

Gymnázium Jana Keplera

obor vzdělání:

7941K41 Čtyřleté gymnázium pro absolventy ZŠ

Maturitní práce z informatiky

Sociální síť - Same

autor: Vít Paulík

vedoucí práce: Karel Jílek

Praha 2018

Prohlašuji, že jsem jediným autorem této maturitní práce a všechny citace, použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium Jana Keplera, Praha 6, Parlérova 2 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu

V Praze dne 13. Března 2018

.....

Anotace:

Maturitní práce má podobu sociální sítě s názvem “Same”. Jedná se o webovou aplikaci, na které se mohou vyjadřovat registrovaní uživatelé pomocí textových a obrázkových příspěvků, které se vkládají na jejich vlastní stránku. Uživatelé se také mohou navzájem sledovat, čímž se nové příspěvky sledovaných sledujícím zobrazí na “news feedu” a na příspěvky reagovat pomocí tzv. “same” (cítím se stejně).

Abstract:

The graduation project is a social network called “Same”. It’s a web application, where users can express themselves using text and image posts, which are posted to their own user page. Users can also follow each other, and watch posts of their followed users on the “news feed”, and react to posts with “same” (meaning “I feel same”).

Obsah:

Instalace	4
Úvod	6
Hlavní část	7
Funkce, design a ukázka	7
Technologie	16
Databáze	18
Frontend	20
Backend	23
Komunikace mezi serverem a frontendem	24
Závěr	25
Zdroje	26
Technologie	26
Využitý kód	26
Ostatní využité zdroje	26

Instalace

1. Naklonování repozitáře z githubu:

<https://github.com/fadexmusic/sa.me>

2. Instalace NodeJS

<https://nodejs.org/en/>

4. Instalace MongoDB (community server)

<https://www.mongodb.com/download-center?jmp=nav#community>

4. Instalace Angular

```
npm install -g @angular/cli
```

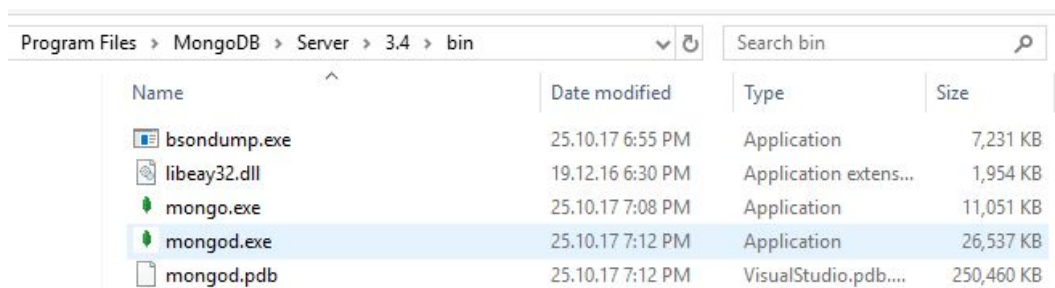
5. Instalace modulů frontendu (ve složce same-fe)

```
npm install
```

6. Instalace modulů backendu (ve složce same-be)

```
npm install
```

7. Spuštění databáze (mongod.exe v místě instalace)



Name	Date modified	Type	Size
bsondump.exe	25.10.17 6:55 PM	Application	7,231 KB
libeay32.dll	19.12.16 6:30 PM	Application extens...	1,954 KB
mongo.exe	25.10.17 7:08 PM	Application	11,051 KB
mongod.exe	25.10.17 7:12 PM	Application	26,537 KB
mongod.pdb	25.10.17 7:12 PM	VisualStudio.pdb....	250,460 KB

8. Spuštění frontendu (ve složce same-fe)

```
ng serve
```

9. Spuštění backendu (ve složce same-be)

```
node index.js
```

10. Otevřít aplikaci - ve webovém prohlížeči na adrese
`http://localhost:4200`

Úvod

Rozhodl jsem se, že jako svůj maturitní projekt vytvořím sociální síť, jejíž koncept jsem už měl v hlavě nějakou dobu. Nápad je založen na podobných existujících sociálních sítích - například si bere hodně inspirace z Instagramu a jeho jednoduchosti. Každý uživatel má svou vlastní stránku, kam může přidávat textové a obrázkové příspěvky. Ostatní uživatelé mohou sledovat stránky jednotlivých uživatelů, čímž se jim nové příspěvky všech sledovaných uživatelů zobrazí na “news feedu”. Projekt by se také dal přirovnat k Twitteru, ale Same je na rozdíl od něj na používání jednodušší, což ubírá na náročnosti pro uživatele, v čemž vidím výhodu. Hlavní myšlenka, která projekt odlišuje od stávajících sítí je založená na reakcích na příspěvky. Místo klasického “liku” uživatel reaguje pomocí “same”, kterým dává najevo, že se cítí stejně nebo/či/anebo smýšlí stejně jako to vyjádřil autor v příspěvku. V dnešní době jsou všude po internetu populární právě různé obrázky/texty, se kterými se člověk může ztotožnit, a přesně na tom je stránka postavena. Autor může sledovat, kolik lidí se ztotožňuje s jeho názory, problémy, vtipy atd..

Jako úkol jsem tedy před sebou měl vypracovat sociální síť a to jak frontend, backend, tak i databázi.

Nejprve jsem měl promyslet všechny funkce, na základě kterých jsem vybral všechny technologie, dále udělat vizuální stránku, následně vytvořit design a schémata databáze a komunikace se serverem a frontendem, a na konec vše převést do kódu.

Hlavní část

Funkce, design a ukázka

Nejdříve jsem si musel definovat funkce, které od sociální sítě chci a jak je zpracuji. Rozhodl jsem pro velmi minimalistický design. Čistý a nepřelácaný design je v dnešní době velmi populární a myslím si, že i v kontextu Samu funguje velmi dobře, jelikož uživateli pozornost zaměřuje na obsah, na který dává stránka důraz. Rozložení stránky je udělané tak, aby fungovalo dobře i na telefonu a aby bylo jednoduché na navigaci. Jde o sloupec uprostřed, ve kterém se uživatel může pohybovat v podstatě jen nahoru a dolů, aniž by musel někam klikat.

První skupina funkcí je určitá práce s uživatelskými účty. Uživatel pro registraci účtu vyplní formulář, do kterého vyplní základní informace.

The screenshot displays a web interface with a dark header bar. On the left of the header is the text "same", in the center is a search input field with the placeholder "search", and on the right is a "log in" link. Below the header, the page content is centered. It starts with the heading "Register". Underneath are five text input fields labeled "username", "email", "avatar url", "password", and "confirm password". At the bottom left of the form is a "register" button, and at the bottom right is a "log in" link. A thin horizontal line separates the form area from the footer, which contains the copyright notice "© 2017 Vitek Paulik".

Všechny formuláře v aplikaci jsou reaktivní a upozorňují uživatele na jakékoliv chyby (email/uživatelské jméno je zabrané, políčko je povinné, hesla se neshodují atd.)

Register

username taken

invalid email

confirm password required

[log in](#)

Když se uživatel registruje, přijde mu email, který registraci potvrzuje.



Dále je přístupný přihlašovací formulář.

same

search

log in

Log in

username

password

log in

[forgot password?](#) [register](#)

© 2017 Vitek Paulík

Když uživatel zapomene heslo, může zadat svůj email a přijde mu nově vygenerované heslo.

Reset Password

your email

reset password



sa.me.socialnetwork komu: mně ↕


Your password has been reset to **8ld5SBdK**, please log in and change your password.

Po přihlášení se uživatel zobrazí “news feed”, kde vidí příspěvky lidí, které sleduje.

same


search

+ | 8 | 1 | 0


 **vratislav** is like 3 minutes ago

už aby byly prázdniny

1 person is the same unsame

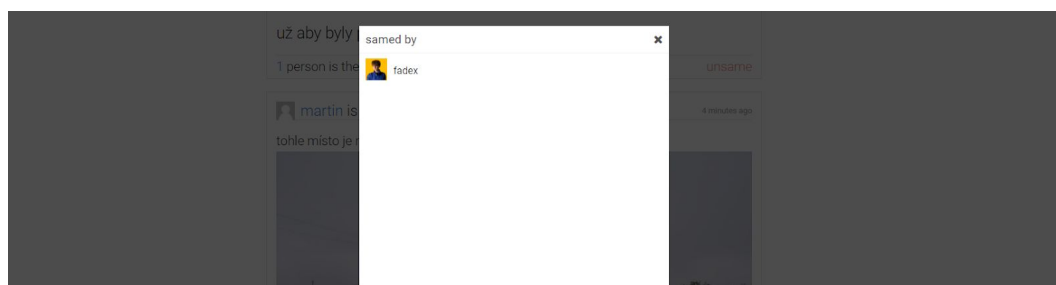
 **martin** is like 4 minutes ago

tohle místo je nejhorší



0 people are the same same


Na příspěvky může reagovat a prohlížet si, kdo na ně reagoval.



Uživatel si může vyhledávat existující uživatele pomocí real-time měnícího se políčka.




Na stránce uživatele si může prohlížet příspěvky uživatele, kdo uživatele sleduje a koho sleduje.



martin / 1 samer / 0 saming


jsem smutný

following

 martin is like


8 minutes ago

tohle místo je nejhorší



0 people are the same

same

 martin is like

11 minutes ago

Nesnáším politiku

Uživatel má na stránce svého účtu možnost upravovat svůj účet pomocí formuláře (měnit informace, heslo, smazat účet). Všechny změny musí být potvrzeny stávajícím heslem.

same

search

+ | 8 | 0 | 0

Edit profile

fadex

vitekpaulik@gmail.com

bio

https://i1.sndcdn.com/avatars-000369519662-mjl5x6-t500x500

type your password to confirm

edit

Edit password

new password

confirm new password

type your old password to confirm

edit password

Delete account

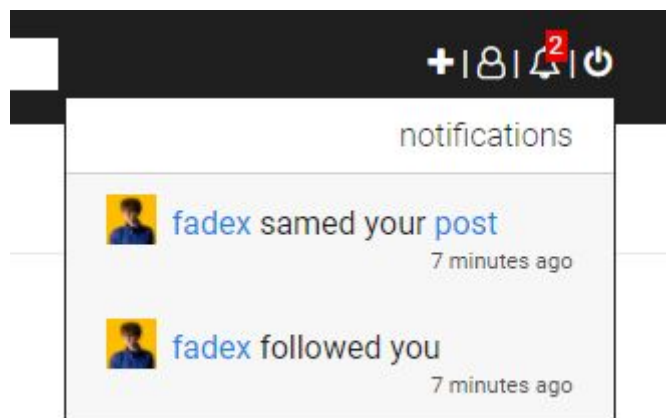
type your password to confirm

delete account

© 2017 Vitek Paulík

Přihlášený uživatel se může skrz menu v pravém horním rohu přesouvat po stránce, zobrazit notifikace, odhlásit se a přidat nový příspěvek.

notifikace



přidávání příspěvku.

Add new post

text image

dnes se cítím fakt skvěle

post

Uživatel také může mazat své příspěvky.



Všechny akce na stránce mají také vizuální odezvu - když uživatel přidá příspěvek, nebo se přihlásí, tak se v pravém dolním rohu zobrazí popup.



Všechny tyto funkce dávají dohromady kompletní aplikaci, ve které se uživatel neztratí a zvládne udělat vše, co potřebuje.

Technologie

Na základně zvolení formy webové aplikace jsem si dále začal vybírat technologie, které použiji.

Vzhledem k tomu, že frontend ovládám dobře, tak jsem měl ve frontendové technologii jasno. Použil jsem framework vytvořený pro single-page aplikace *Angular 5*. Tuto technologii jsem zvolil, protože je přizpůsobena pro vývoj dynamických webových aplikací a zahrnuje v sobě škálu funkcí, které jsou k vývoji potřeba. Angular umožňuje jednoduchou implementaci reaktivních formulářů, dynamického vykreslování, komunikace se serverem atd..

Další na řadě byl výběr způsobu, který zařídí komunikaci mezi frontendem a serverem. Zvolil jsem klasický REST (Representational state transfer), který je založený na jednorázových HTTP requestech GET, POST, PUT, DELETE. Tento systém je vhodný, jelikož na stránce nejsou žádné “kontinuální” requesty, na které je vhodnější použít třeba websockety, a navíc velmi dobře funguje s *Angular*em.

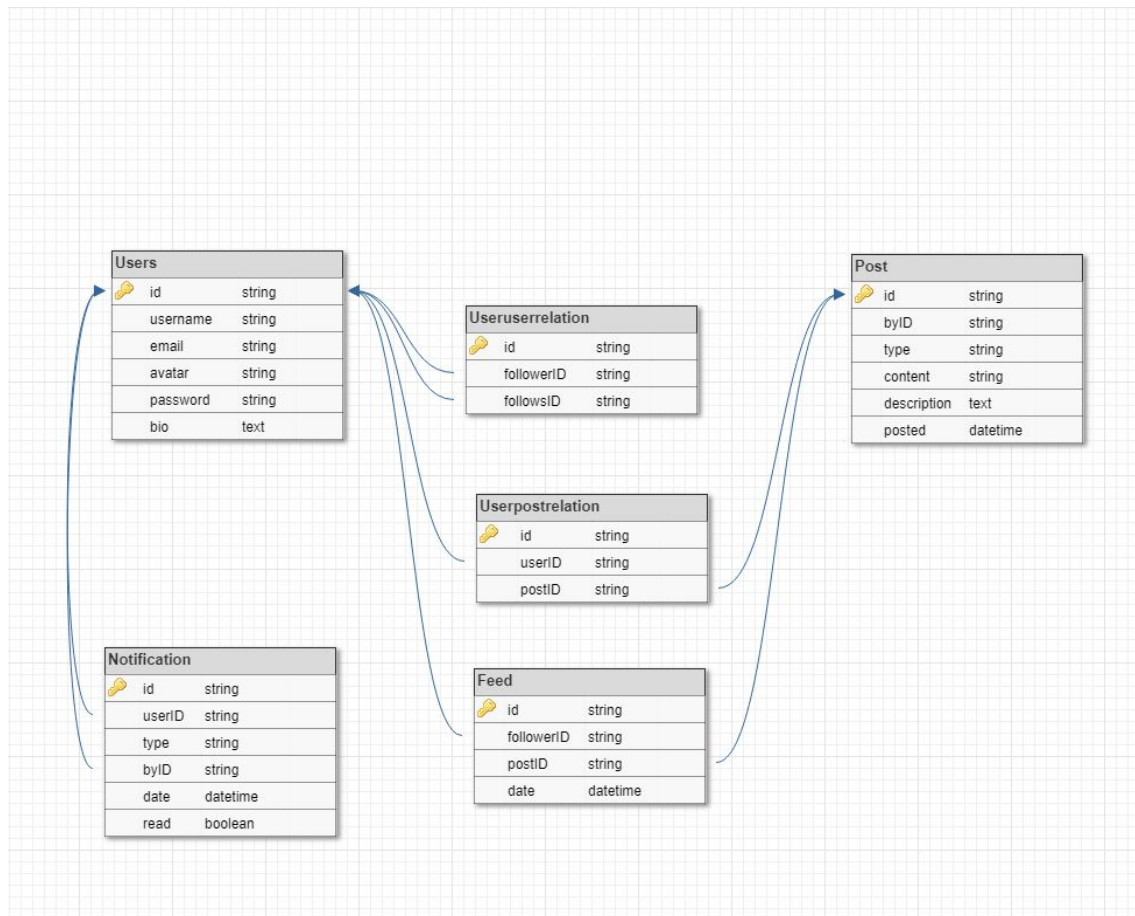
Vzhledem k tomu, že jsem na back-endu nikdy moc nepracoval, zvolil jsem pro něj front endu blízký NodeJS v kombinaci s ExpressJS. Oba jsou totiž založeny na frontendovém jazyku Javascriptu. Express je navíc vytvořený právě pro REST, které jsem zvolil pro komunikaci mezi serverem a frontendem. NodeJS nabízí velkou škálu pluginů, které přináší funkce potřebné k vyvinutí serveru. Použil jsem *nodemailer*, který zařizuje posílání emailů uživatelům, *bcrypt*, který je využit pro enkryptování hesel a následné přihlašování. Následuje JWT, využitý pro generování JSON Web Tokenů, pomocí kterých zařizují bezpečnou komunikaci mezi přihlášeným uživatelem a serverem. Dále *cors*, který řeší problém s cross-origin

hlavičkami a v neposlední řadě async, který zjednodušuje asynchronní programování v NodeJS.

Jako poslední jsem musel zvolit databázovou technologii a rozhodl jsem se pro MongoDB, které je opět přívětivý pro lidi, co nemají takovou zkušenost s backendovým programováním, a přesto nabízí všechny důležité možnosti. Práci s MongoDB jsem ještě doplnil Nodovým pluginem mongoose, který zjednodušuje organizaci dokumentů v databázi, pomocí schémat (které známe například z SQL) a umožňuje jednoduché nastavení indexování dokumentů v tabulce podle sloupců, které usnadňuje vyhledávání v databázi.

Když jsem měl mezi technologiemi tak nějak jasno, mohl jsem se vrhnout na design databáze.

Databáze



Z funkcí a následném naplňování schéma databáze vyšlo 6 tabulek. První tabulka “Users” v sobě uchovává informace o uživateli.

Každý uživatel má unikátní identifikační číslo, uživatelské jméno, email, profilový obrázek, heslo a bio (krátký text, který uživatel může o sobě napsat).

Hned vedle uživatele je tabulka Useruserrelation, která propojuje dvě uživatelské řádky. Vytvoří se když uživatel začne sledovat jiného uživatele. Dřív v sobě identifikační číslo sledujícího a sledovaného.

Další důležitá tabulka je tabulka “Post”, která v sobě drží informace o příspěvcích. Má opět unikátní identifikační číslo, typ (který určuje, jestli se jedná o obrázek, nebo text), content (buďto odkaz na obrázek, nebo samotný text), description (popis, který může, nebo nemusí uživatel napsat k obrázku) a datum, kdy byl příspěvek přidán.

Další tabulka váže uživatele a příspěvek. Když uživatel vytvoří příspěvek, vytvoří se řádek v tabulce Userpostrelation, který nese identifikační číslo uživatele a příspěvku.

Dále je v databázi tabulka “Feed”, která v sobě drží identifikační číslo sledujícího, identifikační číslo příspěvku a datum vytvoření feedu. Tato tabulka je využita pro vytváření feedů z příspěvků sledovaných uživatelů pro sledující.

Poslední tabulka je tabulka “Notifications”, která je využita pro upozorňování uživatelů na určité akce (když je někdo začne sledovat, nebo jim dá “same” na příspěvek). Drží v sobě tedy identifikační číslo uživatele, pro kterého je určena, typ (jestli se jedná o sledování, nebo same), dále identifikační číslo uživatele, kterým byla vyvolána (sledujícím, nebo reagujícím), datum, kdy byla vytvořena a boolean, který zobrazuje zda byla uživatelem zobrazena, nebo ne.

Frontend

Frontend je postavený na frameworku Angular 5. Vzhledem k tomu, že jde o systém postavený pro single-page aplikace, hlavní stavební buňkou je tzv. router. Pokaždé, když kliknu na nějaký odkaz, router mě přesměruje na daný modul (login, feed, user..).

Samotný layout všech modulů je napsaný v HTML. HTML je obohacené o logiku modulů. Které elementy se zobrazují může být dané nějakou proměnnou ze scriptu modulu.

```
1 <h2>Log in</h2>
2 <form [formGroup]="loginForm" (ngSubmit)="login()" id="loginform">
3   <input [ngClass]="{invalid: !valid.username.valid}" type="text" placeholder="username" [formControl]="loginForm.controls['username']">
4   <div class="invalid" *ngIf="!valid.username.valid">{{valid.username.message}}</div>
5   <input [ngClass]="{invalid: !valid.password.valid}" type="password" placeholder="password" [formControl]="loginForm.controls['password']">
6   <div class="invalid" *ngIf="!valid.password.valid">{{valid.password.message}}</div>
7   <button form="loginform">log in</button>
8   <!-- <a [routerLink]="['/register']" id="switch">forgot password</a> -->
9   <a [routerLink]="['/register']" id="switch">register</a>
10  <a [routerLink]="['/reset']" id="switch" style="margin-right: 8px">forgot password? </a>
11 </form>
12
```

nastylování modulů je uděláno pomocí SCSS.

```
1 @import '../assets/variables';
2 $info-height: 50px;
3 $info-height-big: 80px;
4 #head {
5   border-bottom: 1px solid lighten($main, 70%);
6   width: 100%;
7   height: $info-height;
8 }
9
10 #user-info {
11   width: 100%;
12   height: $info-height;
13   position: relative;
14   div {
15     display: inline-block;
16     img {
17       width: $info-height;
18       height: $info-height;
19     }
20   }
21   #avatar {
22     height: $info-height;
23   }
24   #info {
25     height: $info-height;
26     line-height: $info-height;
27     vertical-align: top;
28     div {
29       display: inline-block;
30     }
31     #username {
32       margin-left: 8px;
33       display: inline-block;
34       color: $main;
35     }
36     #followers {
37       display: inline-block;
38       color: lighten($main, 30%);
39     }
40   }
41 }
```

Logika modulů se nepíše v čistém *Javascriptu*, ale v jeho vylepšené verzi *Typescriptu*. Zároveň je typescriptový soubor propojen s HTML a zprostředkovává různé proměnné důležité pro vykreslování.

ukázka modulu pro vytváření příspěvku

```
13 export class NewPostComponent implements OnInit {
14
15     type: string = "text";
16
17     textForm: FormGroup;
18     textValid = {
19         content: { valid: true, message: '' }
20     }
21
22     imageForm: FormGroup;
23     imageValid = {
24         content: { valid: true, message: '' },
25         description: { valid: true, message: '' }
26     };
27     constructor(private router: Router, private fb: FormBuilder, private ns: NewPostService, private nos: NotificationService) {
28         this.textForm = fb.group({
29             content: ['', Validators.required]
30         });
31         this.imageForm = fb.group({
32             content: ['', Validators.required],
33             description: ['']
34         });
35     }
36
37     ngOnInit() {
38         this.textForm.get('content').valueChanges.subscribe(value => {
39             if (value != '' || value != null) {
40                 this.textValid.content.valid = true;
41                 this.textValid.content.message = '';
42             }
43         });
44         this.imageForm.get('content').valueChanges.subscribe(value => {
```

Serverové requesty jsou vytvořeny pomocí tzv. servisů, které v sobě drží jejich logiku a následně se tážají serveru. Jsou injektovány do hlavních skriptů modulů, ze kterých jsou funkce servisu vyvolávány.

příklad servisu na uživatelskou stránku.

```
1 import { Observable } from 'rxjs';
2 import { AuthHttp } from 'angular2-jwt/angular2-jwt';
3 import { serverAddress } from '../app.config';
4 import { Http, RequestOptions, Headers } from 'angular/http';
5 import { Injectable } from '@angular/core';
6
7 @Injectable()
8 export class UserService {
9
10     uri: string = "user/";
11     options: RequestOptions;
12
13     constructor(private http: Http, private ahttp: AuthHttp) {
14         let headers: Headers = new Headers();
15         headers.append('Content-Type', 'application/json');
16         this.options = new RequestOptions({ headers: headers });
17     }
18
19     getUser(name: string): Observable<any> {
20         return this.http.get(serverAddress + this.uri + name + '?by=name').map(res => res.json());
21     }
22     getUserById(id: string): Observable<any> {
23         return this.http.get(serverAddress + this.uri + id + '?by=id').map(res => res.json());
24     }
25     getPosts(id: string, offset: number, limit: number): Observable<any> {
26         return this.http.get(serverAddress + 'posts/' + id + '?offset=' + offset + '&limit=' + limit).map(res => res.json());
27     }
28     getPost(id: string): Observable<any> {
29         return this.http.get(serverAddress + 'post/' + id).map(res => res.json());
30     }
31     followerCount(id: string): Observable<any> {
32         return this.http.get(serverAddress + 'followers/' + id + '/count').map(res => res.text());
33     }
34     followingCount(id: string): Observable<any> {
35         return this.http.get(serverAddress + 'following/' + id + '/count').map(res => res.text());
36     }
37     changeFollow(id: string): Observable<any> {
38         return this.ahttp.put(serverAddress + 'follow/' + id, null, this.options);
39     }
40 }
```

Postup pro vytvoření všech modulů byl téměř stejný, jen se lišil logikou specifickou pro určité funkce modulu.

Velmi dobře se mi pracovalo s formuláři, které má skvěle vyřešený Angular. Na políčka lze přidávat tzv. Validatory, které můžou kontrolovat různé vstupy (je text moc dlouhý? je políčko vyplněné? je zadaný email správně?). Jinak Typescript také velmi pomáhá a vylepšuje zastaralý Javascript

Ve frontendu jsem také použil pár externích modulů. AngularJWT na práci s json web tokeny na frontendu. Ng-click-outside na uživatelský vstup.

Backend

Server má za úkol přijímat požadavky z frontendu, vytahovat data z databáze a následně je posílat zpět na frontend. Frontend se dotáže na nějaké informace, nebo iniciuje změnu. Backend vyhledá relevantní info v databázi a posílá chyby, a nebo získaná data, pomocí objektů a pošle je uživateli.

Při vytváření backendu jsem narazil na zádrhel, a to způsob skládání “news feedu”. Zjistil jsem, že jsou dvě zažité metody - tzv. “push” a “pull”. Jde o to, při jaké akci se má finální feed skládat. V metodě push, kterou jsem nakonec zvolil, se vytváří řádky do tabulky “Feed” pokaždé, když uživatel vytvoří příspěvek. Vytvoří se zvláštní řádek pro každého sledujícího uživatele. U metody pull se feed vytváří pokaždé, když ho sledující načte, tak, že se vytáhnou všechny řádky z tabulky Userpostrelation uživatelů, které sleduje, srovnají se podle data a následně se podle identifikačních čísel vezmou samotné příspěvky a vykreslí se na stránku. Metoda pull mi přišla mnohem náročnější jak na server, tak časově na generování stránky feedu, proto jsem se nakonec rozhodl pro metodu push, u které nevadí, že vytváření feed tabulek trvá déle, protože na ně nikdo přímo nečeká.

získávání notifikací

```
990 /* NOTIFICATIONS */
991 app.route('/notifications')
992 .get((req, res) => {
993   if (req.headers.authorization) {
994     let auth = req.headers.authorization.split(' ');
995     if (auth[0] == "Bearer") {
996       let user = jwt.decode(auth[1], config.secret);
997       Notification.find({
998         userID: user.id
999       }).limit(parseInt(req.query.limit)).sort({
1000         date: -1
1001       }).lean().exec((err, notifications) => {
1002         if (err) console.log(err);
1003         if (notifications.length > 0) {
1004           async.map(notifications, (notification, callback) => {
1005             User.findById(notification.byID, (err, user) => {
1006               if (err) console.log(err);
1007               let n = notification;
1008               n.byUsername = user.username;
1009               n.byAvatar = user.avatar;
1010               callback(null, n);
1011             });
1012           }, (err, notifs) => {
1013             if (err) console.log(err);
1014             res.status(200).json(notifs);
1015           })
1016         } else {
1017           res.status(404).send('no notifs found');
1018         }
1019       })
1020     }
1021   }
1022 });
```

Programování endpointů se v backendu se v podstatě také tolik neliší, jen občas je potřeba kontrolovat validitu dat, nějak data zpracovat, nebo dobře zformátovat vyhledávání.

Komunikace mezi serverem a frontendem

Aby byla komunikace mezi přihlášeným uživatelem bezpečná, rozhodl jsem se využít tzv. json web tokenů. Pokaždé, když se uživatel přihlásí, na serveru se vygeneruje tento token a pošle se uživateli, kterému se uloží do lokální paměti. Pokaždé když uživatel udělá nějaký request spojený se změnou jeho účtu, pošle se i token, který se na serveru ověří a následně se provede akce.

Komunikace je udělána pomocí tzv. endpointů, což je adresa určité akce na kterou se uživatel táže. Například když chci získat seznam lidí co mě sleduje, udělám GET request na endpoint /follows/:id, kde id je uživatelské identifikační číslo.

Závěr

Projekt se dle mého velmi vydařil a naučil jsem se při něm spoustu věcí. První byla změna mého normální postupu, která spočívala v tom, že jsem k práci přistupoval velmi systematicky a chronologicky - projekt jsem si rozvrhl a nejdříve jsem zpracoval všechny důležité části - naplánoval si funkce, podle nich udělal celkový vzhled a design databáze. Dále jsem si promyslel, které requesty budou k funkcím potřeba a pak jsem aplikaci už “jen” naprogramoval. Tento postup velmi usnadnil organizaci procesu vytváření/procesu tvorby webu.

Projekt mě také naučil spoustu novým praktickým informacím. Byl pro mě něco úplně nového a na jiné úrovni náročnosti, než jsem doposud dělal. Vzhledem k tomu, že jsem do této doby nikdy nepracoval s backendem, naučil jsem se spoustu velice užitečných informací ohledně jeho logiky, práce s databází a komunikace mezi uživatelem a server. Díky těmto informacím jsem získal hodnotné zkušenosti, které jsem již dále využil a určitě ještě využiji.

Dále jsem se celkem dobře naučil pracovat s verzovacím systémem GitHub, který je v dnešní době velmi důležitý a určitě se s ním setkám v budoucí práci.

Všechny funkce, které jsem chtěl udělat a popsal na začátku, jsem splnil a výsledkem je funkční aplikace, kterou může používat spousta uživatelů.

Zdroje

Technologie

NodeJS: <https://nodejs.org/en/>

ExpressJS: <https://expressjs.com/>

bcrypt: <https://www.npmjs.com/package/bcrypt>

jwt-simple: <https://www.npmjs.com/package/jwt-simple>

cors: <https://www.npmjs.com/package/cors>

nodemailer: <https://nodemailer.com/about/>

async: <https://caolan.github.io/async/>

bodyParser: <https://www.npmjs.com/package/body-parser-json>

Angular 5: <https://angular.io/>

AngularJWT: <https://github.com/auth0/angular-jwt/>

ng-click-outside: <https://www.npmjs.com/package/ng-click-outside>

MongoDB: <https://www.mongodb.com/>

Mongoose: <http://mongoosejs.com/>

Využitý kód

generování náhodného stringu:

<https://stackoverflow.com/questions/1349404/generate-random-string-characters-in-javascript>

kontrolování obrázků:

<https://stackoverflow.com/questions/9714525/javascript-image-url-verify>

Ostatní využití zdroje

stackoverflow: <https://stackoverflow.com/>