ANGULAR – 3.ČASŤ

Peter Gurský, peter.gursky@upjs.sk

Routing

- Vytvoríme si komponent na prihlásenie
- Chceli by sme ho vidieť **namiesto** zoznamu používateľov
- Cez routing vieme mapovať rôzne koncovky našej URL adresy na rôzne komponenty
  - http://localhost:4200/users
    - Stránka so zoznamom používateľov
  - http://localhost:4200/login
    - Stránka s prihlasovacím formulárom

Pripravíme si routing

- cd src/app
- ng g component login
  - vytvorí komponent, kde budeme vyrábať prihlasovací formulár

Nastavenie routovania v app.routes.ts

```
import { Routes } from '@angular/router';
import { UsersComponent } from './users/users.component';
import { LoginComponent } from './login/login.component';

export const routes: Routes = [
        { path: 'users', component: UsersComponent },
        { path: 'login', component: LoginComponent }
];
```

koniec URL adresy
http://localhost:4200/login

Komponent, ktorý sa má natiahnuť

Nastavíme AppComponent

```
<div style="text-align:center">
        <h1>Správa používateľov</h1>
</div>
<app-users>info o používateľoch</app-users>
<router-outlet></router-outlet>
```

- Vyskúšame
  - http://localhost:4200/users
  - http://localhost:4200/login
  - http://localhost:4200/

Redirect z / na /users

- Pridáme ďalšiu cestu do app-routing.module.ts

```
{ path: '', redirectTo: '/users', pathMatch: 'full' }
```

- kedykoľvek sa znavigujeme na http://localhost:4200/ tak nás presmeruje na http://localhost:4200/users
- pathMatch
  - full – za path '' v URL už nič ďalšie nie je
  - prefix – stačí ak sa prefix URL a path zhoduje, za ním v URL ešte niečo môže byť

Page not found

- keď sa používateľ nanaviguje na neexistujúcu URL, chceme mu zobraziť vlastnú stránku s chybou 404

```
{ path: '**', component: PageNotFoundComponent }
```

- Na poradí pravidiel záleží: použije sa prvé pravidlo, ktorého path sa zhoduje s URL
- Pravidlo s path= '**', ktoré sa zhoduje s každou URL preto píšeme ako posledné

REST server - metódy

Ďalší preddefinovaní používatelia a ich heslá :
Lucia : lucia
John : john
Andrej : andre

- GET: /users
- POST: /login
  - V tele pošleme {"name": "Peter", "password": "sovy"}
  - príde vygenerovaný token
- GET: /users/{token}
  - prídú používatelia aj s neverejnými atribútmi
- GET: /user/{id}/{token}
- GET: /bg-user/{id}/{token}
  - vnútorná reprezentácia používateľa (obsahuje aj heslo)
- POST: /users/{token}
  - Cez POST pošleme JSON používateľa, ktorého chceme uložiť
- DELETE: /user/{id}/{token}

Prihlásenie používateľa - plán

- Spravíme si triedu Auth s premennými name a password
  - Cez HTTP budeme posielať JSON objekt – konverzia z javascriptového objektu sa spraví sama
- V LoginComponent si vytvoríme prihlasovací formulár s tlačidlom, ktoré iniciuje poslanie cez service komunikáciu so serverom
  - Nájdeme si na internete nejakú šablónu pre login formulár
- Použijeme HTTP metódu POST

Model editačného komponentu

- Základný model komponentu, v ktorom budeme editovať prihlasovacie údaje je objekt typu Auth
- Môžeme si ho v komponente vyrobiť prázdneho, aby sa prvky šablóny mali s čím previazať

```
...
export class LoginComponent {
   auth:Auth = new Auth("","");
   ...
}
```

Pomocný text s obsahom modelu/Auth

app/login/login.component.html (časť)

```
<div class="modal-body">
        <p>aktuálne údaje: {{vypisAuth}}</p>
</div>
```

app/login/login.component.ts

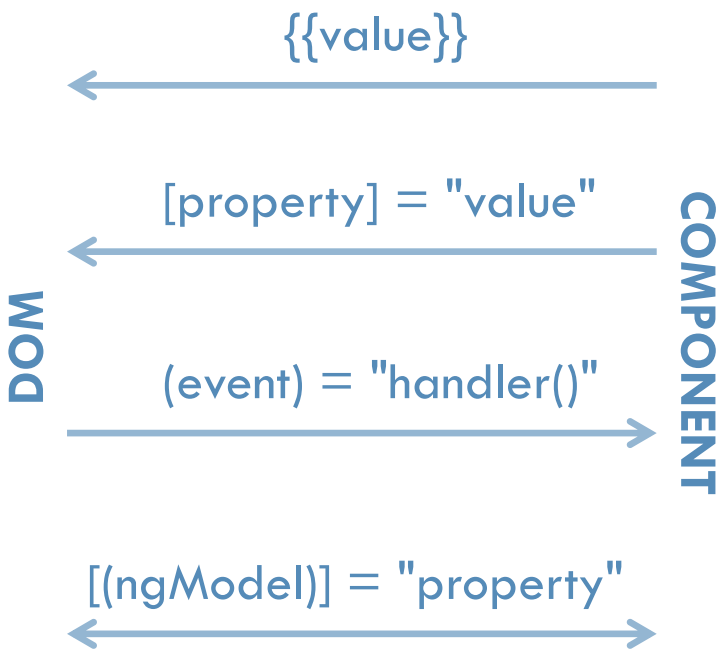
```
get vypisAuth():string {
    return JSON.stringify(this.auth);
}
```

Udalosti vo formulári

- Jedna z hlavných výhod MVC v prehliadači je jednoduchá spätná väzba pri editácii formulárov
- Webový formulár nám mení model komponentu
  - ten môžeme okamžite vyhodnocovať, či je ok
  - ...a dať používateľovi spätnú väzbu

Prepojenie DOM a modelom v komponente v Angulari

- DOM sa buduje ako kombinácia šablóny a obsahu modelu
- V DOM sa odchytávajú udalosti používateľa, ktoré môžeme poslať komponentu
  - zavolaním metódy
  - obojsmerným prepojením s modelom cez [(ngMmodel)]



Udalosti

app/login/login.component.html (časť)

```
<input type="text" (keyup)="setAuthName($event)" [value]="auth.name">
```

app/login/login.component.ts

```
setAuthName(event: any) {
    this.auth.name = event.target.value;
}
```

- premenná $event obsahuje informácie o udalosti
- na zobrazenie obsahu potrebujeme aj mapovanie z komponentu do DOM
- potrebujeme obslužnú metódu
- potrebujeme vedieť, že element input má atribút value

Previazanie modelu komponentu a DOM

Pred tým:

```
<input type="text" (keyup)="setAuthName($event)" value="{{auth.name}}" />
```

Po tom:

```
<input type="text" [(ngModel)]="auth.name" name="login" />
```

Ľubovoľné unikátne meno pre každý ngModel formulára

- importujeme v komponente FormsModule
- ngModel previaže obojstranne element formulára s premennou modelu komponentu
- nepotrebujeme obslužnú metódu
- nepotrebujeme vedieť, že aký element obsluhujeme

Odoslanie obsahu formuára serveru

- ...heslo user-a urobíme analogicky
- odoslanie spravíme cez tlačidlo na uloženie
  - obalíme celú šablónu do elementu <form>
  - v komponente vytvoríme obslužnú metódu onSumbit()

app/login/login.component.html (časť)

```
<form (ngSubmit)="onSubmit()" ... >
...
   <button type="submit" ... >Sign in</button>
</form>
```

Prihlásenie používateľa - service

app/users/users-service.ts

```
import { HttpClient } from '@angular/common/http';
import { Auth } from '../auth';

...
private restServerUrl: string = "http://localhost:8080/";

 login(auth: Auth):Observable<string> {
    return this.httpClient.post(this.restServerUrl + "login", auth,
                                                    {responseType : 'text'});
 }
```

Nepríde JSON ale iba string

- Vrátime tak token komponentu, ktorý bude volať subscribe na Observable z login()

Uvažujme inak

- Token komponentu netreba (nezobrazujeme ho)
  - Môže si ho získať aj service
  - Nemusíme pri každom volaní servisných metód posielať token ako parameter
  - Viaceré komponenty môžu využiť servisné metódy s tokenom
- Zároveň však chceme poslať komponentu informáciu o úspechu prihlásenia
- Kto má volať subscribe() na Observable?
  - subscribe() zavolá stále komponent, lebo iniciuje spustenie komunikácie pri stlačení tlačidla
- Service môže pristúpiť do prúdu prijatých dát pred odoslaním iniciátorovi
  - Môže prijaté dáta spracovať, dokonca aj upraviť a poslať komponentu už upravené

Pošleme true, ak sa prihlásenie podarí

```
private token: string = null;

login(auth: Auth):Observable<boolean> {
    return this.httpClient.post(this.restServerUrl + "login",auth,
                                                    {responseType : 'text'})
    .pipe(map(token => {
        this.token = token;
        return true;
    }));
}
```

Ak sa heslo nezhoduje, pošleme false

```
login(auth: Auth):Observable<boolean> {
  return this.httpClient.post(...)
   .pipe(map(...),
     catchError(error => {
      if (error instanceof HttpErrorResponse &&
            error.status == 401)
          return of(false); // zlé heslo
      }
      return throwError(error); // nejaká iná chyba
  }));
}
```