Angular: Inštalácia, TypeScript, komponenty a ich vlastnosti a udalosti

UINF/PAZ1c

epizóda 10

Tradičné webové aplikácie

- Web = HTML + CSS + JS
- Na serveri čaká aplikácia, ktorá ich generuje
- Archaický prístup
  - Skriptovací jazyk zlepuje HTML súbor a posiela ho prehliadaču
  - Zdroják = kúsky HTML na striedačku s kusmi kódu, ktorý HTML dotvára
  - JS na webe slúži iba spríjemnenie práce s webom – animácie, kontrola vstupov vo formulári

Moderné server-side webové aplikácie

- Stále platí: server generuje HTML + CSS + JS a prehliadač ich len interpretuje
- MVC na serveri
  - HTML šablóny so špeciálnymi tagmi/atribútmi, ktoré sa odkazujú na komponenty aplikácie
  - Komponent aplikácie na základe svojho modelu doplní/nahradí príslušnú časť šablóny
  - Výsledok = šablóna + reprezentácia komponentov sa posiela zo servera ako výsledné HTML + CSS +JS
- Keď sa zmení model, dotiahne sa zo servera iba tá časť HTML, ktorá predstavuje daný komponent – AJAX volania
- V mnohých jazykoch: Java, C#, PHP, JavaScript,...
  - Java: JavaServer Faces, Apache Wicket, Vaadin,...

Moderné client-side webové aplikácie

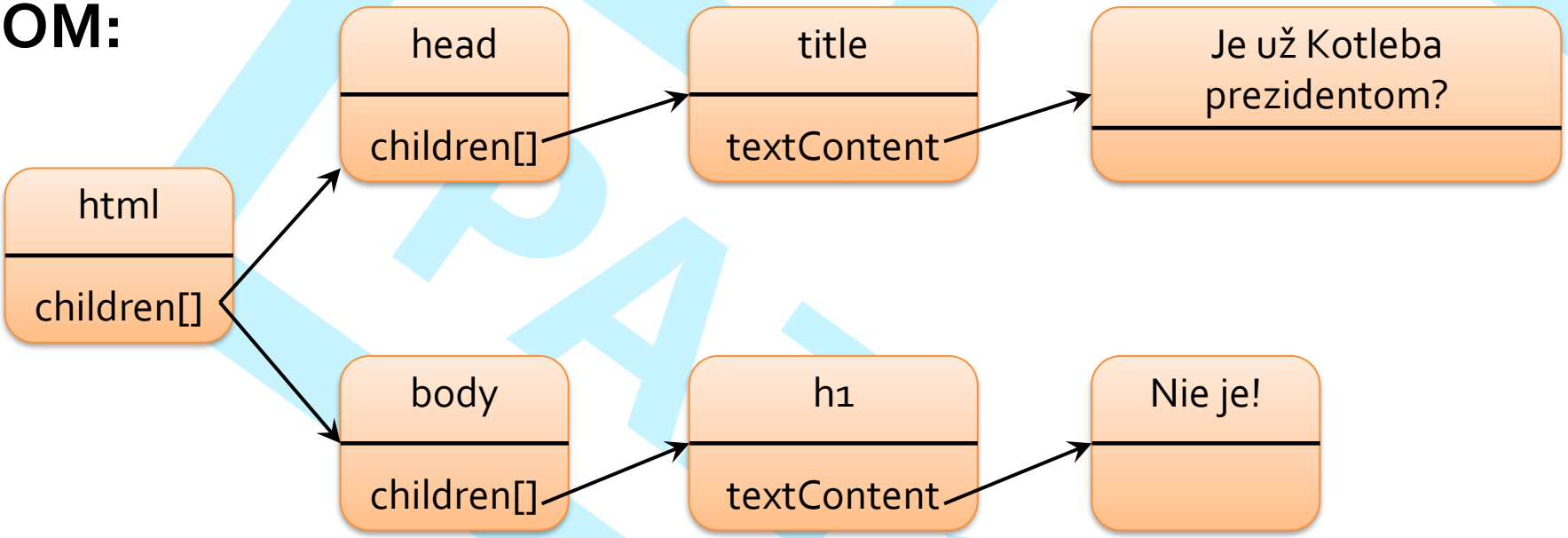
- Čistý JavaScript
- a.k.a. single-page applications, fat (thick) client
- Server poskytuje/prijíma entity – REST volania
  - perzistentná a business vrstva + REST service
  - ľubovoľný jazyk (Java, PHP, JavaScript,...)
- MVC v prehliadači
  - aplikácia modifikuje priamo DOM model zobrazovanej stránky
  - komponenty predstavujú podstromy DOM modelu
  - modely komponentov typicky predstavujú entity poskytované REST serverom
- Angular, Vue, React, Ember, Meteor, ExtJS, Aurelia,..

HTML:

```
<html>
 <head>
  <title>Je už Kotleba prezidentom?</title>
 </head>
 <body>
  <h1>Nie je!</h1>
 </body>
</html>
```

DOM model tvoria JS objekty typu
Node
a okrem textových uzlov aj typu
Element, HTMLElement, ...

DOM:



Angular (od verzie 2)

- Final release verzie 2: 14.9.2016,
  - aktuálne verzia 15
- Využíva jazyk TypeScript
  - Typovaná nadstavba JavaScriptu (od ECMAScript 6)
    - už existujú triedy (syntaktický cukor)
    - moduly
    - lambdy
    - ...
  - dodané anotácie, generiká
- framework = dopĺňame komponenty do štartovacej aplikácie

Vývojové prostredie

- Mnoho mágie na pozadí
  - Typescript sa kompiluje do ECMAScript-u, ktorým rozumie webový prehliadač (ako ktorý ☺)
    - Po každej zmene súboru
  - NodeJS server poskytuje skompilované súbory prehliadaču
  - Prehliadačom sa napojíme na NodeJS server na http://localhost:4200/
  - Prehliadač si stiahne súbory a spustí našu aplikáciu
    - Chová sa tak, ako sa bude chovať, keď sa nasadí naostro

Rozbehávame vývojové prostredie

- Nainštalujeme nodejs
  - *https://**nodejs**.org/*
  - *s ním dostaneme aj balíčkovač npm*
- IDE (od Microsoftu):
  - Visual Studio Code: http://code.visualstudio.com/
- Nainštalujeme si zostavovač Angular projektu
  - npm install -g @angular/cli
- Vytvoríme si kostru projektu
  - ng new PROJECT_NAME
  - cd PROJECT_NAME

Import Bootstrap frameworku

- Čo je Bootstrap ?
  - de-facto štandard na pekné štýlovanie HTML stránok (CSS + JS)
  - http://getbootstrap.com/
  - http://www.w3schools.com/bootstrap/
- Nainštalujeme ho do projektu
  - npm install bootstrap

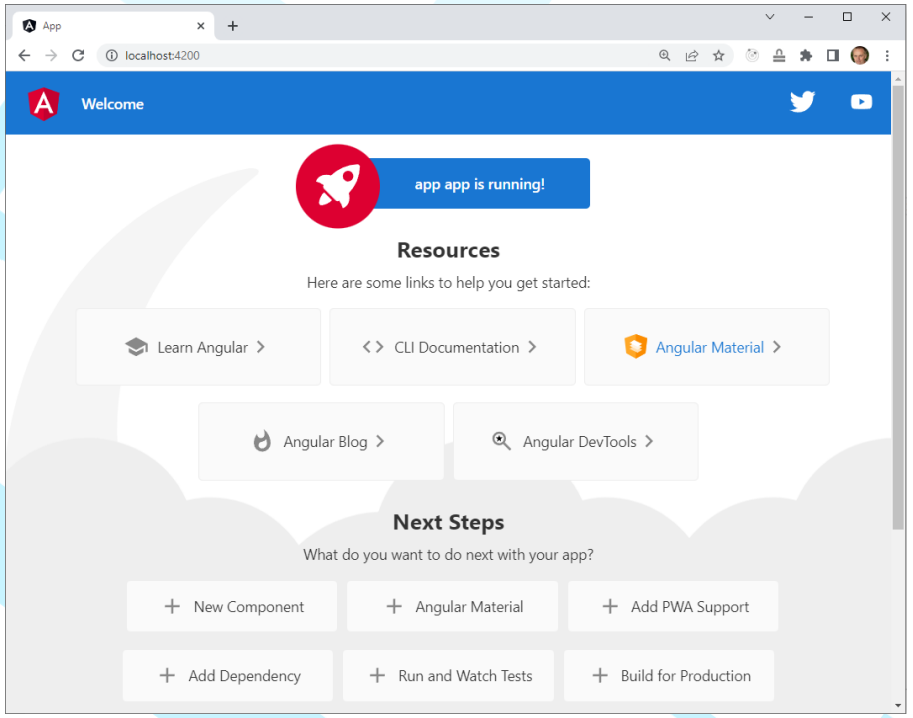
Import Bootstrap frameworku

- Doplníme do angular.json

```
“apps”: [{
…
   "styles": [
      "node_modules/bootstrap/dist/css/bootstrap.min.css",
      "styles.css" ],
   "scripts": [
      "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
   ],
…
}]
```

Spúšťame projekt

- Vojdeme do koreňa projektu a spustíme NodeJS server:
  - ng serve
- V prehliadači zadáme URL http://localhost:4200/



Čo nám to vzniklo?

- node_modules
  - …
- src
  - app
    - **app.component.css**
    - **app.component.html**
    - app.component.spec.ts
    - **app.component.ts**
    - **app.module.ts**
  - index.html
  - main.ts
  - …

kopa modulov/knižníc, ktoré môžeme využiť

tu sa budú nachádzať naše veci

HTML šablóna koreňového komponentu

trieda koreňového komponentu

Koreňový modul
modul= obal pre množinu komponentov, služieb, definuje koreňový komponent,…

Hlavná stránka, ktorá sa v tele odkazuje na koreňový komponent (telo nemeníme, max. hlavičku)

Letmo o OOP v Typescripte

```
class Student {
  meno:string;
  vek: number;
  constructor(name:string, vek:number = 19) {
     this.name = name;
     this.vek = vek;
  }
  povedzMenoAVek(musis:boolean): string {
     if (musis) return meno+" " + vek;
     else return "nepoviem";
  }}

let jano= new Student("Jano");
let janoPovedal = jano.povedzMenoAVek(true); // "Jano 19"
```

Kód vo viacerých súboroch

```
// student.ts
export class Student {
  ...
}

export class SuperStudent extends Student {
  ...
}
```

```
// tabulka.ts
import {Student, SuperStudent} from student;

let jano= new Student("Jano", 25);
let brunhilda = new SuperStudent("Brunhilda");
```

Vloženie komponentu do stránky

src/index.html

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8">
 <title>nazov aplikacie</title>
 <base href="/">
 ...
</head>
<body>
 <app-root></app-root>
</body>
</html>
```

src/app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 title:string = 'app';
}
```

src/app/app.component.html

```
<h1>
 Welcome to {{title}}!
</h1>
```

zobrazenie inštančnej premennej

Vložme vlastný komponent do hlavného

- Ideme do príkazového riadku a v adresári src/app spustíme
  - ng g component users
- Komponent sa importne do src/app-module.ts
- Vznikú 4 súbory komponentu
  - Ako selector v src/app/users/users.component.ts sa nastaví **app-users**
- Ak ho chcem vidieť v inom komponente, pridáme do jeho šablóny
  - **<app-users>**info o používateľoch**</ app-users>**

Zobrazenie jednotlivých hodnôt

app/users/users-component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({...})
export class UsersComponent implements OnInit {
    title: string = "Zoznam používateľov";
    users: string[] = ["Janko", "Marienka"];
}
```

app/users/users-component.html

```
<h2>{{title}}</h2>
<ul>
   <li>{{users[0]}}</li>
   <li>{{users[1]}}</li>
</ul>
```

Zobrazenie zoznamu cez direktívu *ngFor

app/users/users-component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({...})
export class UsersComponent implements OnInit {
    title: string = "Zoznam používateľov";
    users: string[] = ["Janko", "Marienka"];
}
```

app/users/users-component.html

```
<h2>{{title}}</h2>
<ul>
    <li *ngFor="let user of users">{{user}}</li>
</ul>
```

Radšej pracujme s entitami

```
// app/User.ts
export class User {
    constructor(
        public chipId: string,
        public name: string,
        public id?: number,
        public active: boolean = false
    ) { }
}
```

Vyrobme si triedu entity User

```
// app/User.ts
export class User {
    constructor(
        public chipId: string,
        public name: string,
        public id?: number,
        public active: boolean = false
    ) { }
}
```

triedu budeme vedieť použiť v iných súboroch

Vyrobme si triedu entity User

```
// app/User.ts
export class User {
    constructor(
        public chipId: string,
        public name: string,
        public id?: number,
        public active: boolean = false
    ) { }
}
```

nie všetky parametre sa pri volaní konštruktora musia zadať,
id je defaultne **undefined**,
active je defaultne **false**

Vyrobme si triedu entity User

```
// app/User.ts
export class User {
    constructor(
        public chipId: string,
        public name: string,
        public id?: number,
        public active: boolean = false
    ) { }
}
```

z parametrov konštruktora sa stanú verejne prístupné inštančné premenné

```
jano = new User("abcd", "jano", 2);
console.log(jano.name);
```

Pre neúnavných pisateľov

```
export class User {
    private _name: string;

    set name(newName: string) {
        this._name = newName;
    }
    get name(): string {
        return this._name;
    }
}
```

```
jano = new User();
jano.name = "Jano";
console.log(jano.name);
```

Iterujeme entity

app/users/users-component.ts

```
export class UsersComponent implements OnInit {
...
    users: User[] = [new User("abcd","jano"),
                     new User("xyz","marienka")];
}
```

app/users/users-component.html

```
...
<ul>
    <li *ngFor="let user of users">{{user.chipId}}, {{user.name}}</li>
</ul>
```

...alebo do tabuľky

```
<table class="table table-hover">
  <thead>
    <tr><th>id</th><th>id chipu</th><th>meno</th></tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users" >
        <td>{{user.id}}</td>
        <td>{{user.chipId}}</td>
        <td>{{user.name}}</td>
    </tr>
  </tbody>
</table>
```

Odchytenie udalosti click

app/users/users-component.html (časť)

```
<tr *ngFor="let user of users" (click)="selectUser(user)" >
   <td>{{user.id}}</td>
   <td>{{user.chipId}}</td>
   <td>{{user.name}}</td>
</tr>
```

app/users/users-component.ts

```
export class UsersComponent implements OnInit {
...
  selectedUser?: User;

  selectUser(user: User) {
     this.selectedUser = user;
  }
}
```

Podmienené časti šablóny

app/users/users-component.html (časť)

```
<tr *ngFor="let user of users" (click)="selectUser(user)" >
   <td>{{user.id}}</td>
   <td>{{user.chipId}}</td>
   <td>{{user.name}}</td>
</tr>
<p *ngIf="selectedUser">Vybratý používateľ: {{selectedUser.name}} </p>
```

app/users/users-component.ts

```
export class UsersComponent implements OnInit {
...
  selectedUser?: User;

  selectUser(user: User) {
     this.selectedUser = user;
  }
}
```

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/app/services
- Spustíme príkaz
  - ng g service users
- Vzniknú 2 súbory
  - src/app/services/users.service.spec.ts
  - **src/app/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 implements OnInit {

 constructor(private usersService: UsersService) {}
}
```

Čo sa injektuje cez konštruktory, musí byť zaregistrované medzi providermi
(pre Angular <=5)

app/app.module.ts

```
import { UsersService } from
'./users.service';
@NgModule({
  ...
  providers: [UsersService],
  ...
})
export class AppModule { }
```

app/users/users-service.ts

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

}
```

priama registrácia 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 obal pre premennú, ktorá sa môže zmeniť
- Vieme zaregistrovať metódu, ktorá sa má spustiť, keď sa zmení, cez metódu subscribe(metóda)
- Metóda sa zvykne vkladať ako lambda výraz
- 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);
}
```

Zatiaľ vyrobíme jednoduchý Observable obal zo statickej hodnoty už naplnenej premennej

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 hodnota Observable zmení, spustí funkciu (v tomto prípade lambdu)

Pripravíme si Angular

app.module.ts

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

@NgModule({
...
imports: [
   ...
   HttpClientModule
 ],
...
})
export class AppModule { }
```

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://xxx.xxx.xxx.xxx:8080/";

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

Injektneme inštanciu HttpClient

Pretypujeme objekt ktorý vznikol sparsovaním prichádzajúceho JSONu

Otázky?