

ANGULAR 3



Peter Gurský, peter.gursky@upjs.sk

Prihlásenie používateľa - plán

- Spravíme si triedu Auth s premennými name a password
 - ▣ Budeme posielat' JSON objekt – konverzia sa spraví sama
- V Login komponente si vytvoríme prihlasovací formulár s tlačidlom, ktoré iniciuje poslanie cez service komunikáciu so serverom
 - ▣ môžeme použiť material komponent card
- 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
- Použijeme obyčajnú nesignálnu premennú

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

Pomocný text s obsahom modelu/Auth

pages/login/login.html (časť)

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

pages/login/login.ts

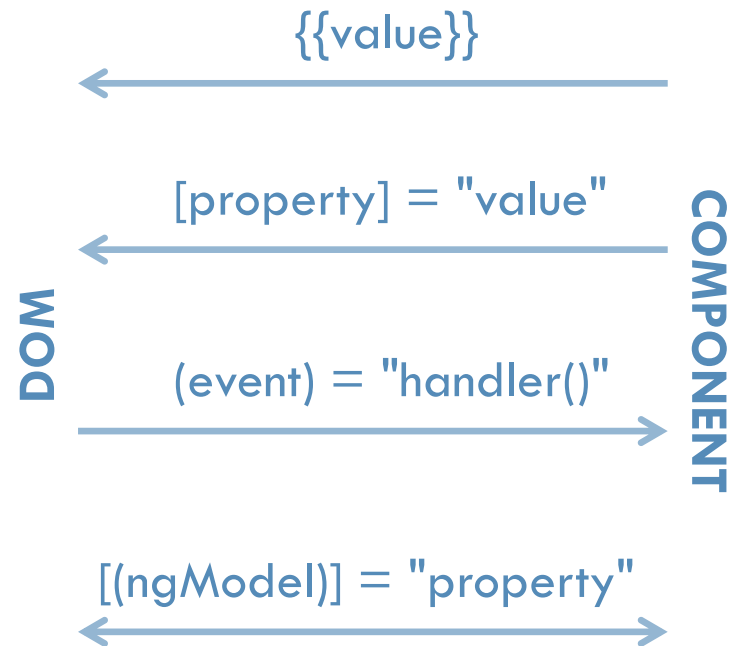
```
get vypisAuth():string {  
    return JSON.stringify(this.auth);  
}
```

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 odchyťávajú udalosti používateľa, ktoré môžeme poslať komponentu
 - ▣ zavolaním metódy
 - ▣ obojsmerným prepojením s modelom cez [(ngModel)]



Udalosti

pages/login/login.html (časť)

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

pages/login/login.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 formulára serveru

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

pages/login/login.html (časť)

```
<form (ngSubmit)="onSubmit()" ... >
```

```
...
```

```
<button type="submit" ... >Sign in</button>
```

```
</form>
```

Prihlásenie používateľa - service

services/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.http.post(this.restServerUrl + "login", auth,
                                                                    {responseType : 'text'});
}
```

Nepríde JSON ale
iba string

- Vrátíme 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 posilať 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 = "";

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

Dorobíme login komponent

- Ak príde **true**, môžeme zobrazit' inú stránku

```
constructor(private router: Router, ...) { }
```

```
this.router.navigateByUrl('/users');    // v subscribe
```

- Ak príde **false**, zobrazíme hlášku o zlom hesle
- Ak príde chyba, zobrazíme hlášku o chybe komunikácie

Refaktor

- Chybové/úspechové hlášky by mohol vypisovať nejaký "message component"
 - ▣ vypisovať chybové, ale aj pozitívne správy
 - ▣ všetky možné chyby komunikácie
 - nedostupný server
 - zlý login alebo heslo
 - nedostatočné práva
 - ▣ je vhodné prebaľiť chyby do chybovej správy
- Na zobrazenie môžeme použiť Material component Snackbar
 - ▣ Vyvoríme si service, cez ktorý budeme posielat' pozitívne aj negatívne správy
 - môžeme si prispôbiť farby cez vlastnosť `panelClass` a nastavenie `css`

Navigačná lišta

- Využijeme material komponent toolbar
 - ▣ potrebujeme aj komponenty button a icon
- V navigácii dáme linky na všetky zatiaľ používané URL adresy

UsersTable komponent

- Zapýtame si rozšírených používateľov
 - GET: `http://localhost:4200/users/{token}`
- Rozšírime triedu User o ďalšie parametre
 - Aj pole objektov typu Group
- Vypíšeme tabuľku s rozšírenými používateľmi