {
    return true;
  }
  usersService.redirectAfterLogin = state.url;
  router.navigateByUrl('/login');
  return false;
};
```

alternatívne:

```
const urlTree = router.parseUrl('/login');
return urlTree;
```

Strážca CanDeactivateFn

- používateľ vyplnil časť formulára a uklikol sa do menu
  - používateľa sa chceme spýtať, či chce zahodiť prácu
- používateľ poslal dáta na server, ale ešte nedošiel výsledok a už sa preklikáva inde
  - Čo ak sa uloženie nepodarilo? Používateľ o tom nebude vedieť. Môžeme počkať kým sa server vyjadrí a až potom presmerovať
- Vytvoríme si všeobecného strážcu pre komponenty, ktorý odovzdá rozhodnutie na metódu canDeactivate() vytvorenú v komponente

Strážca CanDeactivate

ng g guard guards/can-deactivate

CanDeactivateFn sa viaže na konkrétny komponent, resp. v našom prípade na komponenty implementujúce interface CanComponentDeactivate

```
export interface CanComponentDeactivate {
  canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;
}

export const canDeactivateGuard: CanDeactivateFn<CanComponentDeactivate> = (component,
                  currentRoute, currentState, nextState) => {
  return component.canDeactivate();
};
```

Príklad chráneného komponentu

```
export class LoginComponent implements OnInit, CanComponentDeactivate {
  private auth: Auth = new Auth();
  constructor(private router: Router, private usersService: UsersService) {}

  onSubmit() {
    this.usersService.login(this.auth).subscribe(
      loginStatus => {
        if (loginStatus) {
          this.auth = new Auth();
          this.router.navigateByUrl('/');
        }});
  }

  canDeactivate(): boolean | Observable<boolean> | Promise<boolean> {
    const canLeave = !(this.auth.name || this.auth.password);
    if (canLeave) return true;

    const confirmation = window.confirm(
        'Are you sure to leave? The form is partially filled and will be discarded.');
    return of(confirmation);
  }
}
```

Príklad chráneného komponentu

```
export class LoginComponent implements OnInit, CanComponentDeactivate {
  private auth: Auth = new Auth();
  constructor(private router: Router, private usersService: UsersService) {}

  onSubmit() {
    this.usersService.login(this.au
      loginStatus => {
        if (loginStatus) {
          this.auth = new Auth();
          this.router.navigateByUr
        }});
  }

  canDeactivate(): boolean | Obser
    const canLeave = !(this.auth.n
    if (canLeave) return true;

    const confirmation = window.confirm(
        'Are you sure to leave? The form is partially filled and will be discarded.');
    return of(confirmation);
  }
}
```

V pravidle zaevidujeme strážcu:

```
{
  path: 'login',
  component: LoginComponent,
  canDeactivate: [canDeactivateGuard]
}
```

ResolveFn – získanie dát pred aplikovaním pravidla

- ak chceme pred renderovaním stránky získať/pripraviť dáta, aby sa už zobrazil iba finálny komponent
  - ak neviem dodať dáta, môžem sa rovno preroutovať inde a nevytvárať komponent
- Postup:
  - vytvoríme funkciu implementujúcu ResolveFn na získanie dát, ktorá sa zavolá pred vytváraním komponentu v trase
    - ak vráti dáta, vezmeme si ich v komponente v ngOnlnit() z ActivatedRoute
    - ak nastane chyba a teda nevráti dáta, router presmerujeme inam, napr. naspäť

Resolver na získanie konkrétnej skupiny

Buď v UsersService.ts:

```
public groupResolver(route: ActivatedRouteSnapshot,
                     state: RouterStateSnapshot):
                                Observable<Group> | Observable<never> {
  const strId = route.paramMap.get('id');
  return strId ? this.getGroup(parseInt(strId)) : EMPTY;
}
```

Alebo hocikde:

```
export const groupResolver: ResolveFn<Group> = (route, state) => {
  const usersService = inject(UsersService);
  const strId = route.paramMap.get('id');
  return strId ? usersService.getGroup(parseInt(strId)) : EMPTY;
}
```

Komponent a pravidlo

```
export class GroupDetailComponent implements OnInit {
  group: Group;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.data.subscribe(
     (data: { group: Group }) => {
      this.group = data.group;
    });
  }
...
}
```

V pravidle namapujeme resolver na premennú group:

```
{
   path: ':id',
   component: GroupDetailComponent,
   resolve: {
      group: groupResolver
   }
}
```

Strážca CanMatchFn

- použije sa v trase, kde sa robí loadChildren

```
{
  path: 'group',
  loadChildren: () => import('./groups/groups.module'),
  canMatch: [authMatchGuard]
}
```

- metódu canMatchFn píšeme takmer analogicky ako canActivateFn
  - vracia boolean | Observable<boolean> | Promise<boolean>
    - true ak chceme pravidlo aplikovať,
    - false ak nechceme – tiež môžeme pred vrátením false prenavigovať inam
  - na vstupe metódy dostaneme route:Route
    - URL získame cez route.path

Preloading

- máme 2 možnosti
  - defaultne sa všetky moduly, ktoré majú nastavený lazy loading cez loadChildren natiahnu až po naroutovaní na príslušnú URL
  - preloadingStrategy – loaduje sa na pozadí každý modul, ktorý určí stratégia a nie je chránený s CanMatchFn
    - PreloadAllModules – stratégia vyberá každý modul
    - vlastná stratégia – implementujúca PreloadingStrategy

nastavenie preloadingStrategy

- pridáme parameter do metódy prodiveRouter() hlavného konfiguračného súboru

app.config.ts

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(
      routes,
      withPreloading(PreloadAllModules)
    ),
    …
  ],
};
```

Vlastná stratégia

- dodáme si do pravidla vlastné dáta

```
{
  path: 'group',
  loadChildren: () => import('./groups/groups.module'),
  data: { preload: true }
}
```

- vyrobíme vlastnú stratégiu, v ktorej automaticky natiahneme na pozadí iba také moduly, ktoré majú v dátach pravidla preload == true
- ostatné sa dotiahnu až po preroutovaní na nich
- v hlavnom routri nahradíme svoju stratégiu za PreloadAllModules

Vlastná stratégia

```
import { Injectable } from '@angular/core';
import { PreloadingStrategy, Route } from '@angular/router';
import { Observable, of } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class SelectivePreloadingStrategyService implements PreloadingStrategy {

  preload(route: Route, load: () => Observable<any>): Observable<any> {
    if (route.data && route.data.preload) {
      console.log('Preloaded: ' + route.path);
      return load();
    } else {
      return of(null);
    }
  }
}
```

Routovanie – čo sme nebrali

- animácie pri navigácii medzi pravidlami
  - https://angular.io/guide/router#adding-routable-animations
- paralelné routovanie v pomenovaných outletoch
  - https://angular.io/guide/router#displaying-multiple-routes-in-named-outlets