ANGULAR – 11. ČASŤ

Peter Gurský, peter.gursky@upjs.sk

Material table

- Minimalistický príklad:
  - mat-text-column – len pre reťazcové premenné
  - matHeaderRowDef – na iterovanie stĺpcov podľa parametra **name** v mat-text-column – číta **headerText**
  - matRowDef – na iterovanie riadkov, číta row['name'] a row['email']

inštančná premenná users=[
{name: 'Jano', email: 'j@j.sk'},
{name: 'Fero', email: 'f@j.sk'}];

```
<table mat-table [dataSource]="users">
  <mat-text-column name="name" headerText="Name"></mat-text-column>
  <mat-text-column name="email" headerText="E-mail"></mat-text-column>

  <tr mat-header-row *matHeaderRowDef="columnsToDisplay"></tr>
  <tr mat-row *matRowDef="let row; columns: columnsToDisplay"></tr>
</table>
```

columnsToDislpay=['name', 'email']

Material table

- jednoduché získanie dát zo servera
- implementujeme interface AfterViewInit – aby sme si zvykli, keď budeme používať MatPaginator resp. MatSort – kde je to žiadúce

```
users = [];

constructor(private userServerService: UserServerService) {}

ngAfterViewInit() {
  this.userServerService
    .getExtendedUsers()
    .subscribe(users => (this.users = users));
}
```

Zložitejšie stĺpce

ng-container sa nestane elementom v DOM

```
<ng-container matColumnDef="lastLogin">
  <th mat-header-cell *matHeaderCellDef>Last login</th>
  <td mat-cell *matCellDef="let user">
    {{ user.lastLogin | date: 'd.M.y H:mm:ss' }}
  </td>
</ng-container>
```

Vlastná pipe

- Spravíme si vlastnú pipe pre výpis názvov skupín a práv z nich vyplývajúcich
  - transform(values: Group[], property?: string): string { }
- ng g pipe groups-to-string
  - importuje sa do .module.ts
- použijeme ju v šablóne

```
<ng-container matColumnDef="permissions">
  <th mat-header-cell *matHeaderCellDef>Permissions</th>
  <td mat-cell *matCellDef="let user">
    {{ user.groups | groupsToString: 'permissions' }}
  </td>
</ng-container>
```

Dialóg na potvrdenie zmazania

- https://material.angular.io/components/dialog/api
- dialóg je obyčajný komponent, ktorý si nechá v konštruktore injektnúť MatDialogRefa prípadné dodatočné dáta
  - constructor(private dialogRef: MatDialogRef<MyDialogComponent>, @Inject(MAT_DIALOG_DATA) private data: any) { }
  - dialogRef má metódu close(myResult), ktorým vieme odovzdať voľbu používateľa cez Observable kódu, ktorý dialog vyvolal
- vyvolanie dialógu:
  - necháme si injektkúť dialog: MatDialog v konštruktore
  - zavoláme const dialogRef = this.dialog.open(MyDialogComponent, {data: "delete?"})
  - počkáme si na odpoveď: dialogRef.afterClosed().subscribe(result => { … })

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 premennej v komponente
  - @ViewChild(MatPaginator, { static: false }) paginator: MatPaginator;
  - static:false bude default od Angularu 9, nebude to treba písať
- 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

DataSource<Entita>

- Interface, ktorý vynucuje 2 metódy
  - connect(): Observable<Entita[]>
    - Tabuľka sa updatuje, keď výstupný prúd emituje novú hodnotu
  - disconnect()
- V konštruktore implementujúcej triedy si vypýtame, čo potrebujeme v metóde connect
- Inštancia datasource-u je modelom pre tabuľku
  - <table mat-table [dataSource]="datasource">

DataSource<Entita>

- Typický postup je Interface, ktorý vynucuje 2 metódy
  - connect(): Observable<Entita[]>
    - Tabuľka sa updatuje, keď výstupný prúd emituje novú hodnotu
  - disconnect()
- V konštruktore implementujúcej triedy si vypýtame, čo potrebujeme v metóde connect
- Inštancia datasource-u je modelom pre tabuľku
  - <table mat-table [dataSource]="datasource">

Kombinovanie vstupných prúdov

```
connect(): Observable<User[]> {
  const usersObservable = this.usersService.getUsers()
     .pipe(tap(users => this.users = users));

  const dataMutations = [
    usersObservable,
    this.paginator.page,    // MatPaginator
    this.sort.sortChange    // MatSort
  ];

  this.paginator.length = this.users.length;

  return merge(...dataMutations).pipe(map(() => {
    return this.getPagedData(this.getSortedData(
            [...this.users]));
  }));
}
```