ANGULAR 8

Peter Gurský, peter.gursky@upjs.sk

MatPaginator

- Komponent na zobrazenie paginácie
- Keď používateľ interaguje s komponentom, generuje cez výstupný prúd ‘page’ objekt typu PageEvent
- Na nastavenie komponentu je možné použiť viacero vstupných parametrov

```
<mat-paginator
  [length]="users.length"
  [pageIndex]="0"
  [pageSize]="10"
  [pageSizeOptions]="[2, 5, 10, 20]"
></mat-paginator>
```

MatTableDataSource<Entita>

- Zdroj statických dát pre tabuľku
- Má vstavanú podporu pre získavanie vstupov z komponentov MatPaginator a MatSort
- Má podporu pre filtrovanie riadkov pomocou reťazcového filtra
  - Filter sa nastaví cez premennú filter
  - Výsledné dáta zodpovedajúce filtru vieme aj získať cez premennú filteredData

Použitie paginátora

- Potrebujeme získať referenciu na detský komponent paginátora do signálnej premennej v komponente
  - paginator = viewChild.required(MatPaginator);
- Poskytneme túto referenciu pre MatTableDataSource v ngAfterViewInit()
  - this.dataSource.paginator = this.paginator();

Filtrovanie

- Na filtrovanie nemáme žiaden špeciálny komponent, použijeme obyčajný input a reagujeme na udalosť keyup
- daný reťazec potom vložíme do MatTableDataSource cez premennú filter
  - this.dataSource.filter = filterValue
  - je fajn paginátoru povedať, nech sa potom hneď presunie na prvú stránku
    - this.dataSource.paginator.firstPage();
- Ak chceme vlastnú implementáciu fitrovania nasetujeme do MatTableDataSource funkciu v tvare ((data: T, filter: string) => boolean) cez premennú filterPredicate

Feature Module

- Keď aplikácia rastie, môžeme ju rozdeliť na rôzne zamerané časti, ktoré môžeme organizovať do modulov

```
ng generate module groups
```

- vznikne súbor groups.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  declarations: [],
  imports: [CommonModule]
})
export class GroupsModule {}
```

Komponent v module

ng generate component groups-list

- Ak chceme zabaliť komponenty do modulu, uvádzajú sa v @NgModule v časti imports:

```
import { GroupsList } from './groups-list/groups-list';

@NgModule({
  imports: [GroupsList]
})
export class GroupsModule {}
```

Importovanie modulu

Kód nejakého komponentu:

```
import { GroupsModule } from '../modules/groups/groups.module';

@Component({
...
imports: [
    ...
    GroupsModule
    ...
  ],
...
})
export class DajakyComponent { }
```

Podobne sme importovali aj iné moduly ako FormsModule, MatButtonModule,…

Export komponentov

- ak chceme komponentom, ktoré importujú modul umožniť, aby použili komponent z modulu, musíme ho z modulu exportovať, doplníme:

groups.module.ts

```
@NgModule({
  imports: [ GroupsList ],
  exports: [ GroupsList ]
})
export class GroupsModule { }
```

Asynchrónne routovanie

- Umožňuje natiahnuť iba úvodné komponenty a až potom nejaké iné komponenty alebo feature moduly
  - buď natiahnuť na pozadí pokiaľ používateľ už pracuje s úvodnými komponentami
  - alebo ich dotiahnuť až po navigácii na príslušnú URL
- Aj veľké aplikácie pracujú svižne
  - nemá zmysel naťahovať niečo, čo sa nepoužije, napr. ak používateľ ani nemá práva pre danú sekciu
  - ideálne natiahnem len to, čo aj použijem
  - tie časti, kde pravdepodobne používateľ pôjde neskôr, netreba spracovať pred prvým klikom do stránky, no budú už pravdepodobne pripravené, keď tam neskôr pôjde

(Samo)routovanie modulu

- Vytvoríme si súbor group.routes.ts po vzore app.routes.ts
- Zabalíme modulové pravidlá routra do modulu

group.routes.ts

```
import { Routes } from '@angular/router';
import { GroupsList } from
                        './groups-list/groups-list';

export const GROUP_ROUTES: Routes = [
  {path: 'list', component: GroupsList}
];
```

group.module.ts

```
import { GROUP_ROUTES } from './group.routes';

@NgModule({
  imports: [GroupsList
            RouterModule.forChild(GROUP_ROUTES)],
  exports: []
})
export class GroupsModule { }
```

Komponent neexportujeme, má k nemu prístup router modulu

Lazy loading

- nať ahovanie modulu na požiadanie
- importujem do v pravidle v app.routes.ts:

```
{
  path: 'groups',
  loadChildren: () =>  import('./groups/groups.module').then(mod => mod.GroupsModule)
}
```

- ak modul exportujem ako default, tak stačí v pravidle napísať:

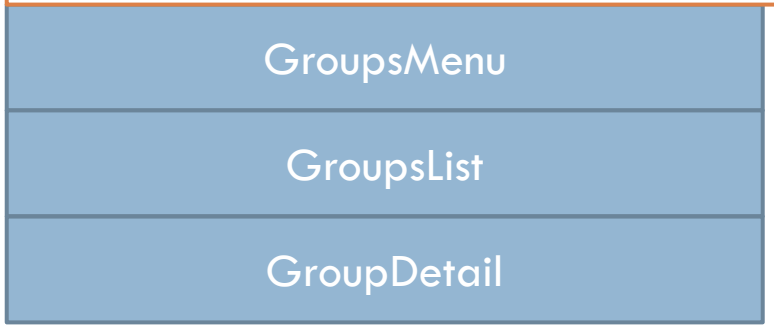
```
{
  path: 'groups',
  loadChildren: () =>  import('./groups/groups.module')
}
```

Hierarchické routovanie

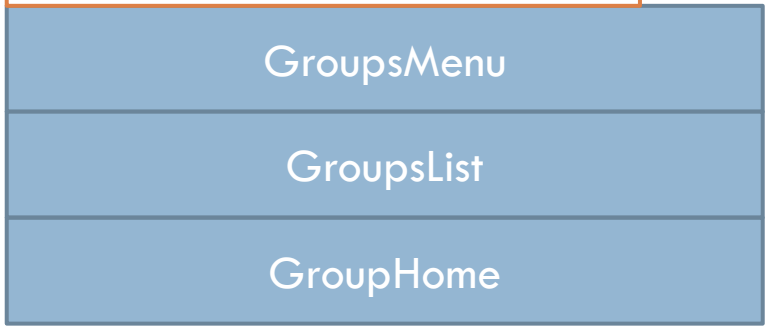
- pod každou trasou môžeme vytvoriť jej deti
- ak je trasa úspešná, vykreslí svoj komponent a ak na jeho koniec dodáme <router-outlet></router-outlet>, tak na tomto mieste umiestni komponent úspešnej trasy niektorého zo svojich detí

```
const groupsRoutes: Routes = [
 {
   path: '',
   component: GroupsMenu,
   children: [
    {
      path: '',
      component: GroupsList,
      children: [
       {
         path: 'detail/:id',
         component: GroupDetail
       },
       {
         path: '',
         component: GroupHome
       } ] } ] } ];
```

http://localhost:4200/groups/detail/2



http://localhost:4200/groups



Hierarchické routovanie

- router defaultne znovupoužíva komponenty detských pravidiel
- každé dieťa predstavuje rozšírenie URL adresy o svoje ju vlastnosť path
  - pozerá sa napravo od toho, čo použil z URL rodič
- deti môžu používať aj absolútne cesty – vtedy path začína s /
- pri navigovaní je možné používať aj "../" na posun o úroveň vyššie v ceste
  - napr. ak chceme z aktuálnej skupiny ísť na skupinu 5:
  - this.router.navigate(['../', { id: 5 }], { relativeTo: this.route });

viac možností, pre druhý parameter:
https://angular.dev/api/router/NavigationExtras