ANGULAR 2

Peter Gurský, peter.gursky@upjs.sk

Flow control v template

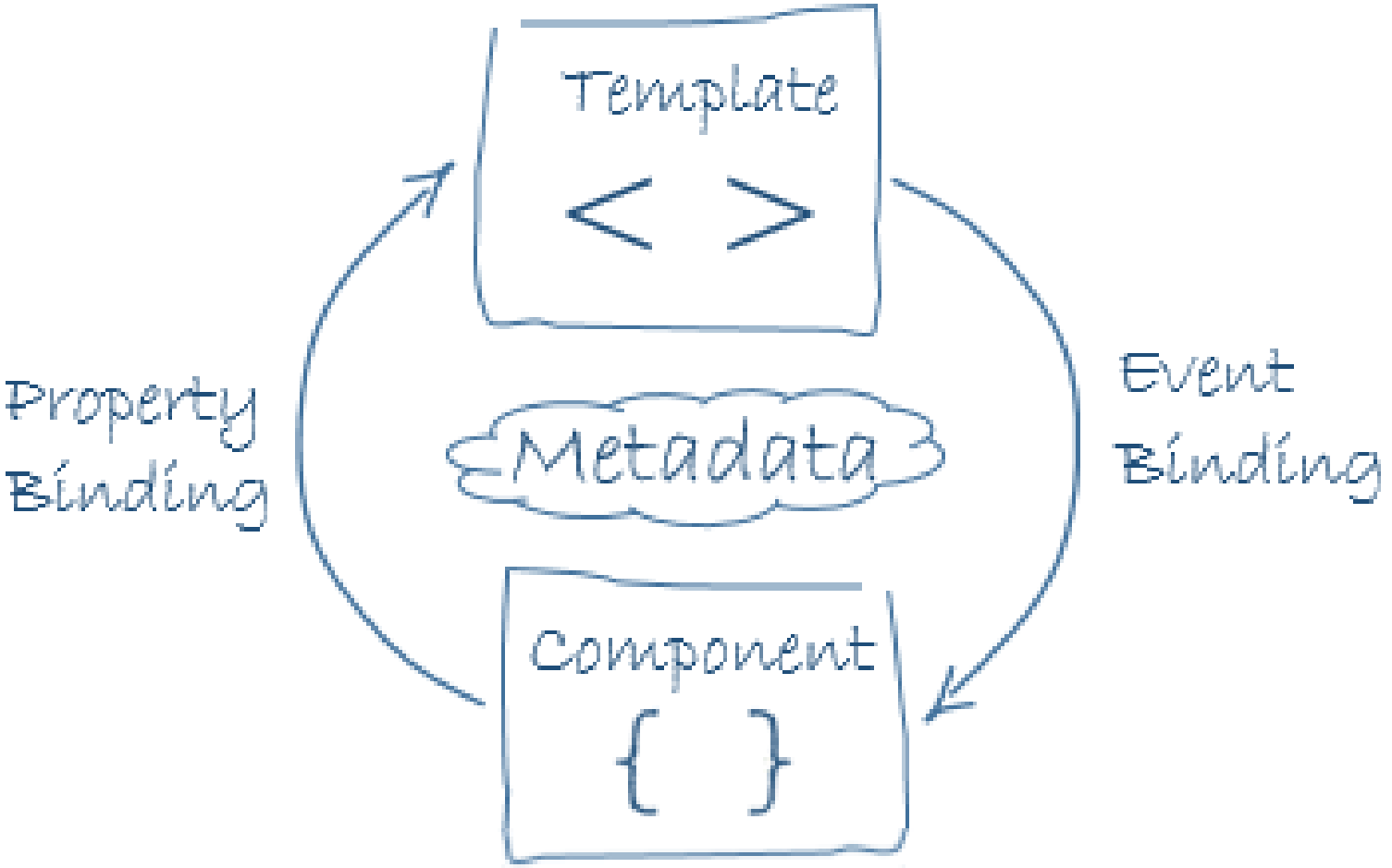
```
@if (a > b) {
  {{a}} is greater than {{b}}
} @else if (b > a) {
  {{a}} is less than {{b}}
} @else {
  {{a}} is equal to {{b}}
}
```

```
@switch (accessLevel) {
  @case ('admin') { <admin-dashboard/> }
  @case ('moderator') { <moderator-dashboard/> }
  @default { <user-dashboard/> }
}
```

```
@for (item of items; track item.id) {
  <li> {{ item.name }} </li>
} @empty {
  <li> There are no items. </li>
}
```

Ak neviete dodať unikátnu hodnotu, použite `track $index`

Architektúra komponentov Angularu



Služby (Services)

- Časom môžeme mať viac komponentov, ktoré potrebujú používateľov
- Používateľov nemá spravovať komponent, ale perzistentná vrstva na serveri
- Náš cieľ - vytvoriť službu starajúcu sa o pole users
  - bude komunikovať s REST serverom
  - a sprostredkovávať tak pre komponenty CRUD operácie nad používateľmi na serveri
- Najprv si spravíme komunikáciu služby s komponentmi
  - Služba bude zatiaľ poskytovať len svoje lokálne pole používateľov

Vytvorenie služby

- Prejdeme do adresára, kde chceme mať službu, napr. src/services
- Spustíme príkaz
  - ng g service users
- Vzniknú 2 súbory
  - src/services/users.service.spec.ts
  - **src/services/users.service.ts**
    - Vytvoríme getter vracajúci pole používateľov

Komponent potrebuje službu - injektneme

app/users/users-component.ts

```
import { UsersService } from '../services/users.service';
...
export class UsersComponent {
 constructor(private usersService: UsersService) {}
}
```

Alternatívne:

```
export class UsersComponent {
usersService = inject(UsersService);
}
```

app/users/users-service.ts

```
@Injectable({
providedIn: 'root'
})
export class UsersService {

}
```

priama registrácia v app.module od Angular 6

Komponent potrebuje service

app/users/users-component.ts

```
export class UsersComponent implements OnInit {

 constructor(private usersService: UsersService) {}

  ngOnInit() {
      this.updateUsers();
  }

  updateUsers() {
      this.users = this.usersService.getUsers();
  }
}
```

Čo môže trvať dlho* nerobíme v konštruktore!

* napr. komunikácia so serverom

Túto metódu musíme v službe vytvoriť

Príprava na dlhé čakanie na dáta

- Synchrónne volanie:
  - Component si vypýta dáta od servisu a čaká
  - Service si vypýta dáta zo servera a čaká
  - Pokiaľ sa čaká, žiaden JavaScript nefunguje (udalosti používateľa – kliky, editácia,...)
- Asynchrónne volanie
  - Component si vypýta dáta od servisu a zaeviduje funkciu, ktorá sa má spustiť, keď dáta dôjdu
  - Service si vypýta dáta zo servera a zaeviduje funkciu, ktorá sa má spustiť, keď dáta dôjdu
  - Pokiaľ sa čaká na dáta, všetko funguje

Observable

- Trieda predstavujúca prúd/producenta dát, ktoré prídu v budúcnosti
- Vieme zaregistrovať metódu, ktorá sa má spustiť, keď dáta prídu - cez volanie subscribe(metóda)
- Metóda sa zvykne vkladať ako šípková funkcia
- Moderný spôsob robenia asynchrónnych volaní
- Zmeňme metódu v službe:
  - Operátor „of“ vyrobí nový objekt typu Observable
  - Musíme ho importovať

```
import { Observable, of } from 'rxjs';

getUsers(): Observable<User[]> {
      return of(this.users);
}
```

Vyrobíme prúd dát, ktoré sa pošlú, keď na konci niekto zavolá subscribe. Tieto dáta dodáme cez parameter of().

Zaregistrovanie poslucháča v komponente

app/users/users-service.ts

```
getUsers(): Observable<User[]> {
      return of(this.users);
}
```

app/users/users-component.ts

```
updateUsers() {
   this.usersService.getUsers().subscribe(users => this.users = users);
}
```

Keď sa dáta z Observable prídu, spustí sa naša funkcia. Dáta dostane ako vstupný parameter users.

REST server

- Použijeme predpripravný REST server,
  - potrebujeme javu 17 alebo novšiu
  - stiahneme si súbor films-server.jar
    - java -jar films-server.jar
- Keď ho spustím, počúva na adrese http://localhost:8080/
  - Vyskúšame cez Postman-a http://localhost:8080/users
- Naša Angular aplikácia je prístupná cez NodeJS server na http://localhost:4200/

Pripravíme si Angular

app.config.ts

```
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes),
              provideHttpClient()
             ]
}
```

getUsers() cez AJAX volanie

app/users/users-service.ts

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
...
constructor(private http: HttpClient) { }

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

getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.restServerUrl);
}
```

Injektneme inštanciu HttpClient (alebo použijeme inject)

Hlúpe pretypovanie objektu ktorý vznikol z JSONu.
Typescript sme presvedčili, že objekt bude mať rovnakú štruktúru, ako naša trieda User, ale nikde sa to neoveruje.

Ak chceme naozajstnú inštanciu triedy aj s metódami

app/users/users-service.ts

```
import { map } from 'rxjs';
...
getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.restServerUrl).pipe(
      map(jsonUsers => jsonUsers.map(jsonUser => User.clone(jsonUser))));
}
```

Chyby pri Observable

- Ak sa komunikácia nepodarí, funkcia vložená cez subscribe sa nespustí, ale vyletí chyba do konzoly
- Ak chceme túto chybu odchytiť, môžeme v subscribe použiť objekt s listener funkciami na prípady že prídu ďalšie dáta a že príde chybový objekt

app/users/users-component.ts

```
updateUsers() {
    this.usersService.getUsers().subscribe({
        next: users => this.users = users,
        error: error => console.log('chyba: ' + JSON.stringify(error))
    });
}
```

Angular Material

- https://material.angular.io/
- ng add @angular/material
- pre každý komponent knižnice je potrebné importovať do komponentu príslušný modul
  - treba pozrieť dokumentáciu komponentu – sekcia API

Material table

- Komponent na zobrazovanie tabuľkových dát
  - Samotný komponent slúži iba na základné zobrazenie, pre zložitejšie operácie môžeme spolupracovať s ďalšími komponentmi/servismi
- na pagináciu MatPaginator
- na usporiadanie MatSort
- na filtrovanie, komunikáciu s externým úložiskom a zložitejšiu manipuláciu s dátami sa využíva
  - MatTableDataSource pre statické dáta
  - DataSource pre zložitejšie veci

Material table

- Minimalistický príklad:
  - mat-text-column – len pre primitívne 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']

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": "upjs"}
  - 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}

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

Komponent pre login formulár

- 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

~~<app-users-list></app-users-list>~~

<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.

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

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

Prihlásenie používateľa - plán

- Spravíme si triedu Auth s premennými name a password
  - Budeme posielať 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