

Gymnázium, Praha 6, Arabská 14

Programování



Ročníková práce

Květen 2019

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

Týmová ročníková práce

Předmět: Programování

Téma: Internetové fórum

Autor: Čech David, Patzák Dominik, Petrů Anna, Korladinov Viktor

Třída: 3.E

Školní rok: 2018/2019

Vedoucí práce: Lána Jan

Třídní učitel: Mgr. Urzová Jana

Prohlašujeme, že jsme jedinými autory tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů udělujeme bezúplatně škole Gymnázium, Praha 6, Arabská14 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 10. května 2019

Anotace

Tato ročníková práce se zabývá naším postupným naprogramováním diskuzního prostoru pro studenty nazvaného Gyarab Forum. Zabývá se strukturou našeho programu a jednoduchým popsáním všech komponent a funkcí. Gyarab Forum má představovat prostor, který by využívali studenti ke sdílení znalostí, otázek, poznámek či řešení různých školních problémů. To vše formou příspěvků zorganizovaných pod určitou tematikou. Tyto příspěvky může uživatel okomentovat nebo ohodnotit pomocí známého lajk („To se mi líbí“) nebo dislajk („To se mi nelíbí“).

Abstract

This course work is about the gradual programming of an online space Gyarab Forum, where students can participate in discussions. The paper deals with the structure of the program and basic description of its components and functions. This program should become a place for students to share their knowledge, questions, notes or solutions to schoolwork. Everything is displayed in the form of posts organised under a particular theme. User can comment on the posts or give them a like or a dislike.

Obsah

1. Zadání	3
2. Úvod	4
3. Použité technologie	5
3. 1. IntelliJ IDEA.....	5
3. 2. GitHub	5
3. 3. React	6
3. 4. Redux	8
4. Instalace	10
4. 1. Frontend.....	10
4. 2. Backend.....	10
5. Konfigurace	11
5. 1. Spouštění.....	11
6. Backend	13
6. 1. Struktura Projektu	13
6. 2. Domain	14
6. 3. Repozitář	15
6. 4. Service a Rest	16
7. Frontend	17
7. 1. Struktura projektu	17

7. 1. Action-creators a reducers.....	19
7. 1. a) AuthActionCreator	19
7. 1. b) CommentActionCreator	19
7. 1. c) ForumActionCreator	20
7. 1. d) PostActionCreator	20
7. 1. e) ForumReducer a rootReducer	21
7. 2. Samotné komponenty	22
7. 2. a) Navigation	22
7. 2. b) AccountInfo	23
7. 2. c) LilPost	24
7. 2. d) RenderLilPosts	25
7. 2. e) BigPost.....	26
7. 2. f) Comment	27
7. 2. g) CreatePost a ReactQuill.....	27
7. 2. h) Login.....	29
7. 2. i) Register	29
7. 2. j) ForumRouter	30
8. Závěr	31
9. Zdroje.....	32

1. Zadání

Cílem ročníkové práce je vytvořit internetové fórum s tematikou Gymnázia Arabská. Na tomto fóru se uživatelé můžou zaregistrovat, poté se přihlásit na svůj účet a přidávat své příspěvky. Jako příspěvek půjde přidat text nebo obrázek. Uživatelé budou také moci prohlížet a komentovat příspěvky jiných uživatelů, nebo na ně reagovat kliknutím na like nebo dislike, a tím vyjádřit beze slov zda se jim líbí či nelíbí. Fórum bude rozděleno do několika větví podle témat příspěvků, jako například maturita, povinná četba, matematická olympiáda a další. Uživateli umožňuje pohyb mezi těmito větvemi postranní panel a stromová struktura. Různým lidem budou moci být přiděleny role a práva na úpravu stránek. Stránka by měla mít jak mobilní, tak i standardní verzi v internetových prohlížečích.

2. Úvod

Dnes již prakticky každý mladý člověk vlastní jeden, nebo i více účtů na různých sociálních mediích, a ví jak se v nich pohybovat. Klasicky je to formou příspěvků s obrázky nebo texty. Tento typ komunikace se stala velmi populární. Nejznámější sociální média jsou v dnešní době Facebook, Instagram, Tumblr nebo například Twitter. Na nich ale lidé sdílejí své zážitky či zajímavé myšlenky. Je k nim v našich myslích vázaná zábava. Naším programem se pokoušíme zavést tuto variantu sdílení i ke školním tématům. Má se stát místem, kde si lidé mohou sdílet své školní poznámky, úkoly a řešení formou příspěvků, kterou znají z populárních aplikací.

Tato práce pojednává o tom, jak funguje program, ale také věnuje tomu, jak uživatel tj. diskutující může tuto aplikaci použít. Aplikaci jsme tedy vytvářeli s ohledem na koncového uživatele. Hlavním cíle bylo, aby aplikace byla jednoduše ovládatelná, intuitivní, přehledná a logická. Uživatel si v naší aplikaci vytvoří účet se jménem, emailem a heslem, do kterého se následně přihlásí. Ze svého účtu může přidávat příspěvky s obrázkem anebo s různě upraveným a dlouhým textem. Lze také otevřít příspěvek jiného uživatele, a vyjádřit se k němu komentářem, Likem („To se mi líbí“) nebo Dislikem („To se mi nelíbí“). Při načtení hlavní stránky aplikace je automaticky nabídnut uživateli jeden příspěvek. Příspěvky jsou seskupeny podle jejich témat do různých fór. Největší problém se kterým jsme se potýkali, byla nedostatečná znalost použitých jazyků. Museli jsme se tedy seznámit do detailu s daným jazykem a teprve pak začít programovat.

3. Použité technologie

3. 1. IntelliJ IDEA

Jako vývojové prostředí (IDE) jsme zvolili IntelliJ IDEA. Důvodů pro naši volbu bylo několik. Tím hlavním důvodem byla podpora aplikačních rámců (frameworků), se kterými jsme chtěli pracovat. Pro frontend to byly React a Redux a pro backend Spring a Hibernate. Další příjemnou vlastností IntelliJ IDEA je zabudovaný VCS (Version Control System) přímo ve vývojovém prostředí. V našem případě to byl Git, ale IntelliJ podporuje i jiné VCS. Toto zabudování nám ulehčuje standardní operace pro spravování verzí projektů na několika zařízeních (push, pull, commit). Také přehledně ukazuje chyby, které se přihodily během vykonávání těchto operací, změny provedeny v projektu od posleného pullu nebo změny, které způsobil poslední pull. V poslední řadě to byly věci, jež by měly být pro každé vývojové prostředí samozřejmostí, např.: upozornění na chybnou syntaxi, nabídka doplnění názvů funkcí a proměnných, snadný pohyb mezi soubory apod.

3. 2. GitHub

Pro spravování verzí projektů jsme společně s VCS ve vývojovém prostředí používali GitHub, webovou službu, která poskytuje bezplatný webhosting pro open-source projekty. Náš projekt jsme museli rozdělit do dvou repozitářů na frontend a backend. Na GitHubu se oba naše repozitáře nacházejí v organizaci naší školy ***<https://github.com/gyarab>*** a nazývají se ***forum-frontend*** a ***forum-backend***.

3.3. React

React je framework, který kombinuje JavaScript a HTML (jsx). Je vyvíjen Facebookem a komunitou nezávislých developerů a firem, přičemž je open source od roku 2013 (další známou stránkou používající React je Instagram). Základním stavebním blokem v Reactu jsou komponenty (Component), které reprezentují různé prvky webových stránek. Důležitou součástí komponent je jejich state (stav), do kterého se ukládají data, jež má komponenta zobrazovat. Dále mohou komponenty dostávat data od ostatních komponent nebo jiných částí projektu, tato data se nazývají props.

Každá komponenta musí mít funkci render, jež vrátí HTML, do kterého můžeme vložit námi uložená data pomocí složených závorek (viz ob. 1). Komponenty mají vlastní životní cyklus (viz ob. 2). Tento životní cyklus můžeme v komponentách využívat, jelikož existuje několik funkcí, jež se spustí v patřičných fázích životního cyklu, např.: funkce `componentDidMount` se spustí, když se komponenta poprvé vyrenderuje na stránce, a funkce `componentDidUpdate` se spustí poté, co dojde ke změně dat v komponentě (komponenta dostala nové props, nebo byl změněn její state), kvůli čemuž je nutno ji znovu vyrenderovat. Pro React jsme se rozhodli, protože roste poptávka po vývojářích v tomto frameworku a také protože se jedná o rychle pracující framework, jehož propojení JavaScriptu a HTML je v kódu intuitivní a příjemné.

```

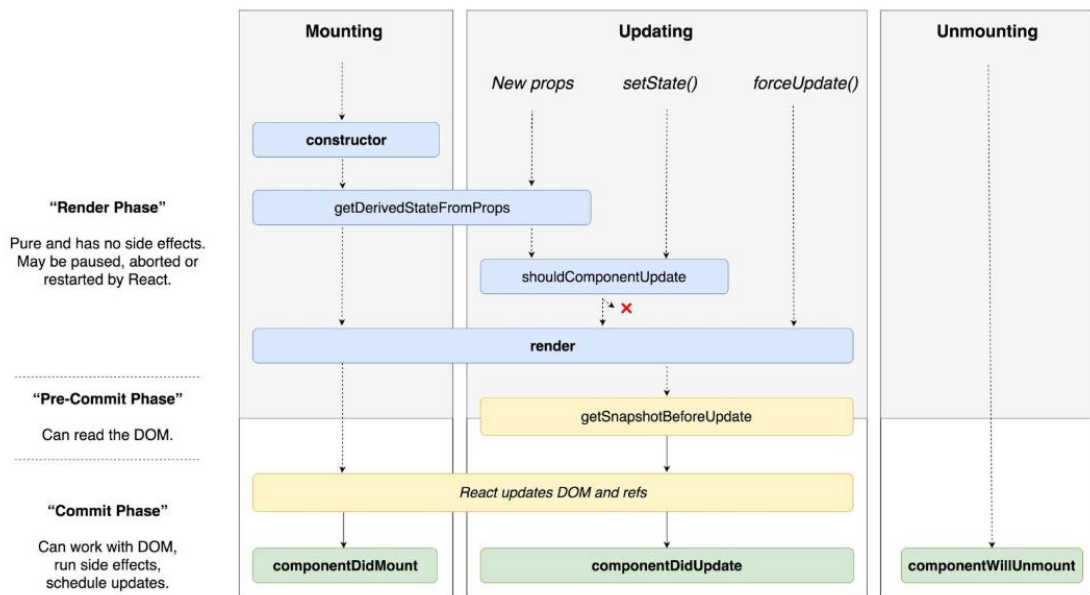
class AccountInfo extends Component {

  constructor() {
    super();
    this.state = {
      userJson: JSON.parse(atob(localStorage.getItem("auth").split(".")[1])),
    };
  }

  render() {
    return (
      <div className="accountinfo">
        { /*Shows the email if logged in*/ }
        <div className="username-wrapper">You are logged in as {this.state.userJson.sub}</div>
      </div>
    );
  }
}

```

Obr. 1: React komponenta



Obr. 2: Lifecycle komponenty

3. 4. Redux

Společně s Reactem používáme Redux, což je také JavaScriptový framework. Hlavní myšlenkou Reduxu je vytvoření uložště (store) dat pro celou aplikaci, ke kterému se může každá komponenta připojit a požádat si o patřičná data. Zároveň zajišťuje získávání dat z api. Redux se skládá v našem projektu ze tří částí: store.js, action-creators a reducers. V store.js Redux sám vytváří již dříve zmiňované uložště. Reducers mají jako jediný přístup k datům v uložšti. Pokud tato data chceme měnit, měníme je přes reducers (viz ob. 3), nikdy je neměníme přímo. V action-creators jsou funkce (viz ob. 4), které získávají data z api a posílají je reducers, které tato data ukládají do uložště. Uložště je připojeno k celé aplikaci pomocí Provideru (React-Redux komponenta) v App.js. Komponentám připojeným k uložšti říkáme pomocí Reduxu (mapStateToProps a mapDispatchToProps), které funkce z action-creators a která data z uložště chceme mít k dispozici (viz ob. 5). Redux nám poté tyto funkce a data pošle jako props, jež můžeme následně v komponentách využívat.

```
const initialState = {
  status: false,
  storage: [],
  posts: [],
  arrayOfForums: [],
  post: "",
  logged: false,
  updatedPost: {post: {}, attitudeDto: {}},
  updatedComment: {comment:{},attitudeDto: {}},
  comments: ""
};

export default function (state = initialState, action) {
  switch (action.type) {
    case 'SEARCH_FORUM_BY_NAME':
      return {
        ...state,
        arrayOfForums: action.payload
      };
    case 'FETCH_ALL_FORUM_NAMES':
      return {
        ...state,
        storage: action.payload
      };
  }
}
```

Obr. 3: Forum reducers

```

export const searchForumByName = (name) => dispatch => {
  fetch(url+'/forum/search/' + name)
    .then(response => response.json())
    .then(item => dispatch({
      type: 'SEARCH_FORUM_BY_NAME',
      payload: item
    }));
};

```

Obr. 4: Action-reducer funkce

```

RenderLilPosts.propTypes = {
  fetchPosts: PropTypes.func.isRequired,
};

const mapStateToProps = state => ({
  posts: state.forums.posts
});

const mapDispatchToProps = (dispatch) => ({
  fetchPosts: (forumId, forumPage) => {
    dispatch(fetchPosts(forumId, forumPage))
  }
});

export default connect(mapStateToProps, mapDispatchToProps)(RenderLilPosts);

```

Obr. 5: Extrakce dat pomocí funkcí

4. Instalace

4. 1. Frontend

1. K instalaci naší frontend aplikace je potřeba mít nainstalovaný npm (Node Package Manager), sbírku nástrojů pro instalaci a spravování modulů v JavaScriptu, a Git, což je typ Version Control System.
2. Do příkazové řádky s Gitem zadáme příkaz ***git clone https://github.com/gyarab/forum-frontend.git***, který nám vytvoří složku **forum-frontend** se soubory z našeho GitHub repozitáře.
3. Zadáme příkaz ***cd forum-frontend*** a ten nás přenese do námi vytvořené složky, ve které se nachází soubor package.json, s nímž budeme dále operovat.
4. Napíšeme příkaz ***npm install***, pomocí něhož zavoláme Node Package Manager, který nainstaluje všechny JavaScript moduly zaznamenané v souboru package.json.
5. Pro spuštění aplikace vždy používáme příkaz ***npm start***, který za pomoci nainstalovaných modulů nainstaluje naši frontend aplikaci.

4. 2. Backend

Tuto část můžete přeskočit, pokud si zvolíte použít .zip soubor.

Projekt nainstalujeme tak, že do libovolné složky naklonujeme projekt forum-backend, který se nachází na ***https://github.com/gyarab/forum-backend.git*** pomocí ***git clone https://github.com/gyarab/forum-backend.git*** a poté zadáme ***cd forum-backend*** a nakonec ***mvnw clean install***.

5. Konfigurace

Konfigurace programu probíhá editováním dvou souborů(viz ob. 7, 8):

1. `\src\main\resources\application.yml`
2. `\src\main\resources\application-embedded-mariadb.yml`

První soubor konfiguruje samotnou aplikaci, druhý konfiguruje databázi.

- `server.port` definuje port, na kterém aplikace poběží.
- `spring.*` obsahuje nutná konfigurace pro správné fungování systému.
- `lemon.*` konfiguruje knihovnu spring-lemon, kterou používáme při autentizaci uživatelů.

V **`application-embedded-mariadb.yml`** je definován port, url databáze a místo, kde se bude ukládat báze. Tato konfigurace bude použita jen v případě, že spustíme aplikaci s aktivním profilem `embedded-mariadb`.

5. 1. Spouštění

Aplikaci můžeme spustit přes IDE (musíme nastavit správný profil) nebo následovně:

1. **`mvnw clean install`**
2. **`cd target`**
3. **`java -jar -Dspring.profiles.active=embedded-mariadb forum-backend-0.0.1-SNAPSHOT.jar`**

```

server:
  port: 7373
spring:
  liquibase:
    enabled: false
  main:
    allow-bean-definition-overriding: true
  security:
    oauth2:
      client:
        provider:
          facebook:
            user-info-uri: https://graph.facebook.com/me?fields=email,name,verified
        registration:
          google:
            client-id: 1011974249454-6gq0hr0lgqh3cndogqns5r69tkk2nd84.apps.googleusercontent.com
            client-secret: saDA6Cj60wipncFM-hzBD-C6
          facebook:
            client-id: 1234020186718741
            client-secret: 0c0abaf685a83e879e8e48b1167c96ab
  lemon:
    admin:
      username: admin@example.com
      password: admin!

  application-url: http://localhost:3000
  jwt:
    secret: 27924159D8B777B9457245B231294A62
  cors:
    allowed-origins: '*'

```

Obr. 6: Příklad application.yml

```

spring:
  datasource:
    url: "jdbc:mysql://localhost:${mariaDB4j.port}/forum_db?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLeg
  jpa:
    hibernate:
      ddl-auto: update
mariaDB4j:
  port: 3313
  dataDir: /var/db/mariadb
debug: true

```

Obr. 7: Příklad embedded-mariadb.yml

6. Backend

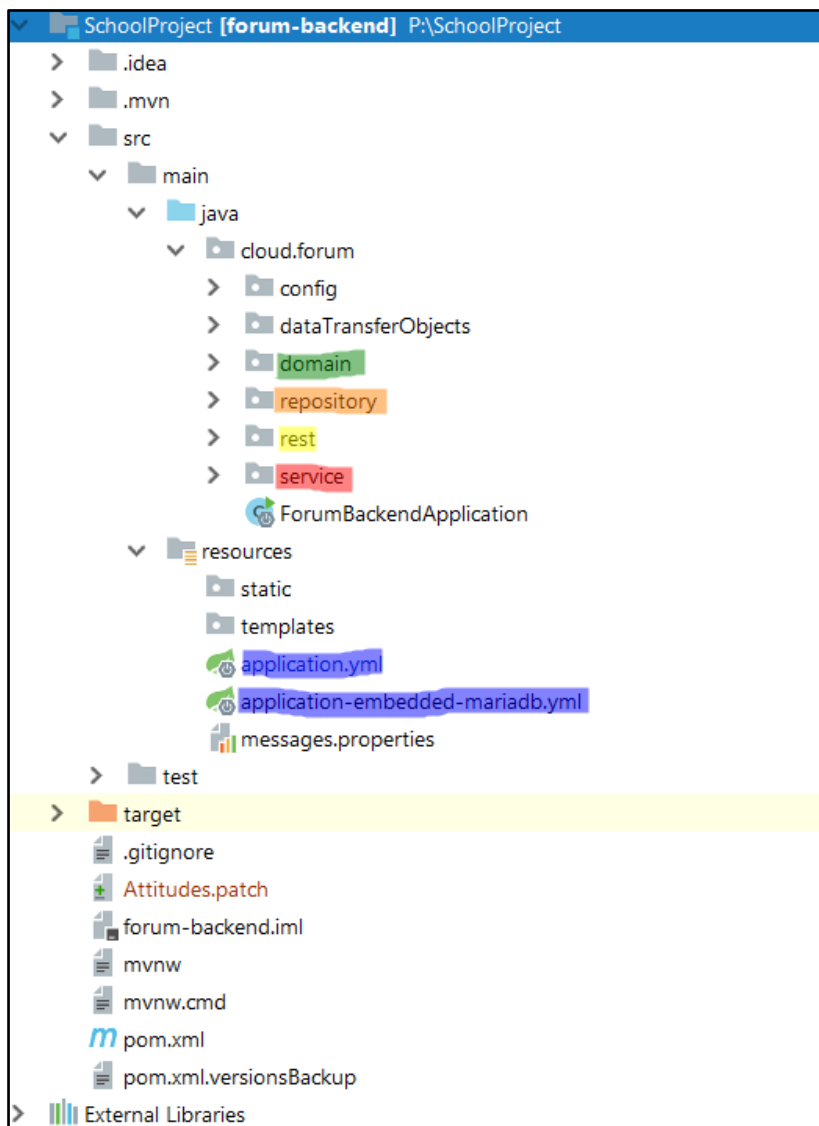
6. 1. Struktura Projektu

Projekt se skládá ze čtyř základních složek/modulů:

- Domain
- Repository
- Service
- Rest

Comment
CommentAttitude
Forum
LemonUser
Post
PostAttitude

Obr. 8 Domain module



Obr. 9: Backend struktura

6.2. Domain

Pod entitou si můžeme představit soubor informací, kteý se nachází na jednom řádku v tabulce (v databázi). Kromě toho je to model (vzor), podle kterého vytváříme tabulky a sloupce v tabulkách. V modulu domain se nachází všechny entity.

Máme celkově šest entit/modelů:

- **Forum** - Má automaticky generované Id, jméno, jméno autora a Id autora. Může mít otce a syny.
- **Post** - Tento model má také automaticky generované Id, název, obsah, lajky, dislajky, název a Id autora, a fórum, do kterého daný Post patří. Tabulka Post je spojená s tabulkou komentáře a tabulkou Attitudes, které jsou vysvětleny níže.
- **Comment** je velice podobný modelu Post. Má název, obsah, počet lajků a dislajků, název a Id autora, Post, ke kterému se Comment vztahuje, a také list Attitudes.
- **CommentAttitude** a **PostAttitude** jsou takzvané „postoje“. Jsou v nich zapsané reakce uživatelů na komentáře a posty. Reakce mohou být následující – NEUTRAL, LIKE, DISLIKE. V databázi jsou reprezentováni pomocí 0, 1 a 2 díky tomu, že jsou definovány jako výčtové typy (anglicky enumerated, zkráceně enum). Aby nebyla tabulka postojů příliš velká, komentáře nebo příspěvky se kterými uživatel neměl interakci, nejsou v tabulce nijak reprezentovány. I absence kombinací Post – User nebo Comment – User je považováno za reakci (NEUTRAL), aby backend mohl v každém případě poskytnout informaci frontendu ohledně postoje.
- **LemonUser**. Stejně jako LemonService, LemonRepository a LemonController, je tento objekt rozšířením třídy z knihovny spring-lemon, kterou používáme k autentizaci uživatelů. Třidu AbstractUsers jsme rozšířili o políčko „jméno“, které původně v tabulce nebylo. Postupovali jsme podle dokumentace knihovny:

<https://github.com/naturalprogrammer/spring-lemon/wiki/Getting-Started-With-Spring-Lemon>

6.3. Repozitář

Repozitáře (repositories), s pomocí aplikačního rámce Hibernate, umožňují komunikaci mezi naší aplikací a databází. V případě **GET** requestu převádí Hibernate jeden nebo více sloupců tabulky (podle požadavků) na entity – tj. Java objekty. Dále s nimi pracujeme a následně poskytujeme frontendu přes naše API rozhraní.

```
@Override
public Post findById(Long id) {
    return postRepository.findById(id).isPresent() ? postRepository.findById(id).get() : null;
}
```

Obr. 10: Get request

Repozitář vrací objekt **Optional**, který obsahuje metodu **.isPresent()**. V případě, že byl v databázi nalezen objekt se zadaným id, funkce **findById()** vrátí objekt Post, jinak vrátí null. V případě **SAVE** requestu Hibernate přeměňuje Java objekty, které mu byly poskytnuty, na řádky ve správné tabulce. Příklad: Máme objekt, který chceme zapsat do databáze(viz ob. 11):

```
@Override
public Post createPost(Post post) { return postRepository.save(post); }
```

Obr. 11: Save request

Pomocí **repository.save(Object)**, který vrací objekt, který byl právě zapsán, dostaneme další řádek v tabulce (viz ob. 12):

id	content	dislikes	likes	title	forum_id
7	<p>^{Hello}</p>	3	12	Hey	1
51	^{<s><u>AHOJfbjkwethler...</u></s>}	1	0	Me	1
52	<p>szdfb</p>	0	1	ASDFV	1
64	<p>Hello!</p>	0	1	This post is meant for forum 1	1
67	<p>sdf</p>	0	0	asdf	1
68	<p>er,gnerge</p>	0	0	Heeeleleoeoele	1

Obr. 12: Tabulka posts

V případě **UPDATE** requestu Hibernate změní některou hodnotu již existujícího objektu. Tyto requesty voláme přes servisy, jež jsou popsány v další kapitole.

6. 4. Service a Rest

Balíček **service** obsahuje tři rozhraní (interface) a čtyři třídy, jež jsou implementacemi těchto rozhraní (čtvrtá implementuje službu knihovny spring–lemon). Služby spojují řadiče API (REST Controllers) a repozitáře.

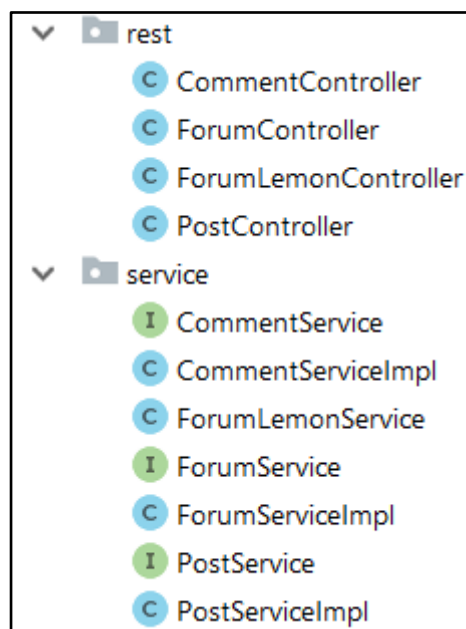
Balíček **rest** má celkově čtyři řadiče, které poslouchají na specifické url, které jsme definovali pomocí anotací **@GetMapping**, **@PostMapping** nebo **@PutMapping** podle toho, jaký request očekáváme. Každý z těchto řadičů odpovídá za jinou tabulku - posts, comments, forums, users. Například lajkování postů je žádost typu **PUT** (=UPDATE), jelikož pouze měníme počet lajků již existujícího příspěvku. Vytváření nového komentáře nebo postu bude **POST** a například

žádost o jména všech fór bude **GET**. Různé žádosti vyžadují různá oprávnění, například jenom uživatel může vytvářet komentáře, ale i nepřihlášení mohou načítat příspěvky. Operace, které jsou určeny pouze pro přihlášené, jsme vymezili v souboru **config/SecurityConfig**.

GET žádosti podporují **Pageable**, což znamená, že můžou posílat data po částech.

Spolupráci modulů service a rest vysvětlíme příkladem: Uživatel načte naši stránku poprvé. Frontend pošle **GET** žádost serveru, aby dostal příspěvky z fóra třeba s id 2. Pošle je tam, kde poslouchá Controller, který je zodpovědný za tabulku posts. Jakmile backend obdrží tuto **GET** žádost, zkontroluje, zda má uživatel práva na přístup k těmto datům. Pokud ano, přepošle žádost do **PostService**, jehož úkolem je manipulace s přijatými daty. Služba následně získá potřebné informace z repozitáře a vrátí informace do řadiče, který je odešle uživateli jako odpověď (response). Seznam všech endpointů naleznete zde:

<https://documenter.getpostman.com/view/5103979/S1LvW99D>.

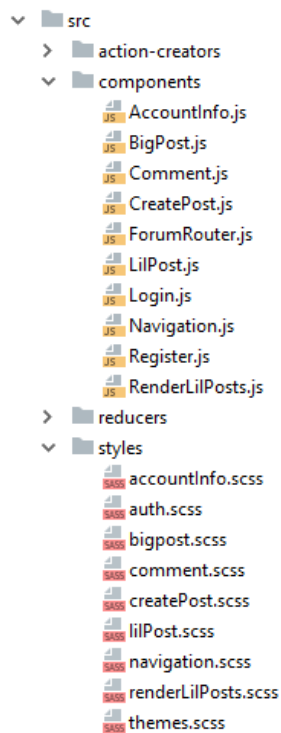


Obr. 13: Struktura Rest a Service

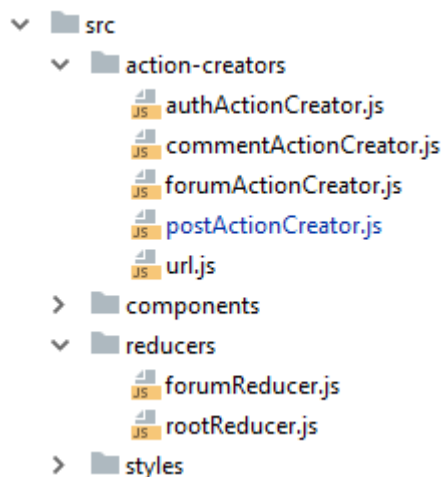
7. Frontend

7. 1. Struktura projektu

Frontend naší aplikace se skládá převážně z reactových komponent, přičemž většina z nich má definované vlastní kaskádové styly (viz obrázek). V našem projektu však nepoužíváme standardní css, nýbrž jeho rozšířenou syntaxi scss (Sassy CSS). Scss soubor každé komponenty, která má vlastní scss soubor, se nazývá stejně jako tato komponenta, tyto názvy se liší pouze notací (LilPost.js – reactová komponenta, lilpost.scss – kaskádové styly). Tři komponenty však nemají vlastní scss soubor, tudíž nedodržují tuto strukturu, těmito komponentami jsou: ForumRouter, Login a Register. Login a Register sdílejí jeden scss soubor, jelikož jsou velmi podobné, co se html týče, a ForumRouter je komponenta, jež pouze definuje v projektu, které komponenty se ukazují pro jednotlivá url, proto nepotřebuje vlastní kaskádové styly. Standardní struktura našeho projektu se vymyká ještě soubor themes.scss, jelikož se nepoutá k žádné konkrétní komponentě, nýbrž definuje barvy, které používají ostatní scss soubory. Kromě komponent a kaskádových stylů obsahuje naše frontendová aplikace také soubory, pomocí nichž komunikuje s api a pomocí nichž si vytváří lokální úložiště. Tyto soubory jsou uloženy zpravidla ve složkách action-creators a reducers (obrázek). V projektu dělíme action-creators podle jejich funkce na čtyři soubory. Pro autorizaci je to authActionCreator, pro komentáře commentActionCreator, pro fóra forumActionCreator a pro příspěvky postActionCreator. Url.js není action-creator, přestože se nachází ve složce s action-creators. V tomto souboru je uloženo url, na němž běží backend, aby se mohlo globálně měnit (viz obrázek). Všechny funkce v action-creators, posílají do reducerů náklad (payload), jenž dostávají od backendu v podobě odpovědí (responses). V reducerech se tento náklad zapisuje do reduxového lokálního úložiště, které se nachází v souboru store.js. Základem celého frontendu je soubor App.js, v němž se nachází komponenta ForumRouter napojená na reduxové úložiště, což vytváří finální aplikaci.



Obr. 14: Frontend struktura



Obr. 15: Action-creators a reducers

```
export const url = "http://localhost:7373";
```

Obr. 16: Používaná url

7. 1. Action-creators a reducers

7. 1. a) AuthActionCreator

V `authActionCreatoru` jsou dvě funkce: **`login`** a **`register`**. Obě přijímají objekt `creds` jako parametr od komponent, ve kterých jsou formuláře na přihlášení a registraci a od kterých dostávají přihlašovací údaje. Funkce **`register`** poté pošle tyto přihlašovací údaje backendu v podobě **`POST`** requestu na url **`url/api/core/users`** a dostává od backendu odpověď. Nakonec volá funkci **`login`**, které předává stejné přihlašovací údaje, které zpočátku dostala od své komponenty. Funkce **`login`** také posílá přihlašovací údaje backendu v podobě **`POST`** požadavku, avšak na url **`url/api/core/login`**. Od backendu dostává odpověď, podle níž uživatele buď přihlásí, nebo jeho požadavek na přihlášení odmítne.

7. 1. b) CommentActionCreator

V `commentActionCreatoru` jsou čtyři funkce, které se zabývají správou komentářů. První je funkce **`fetchComments`**, která dostává jako parametry stránku a id příspěvku, ke kterému má dostat komentáře. Následně posílá **`GET`** request na url **`url/comments/postId/post?page=postPage&size=10`**, kde `postId` a `postPage` jsou parametry této funkce. Parametr **`postPage`** a velikost (`size`), což není parametr a rovná se 10, určuje, které komentáře má funkce dostat jako odpověď, např.: pro `postPage` 0 by to bylo prvních 10 komentářů pro daný post, pro `postPage` 1 by to byly komentáře 11-20. Jako odpověď od backendu dostává funkce **`fetchComments`** tyto komentáře a ukládá je přes reducer do reduxového úložiště. Druhá funkce **`createComment`** dostává jako parametry objekt `comment` a `postId` od komponenty, ve které se nachází formulář na vytvoření komentáře. Objekt `comment` posílá backendu v **`POST`** requestu na url **`url/comments/create/postId`**, v němž `postId` je parametrem, který udává příspěvek, ke kterému se komentář vztahuje. Po přijetí odpovědi od backendu ukládá funkce tento komentář přes reducer i do úložiště. Třetí funkcí je **`deleteComment`**, má jeden parametr `commentId` a posílá na url **`url/comments/delete/commentId`** DEL požadavek. Po obdržení response od backendu maže komentář s tímto id i z úložiště. Poslední funkce se nazývá **`updateComment`** a zodpovídá za reakce na komentáře (like, dislike). Jako parametry dostává `attitude`, která slouží k identifikaci reakce (like, dislike, neutrální), a `commentId`. Backendu posílá **`PUT`** request na url **`url/comments/update/attitude/commentId`**, kde `commentId` a `attitude` jsou již dříve

zmíněné parametry. Odpověď v podobě aktualizovaného komentáře, kterou obdrží od backendu, přeposílá reduceru, jenž tuto odpověď ukládá do úložiště.

7. 1. c) ForumActionCreator

Ve forumActionCreatoru jsou tři funkce: **searchForumByName**, **createForum** a **fetchAllForumNames**, jejichž úkolem je dostávat a vytvářet nová fóra. Funkce **searchForumByName** dostává jako parametr jméno fóra, které má dostat. Posílá **GET** požadavek na url **url/forum/search/name**, kdy name je parametrem, a jako odpověď dostává fóra, jejichž název obsahuje toto jméno. Tyto fóra ukládá přes reducer do úložiště. Druhou funkcí je **createForum**, jež obdrží od komponenty objekt forum jako parametr. Následně na url **url/forum/create** zašle **POST** request s tímto fórem a po obdržení odpovědi zapisuje toto fórum i do reduxového úložiště. Poslední funkce **fetchAllForumNames** posílá na url **url/forum/all** **GET** request, v něhož odpovědi dostává názvy všech fór, které přeposílá do úložiště skrz reducer.

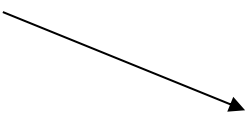
7. 1. d) PostActionCreator

Největším naším action-creatorem je postActionCreator, který se skládá ze sedmi funkcí. Funkce **fetchPosts** dostává jako parametry id fóra a stránku fóra, z něhož má dostat příspěvky. Na url **url/post/forum/forumId/posts?page=forumPage&size=2** (kde forumId a forumPage jsou parametry) posílá **GET** požadavek, jehož odpovědí jsou příspěvky z fóra s daným id a na dané stránce s velikostí 2 (př.: když se stránka rovná nule, dostane první dva příspěvky, když se stránka rovná jedničce, tak dostane třetí a čtvrtý příspěvek). Tuto odpověď ukládá do úložiště přes reducer. Další funkcí je **fetchPostById**, jejíž jediným parametrem je id příspěvku, který má z api dostat. Posílá **GET** request na url **url/post/id/postId**, v němž postId je parametr. Odpovědí je příspěvek s daným id, který je přeposlán do reduceru, aby byl uložen do reduxového úložiště. Třetí funkce se nazývá **getPostById** a její úkol je stejný jako úkol funkce **fetchPostById**. Rozdílem je, že **getPostById** neposílá požadavky do backendu, místo toho dostává příspěvek s daným id z reduxového úložiště přes reducer. Účelem zapojení této funkce v našem projektu je odstranění zasílání zbytečných requestů backendu. Další funkcí, která komunikuje pouze s reduxovým úložištěm je **resetPosts**. Tato funkce maže v lokálním úložišti přes reducer všechny příspěvky, které jsou v něm uloženy. Pátou funkcí je **deletePost**, která

dostává jako parametr `postId`. Api poté posílá **DEL** request na url **`url/post/delete/postId`**, kde `postId` je parametrem. Poté, co obdrží odpověď od backendu, maže přes reducer tento příspěvek i v úložišti. Předposlední funkcí je **`createPost`**, jež dostává jako parametry objekt `post` a `id fóra`. Následně posílá tento objekt v **POST** požadavku backendu na url **`url/post/create/forumId`**, v němž `forumId` je parametrem. Po obdržení odpovědi, zapisuje tento nový příspěvek do úložiště. Poslední funkce v `postActionCreatoru` se nazývá **`updatePost`** a jejími parametry jsou `attitude` a `id příspěvku`. Tato funkce posílá **PUT** request na url **`url/post/update/attitude/postId`**, v němž `attitude` a `postId` jsou parametry. `Attitude` stejně jako ve funkci **`updateComment`** určuje o jakou reakci se jedná (0 je neutrální, 1 je like a 2 je dislike), `postId` udává, na který příspěvek uživatel zareagoval. V odpovědi od backendu se skrývá aktualizovaný příspěvek (s naší reakcí), který je zaslán do úložiště přes reducer.

7. 1. e) ForumReducer a rootReducer

Přes **`forumReducer`** se ukládají data do reduxového úložiště. Podle typu akce poslané z `action-creatoru` se zvolí případ (case) a přepíše se položka v úložišti na náklad (payload) této akce, zbytek úložiště zůstane netknutý (vizualizace). `RootReducer` slouží jako soubor, v němž se sdruží všechny reducery do jednoho, který se napojí na store (úložiště), v našem programu však nemáme více reducerů, nicméně hodí se pro případ rozšíření současného projektu.



```

case 'FETCHED_POST_BY_ID':
  return {
    ...state,
    post: action.payload
  };

export const fetchPostById = (id) => dispatch => {
  fetch(url+"/post/id/" + id)
    .then(response => response.json())
    .then(e =>
      dispatch({
        type: 'FETCHED_POST_BY_ID',
        payload: e
      }));
};

const initialState = {
  status: false,
  storage: [],
  posts: [],
  arrayOfForums: [],
  post: "",
  logged: false,
  updatedPost: {post: {}, attitudeDto: {}},
  updatedComment: {comment:{},attitudeDto: {}},
  comments: ""
};

```

Obr. 17: ForumReducer a rootReducer

7. 2. Samotné komponenty

7. 2. a) Navigation

Navigation je komponenta přítomná na všech url



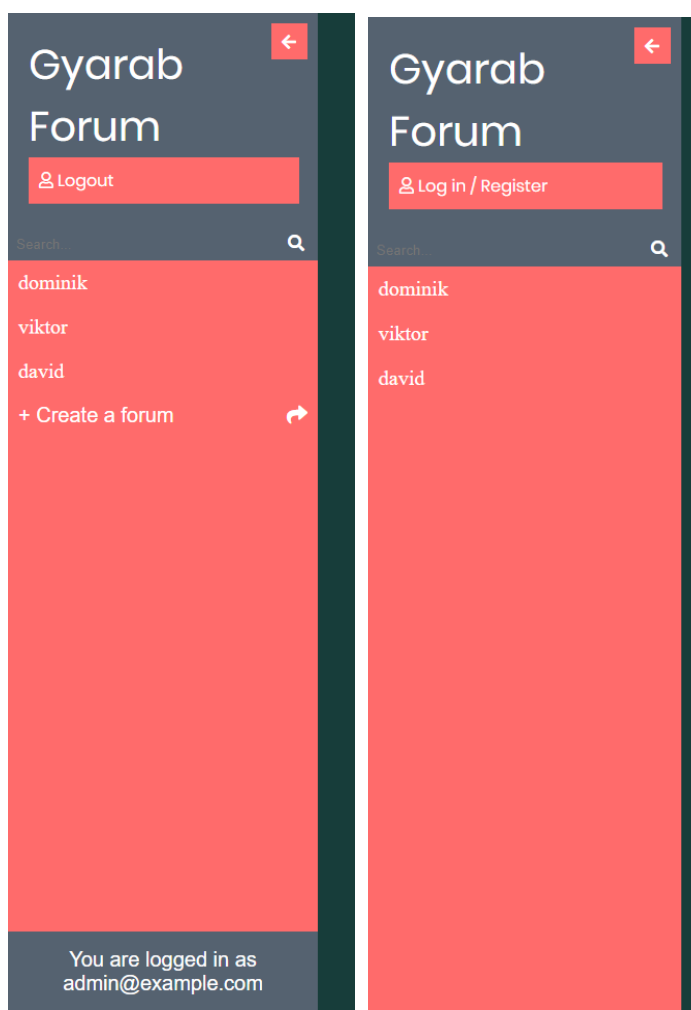
Obr. 18: Button

našeho projektu, protože slouží k navigaci mezi jednotlivými url.

Má dvě podoby: tlačítka (button) a postranního panelu (sidebar). Tlačítko (viz ob. 18) slouží k otvírání sidebaru, přičemž Navigation samotná je postranní panel (viz ob. 19).

Otevřená Navigation má několik částí. Nápis Gyarab Forum slouží jako link na úvodní stránku. Pod ním se nachází tlačítko, které slouží buď jako link na stránku s přihlášením a registrací, nebo na odhlášení uživatele (podle toho, jestli je uživatel již přihlášen).

Níže se nachází sekce se samotnými názvy fór, na obrázku se tyto fóra jmenují dominik, viktor a david. Navigation tyto fóra dostává z úložiště, do kterého se dostávají z api pomocí funkce fetchAllForumNames, jež se nachází ve forumActionCreatoru. K této sekci také náleží pole pro vytvoření nového fóra a tlačítko pro jeho odeslání, obě tyto

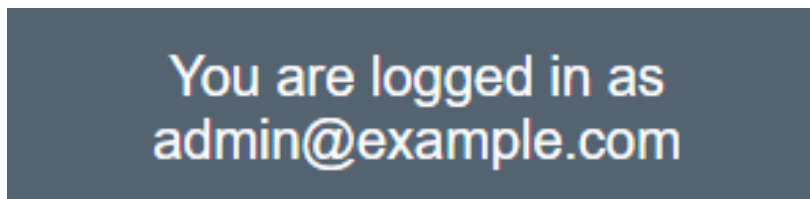


Obr. 19: Navigation bar a práva uživatele

položky se nacházejí pod názvy fór. Odeslání nového fóra zajišťuje funkce `createForum` nacházející se ve `forumActionCreatoru`. Mezi tlačítkem a sekci s fóry je kolonka pro zadání názvu fóra, které chce uživatel vyhledat, a v dolní části komponenty se v případě přihlášeného uživatele ukazuje komponenta `AccountInfo`. Získání vyhledávaných fór z api zajišťuje funkce `searchForumByName` ve `forumActionCreatoru`. Poslední částí komponenty je tlačítko v pravé horní části, které slouží k zavření `Navigation`, také se však sidebar zavře, jakmile uživatel klikne mimo něj. Zbytek webové stránky je ztmaven, když je `Navigation` otevřena.

7. 2. b) `AccountInfo`

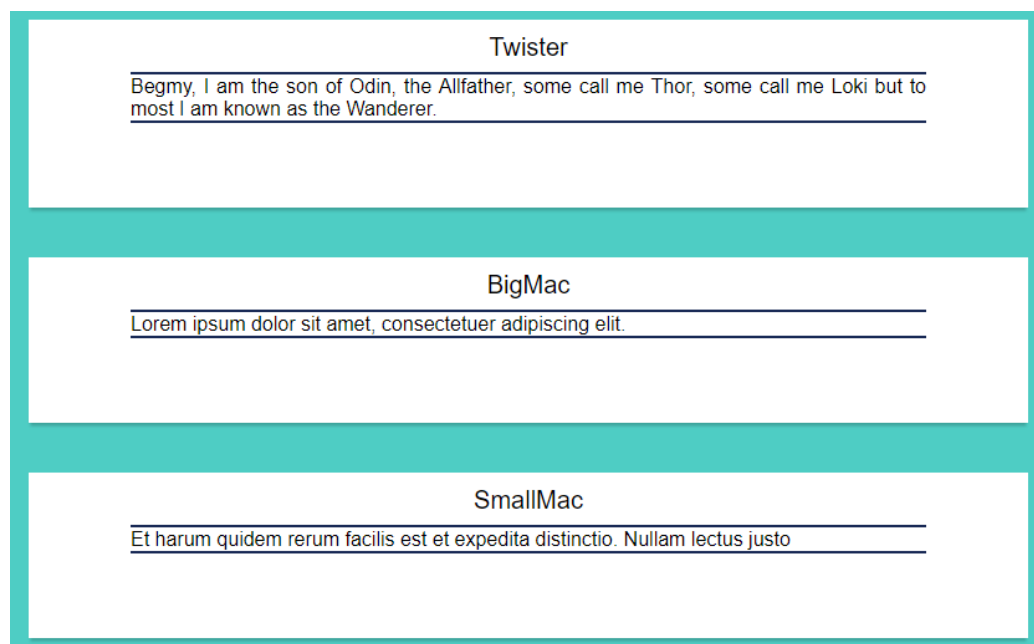
Komponenta `AccountInfo`, jež se nachází v `Navigation`, slouží momentálně pouze k zobrazování e-mailu, pod kterým je uživatel přihlášen. Jedná se o samostatnou komponentu především, aby se zpřehlednila komponenta `Navigation`.



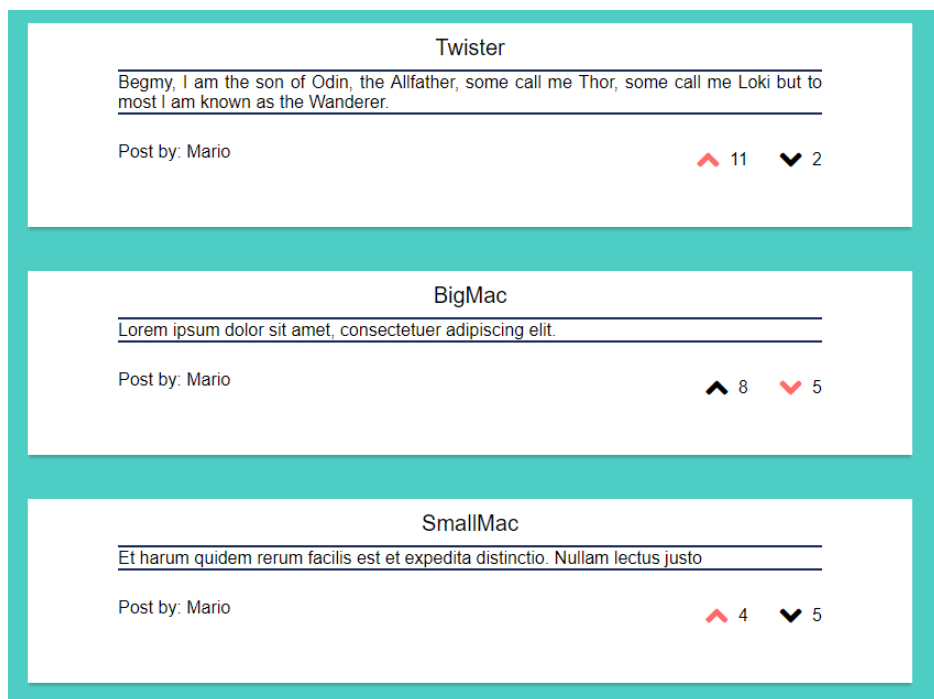
Obr 20: Zobrazení e-mailu uživatele

7. 2. c) LilPost

V našem projektu jsou jednotlivé příspěvky uživatelů (posty) reprezentovány dvěma komponentami: LilPost a BigPost. LilPost má sloužit jako náhled samotného postu, proto ukazuje pouze část jeho obsahu a nadpis. Pokud je uživatel přihlášen ukazuje ještě jméno autora příspěvku/ikonu popelnice a počet pozitivních a negativních ohlasů (lajk, dislajk). Jestli se ukazuje jméno autora, nebo ikona popelnice záleží na tom, zdali je přihlášený uživatel autorem postu, nebo ne. Autor příspěvku má totiž možnost svůj příspěvek odebrat kliknutím na tuto ikonu (to zajišťuje funkce deletePost v postActionCreatoru). Každý LilPost má tři části – záhlaví, tělo a zápatí. V záhlaví je ukázaný nadpis postu, v těle obsah příspěvku a v zápatí jméno autora/ikona popelnice a ohlasy. Tyto části jsou odděleny čarami pro přehlednost. V zápatí může uživatel zareagovat na příspěvek kliknutím na jedno ze dvou tlačítek, které se následně zabarví (viz ob. 22). Svou reakci může uživatel změnit z lajku na dislajk a naopak, nemůže však zareagovat víckrát po sobě stejně. Na posty reagují uživatelé pomocí funkce updatePost, jež se nachází v postActionCreatoru. Záhlaví a tělo LilPostu slouží jako link na BigPost toho příspěvku, jako jehož náhled slouží daný LilPost.



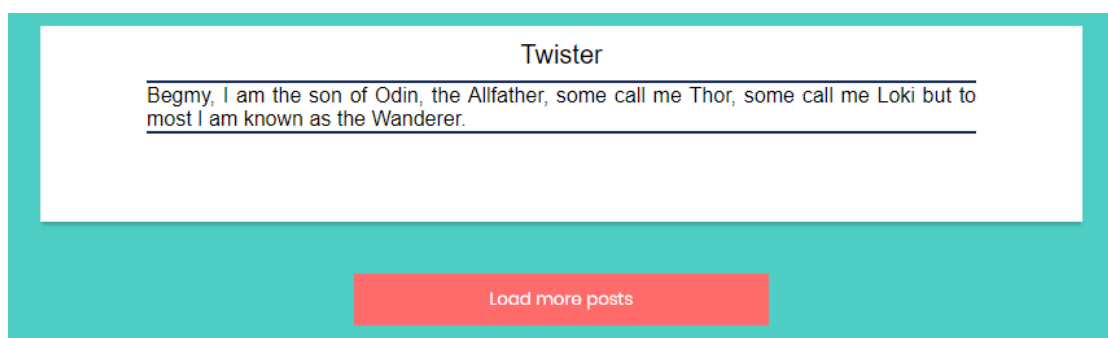
Obr. 21: Lilpost



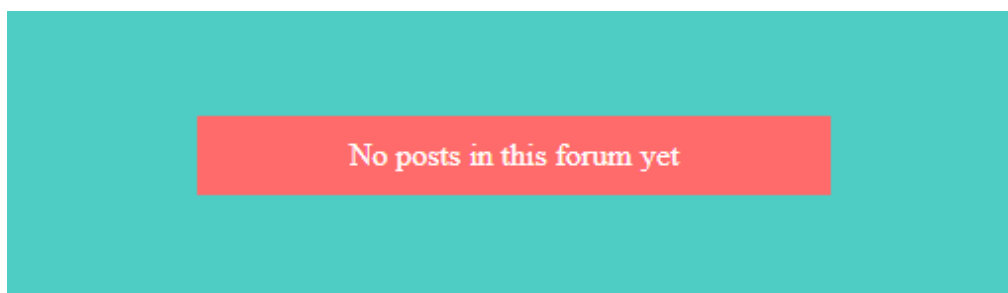
Obr. 22: Like a Dislike u Lilpostu

7. 2. d) RenderLilPosts

S komponentou LilPost úzce souvisí komponenta RenderLilPosts, jejíž funkcí je, jak již z názvu vyplývá, renderovat LilPosty. Zprvu se vždy načte pouze první příspěvek, přičemž při každém dalším kliknutí na tlačítko Load more posts, jež se nachází pod sekci s příspěvky (viz ob. 23), se načte další post. Komponenta příspěvky získává z úložiště, do kterého se dostávají z api pomocí funkce fetchPosts, jež se nachází v postActionCreatoru. Když jsou již načteny všechny příspěvky, tlačítko Load more posts zmizí. Posty se renderují podle toho, na kterém fóru se právě uživatel nachází, přičemž pro úvodní stránku se načítají příspěvky z fóra s id 1. Pokud na daném fóru nejsou žádné příspěvky, komponenta sdělí uživateli tuto informaci (viz ob. 24).



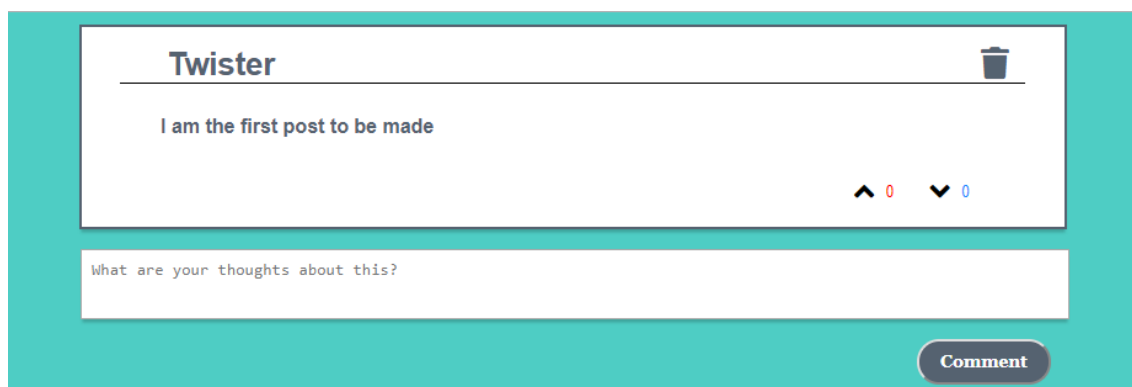
Obr. 23: Render button více postů



Obr. 24: Absence Lilpostů v daném fóru

7. 2. e) BigPost

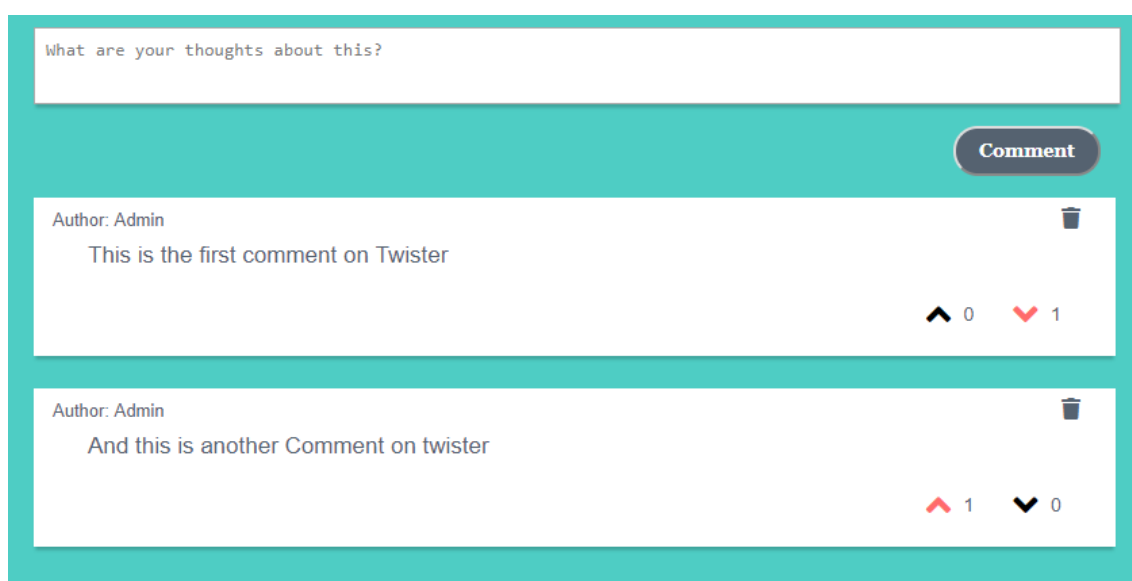
Druhou verzí zobrazování příspěvků je komponenta BigPost. Na stránku s touto komponentou se uživatel dostane kliknutím na kartu LilPostu, která reprezentuje tentýž příspěvek. V Bigpostu jsou uloženy stejné informace o příspěvku jako v LilPostu, avšak zobrazuje se v něm celý text daného postu, nikoliv pouze jeho část. Kromě toho se v BigPostu také zobrazují komentáře týkající se daného příspěvku, které získává komponenta z reduxového úložiště, do nějž je z api přeposílá funkce `fetchComments` v `commentActionCreatoru`. Nachází se v něm také formulář pro vytvoření nového komentáře, což zajišťuje funkce `createComment` v `commentActionCreatoru`. Formulář pro vytvoření nového komentáře se nachází pod BigPostem a komentář se odesílá pomocí tlačítka `Comment` (viz ob. 25). Příspěvek, který BigPost zobrazuje, bere komponenta z reduxového úložiště, do nějž se dostává díky funkci `fetchPostById` a `getPostById`, přičemž obě tyto funkce jsou v souboru `postActionCreator.js`. Také se dá na post přes BigPost zareagovat pomocí tlačítek v pravo dole (díky funkci `updatePost` v `actionCreatoru`). V poslední řadě může autor příspěvku post smazat pomocí ikony popelnice v pravém horním rohu, což zajišťuje funkce `deletePost` v `postActionCreatoru`.



Obr. 25: Bigpost

7. 2. f) Comment

Komponenta Comment se zobrazuje v BigPostu a reprezentuje komentáře uživatelů týkající se daného příspěvku. Comment ukazuje, kdo je jeho autorem v levém horním rohu, svůj obsah a reakce ostatních uživatelů na tento komentář v pravo dole (viz ob. 26). Stejně jako u BigPostu a LilPostu, pokud je přihlášený uživatel autorem komentáře, má možnost tento komentář odstranit pomocí ikony popelnice v pravém horním rohu (to zajišťuje funkce deleteComment v commentActionCreatoru). Reaguje se na komentář přes ikony v pravém dolním rohu, což zajišťuje funkce updateComment v commentActionCreatoru.

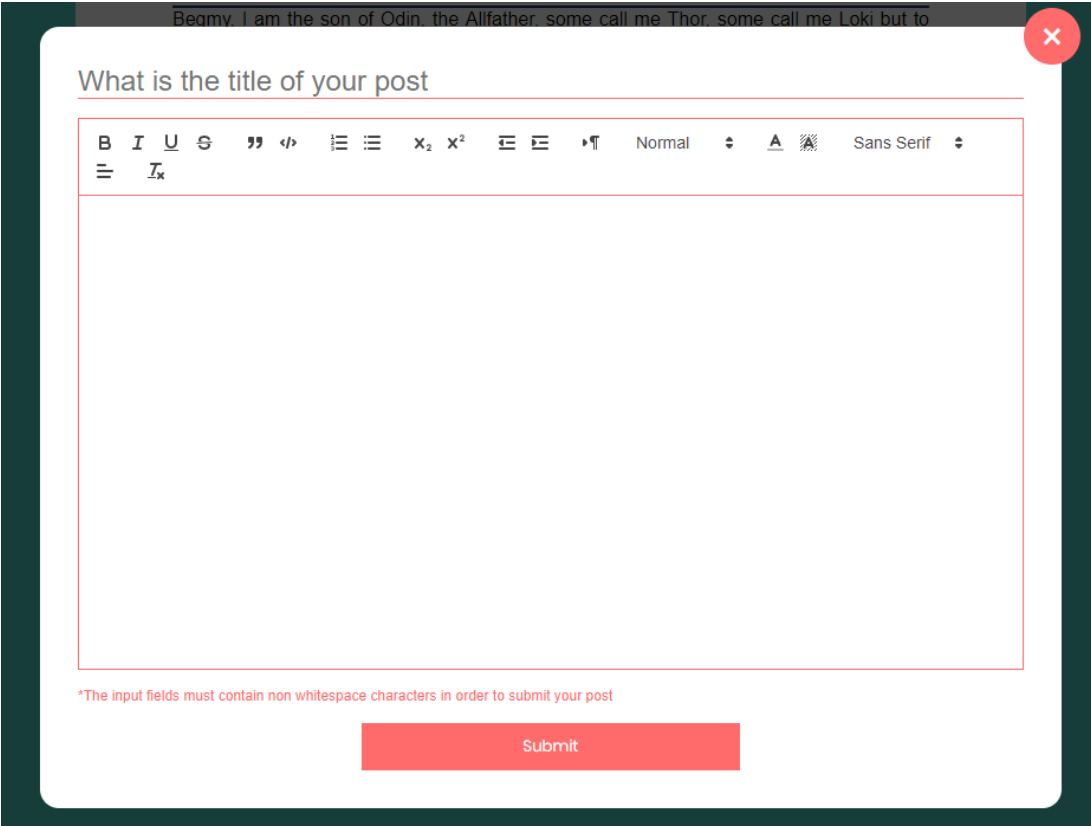


Obr. 26: Příklad Commentu

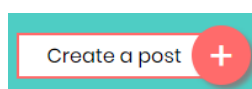
7. 2. g) CreatePost a ReactQuill

Komponenta CreatePost slouží k vytváření nových příspěvků do jednotlivých fór. Podobně jako Navigation má dvě podoby: tlačítko a modal (viz ob. 27, 28). Tlačítko slouží k otvírání modalu a v modalu se nachází samotný formulář pro vytvoření nového postu. Tento formulář má dvě pole, první slouží pro vytvoření nadpisu příspěvku a druhý na vytvoření jeho obsahu. Druhé pole je komponenta ReactQuill, což je rich text editor. ReactQuill dává možnost našim uživatelům měnit kaskádové styly textu, který píšou, a jeho výstupem je html. Tlačítko Submit v dolní části modalu slouží pro odeslání příspěvku, avšak komponenta dovolí příspěvek odeslat, právě když nejsou pole formuláře prázdná (nesmí být ani whitespace). Tento příspěvek je poslán api přes

funkci `createPost`, která se nachází v `postActionCreatoru`. V pravé horní části je poté tlačítko na zavření modalu, přičemž stejně jako u komponenty `Navigation` existuje i možnost kliknutí mimo modal, která také vede k zavření modalu. Když uživatel zavře modal a pole nejsou prázdná, jejich obsah se uloží a při opětovném otevření modalu může uživatel pokračovat v psaní. Obsah těchto polí se smaže pouze při opuštění aktuálního url. Zbytek webové stránky je stejně jako u `Navigation` zmaven, když je modal otevřen. Uživatel vždy vytvoří příspěvek do fóra, na kterém se zrovna nachází, to znamená, že nelze vytvořit příspěvek na fórum s id 2 z fóra s id 1. Tlačítko pro otevření modalu se člověku vůbec nezobrazí, jakmile není přihlášen.

A screenshot of a modal form for creating a post. The modal has a dark green border and a white background. At the top, there is a text input field with the placeholder text "What is the title of your post". Below the title field is a rich text editor toolbar with various icons for bold, italic, underline, link, unlink, list, indent, decrease indent, increase indent, text color, background color, and font family. Below the toolbar is a large text area for the post content. At the bottom of the modal, there is a red "Submit" button. A small red error message is visible below the text area: "*The input fields must contain non whitespace characters in order to submit your post". A red close button with a white 'x' is located in the top right corner of the modal.

Obr. 27: ReactQuill



Obr. 28: Button na vytvoření příspěvku

7. 2. h) Login

Login je komponenta, která se stará o sbírání přihlašovacích informací potřebných k přihlášení (e-mail a heslo). Login obsahuje formulář s kolonkami pro každý z těchto přihlašovacích údajů. Pod těmito kolonkami se nachází tlačítko Login (viz ob. 29), které po kliknutí odesílá přihlašovací údaje pro prověření do backendu přes funkci login v authActionCreatoru. Poslední položkou v této komponentě je odkaz na url s registrací pro uživatele, kteří ještě nemají vytvořený uživatelský účet.

The image shows a login form on a teal background. It consists of two white input fields: the top one is labeled 'E-Mail' and the bottom one is labeled 'Password'. Below these fields is a red rectangular button with the word 'Login' in white text. At the bottom left of the form area, there is a link that says 'Or Register here' in a purple color.

Obr. 29: Login Component

7. 2. i) Register

Komponenta Register stejně jako komponenta Login obsahuje formulář s několika kolonkami (uživatelské jméno, e-mail, heslo a potvrzení hesla). Pod těmito kolonkami je tlačítko Register (viz ob. 30), které po kliknutí posílá uživatelem zadané přihlašovací údaje backendu pro vytvoření uživatelského účtu (přes funkci register v authActionCreatoru).

Registration form layout:

- Username input field
- E-mail input field
- Password input field
- Re-enter Password input field
- Register button (red)

Obr. 30: Registrace

7. 2. j) ForumRouter

Jak již bylo řečeno ForumRouter je komponenta, jež definuje, které komponenty se ukazují pro jednotlivá url, a nemá vlastní kaskádové styly. V našem projektu máme 5 obdob url. První z nich končí /, na ní se ukazují komponenty Navigation, CreatePost a RenderLilPosts. Další obdoba končí na /login, na ní se načítají komponenty Login a Navigation. Podobně je to s komponentou Register na url končící /register. O trochu komplikovanější je to s url končící forums/forumId, jelikož forumId se mění podle toho, na kterém fóru se právě uživatel nachází. Stejně jako v prvním případě obsahuje tato url komponenty Navigation, CreatePost a RenderLilPosts. V poslední situaci jde o podobný princip, ovšem místo fóra se na url zakončeným /bigpost/postId jedná o příspěvek (kde postId je id tohoto příspěvku), proto se na něm renderují komponenty Navigation, BigPost a Comment.

```

1  import ...
2
3
4
5
6
7
8
9
10 class ForumRouter extends Component {
11
12   render() {
13     return (
14       <div>
15         <Route path="/" component={Navigation}/>
16         <Route path="/" exact component={RenderLilPosts}/>
17         <Route path="/login" exact component={Login}/>
18         <Route path="/register" exact component={Register}/>
19         <Route path={`/${forums}/:forumId`} exact component={RenderLilPosts}/>
20         <Route path={`/${bigpost}/:postId`} component={BigPost}/>
21       </div>
22     );
23   }
24 }
25
26 export default ForumRouter;

```

Obr. 31: ForumRouter

8. Závěr

Tento projekt byl zajímavým přínosem pro nás všechny, zejména kvůli spolupráci a učení se nových možností pro vývoj webových aplikací. Týmová spolupráce je často nezbytná pro vývojáře, a tak si této zkušenosti, se kterou jsme občas měli problémy, vážíme. React je bezpochyby velmi značným přínosem, rozhodneme-li se pracovat v tomto odvětví. Co se týče samotné práce, zadání jsme z velké většiny splnily, ovšem unikly nám nějaké námi slíbené funkce. Na druhou stranu jsme však přidali odstranění komentářů a postů, které jsou dle našeho názoru vítanou a neméně důležitou funkcí. S prací jsme spokojeni a v budoucnu je zde možnost dalšího rozšíření projektu.

9. Zdroje

Javascript whitespace check. Dostupné z URL:

<<https://stackoverflow.com/questions/2031085/how-can-i-check-if-string-contains-characters-whitespace-not-just-whitespace>> (Navštíveno 9. 5. 2019)

React-Quill. Dostupné z URL: <<https://github.com/zenoamaro/react-quill>> (Navštíveno 9. 5. 2019)

CSS. Dostupné z URL: <<https://www.w3schools.com/css/default.asp>> (Navštíveno 9. 5. 2019)

Github. Dostupné z URL: <<https://cs.wikipedia.org/wiki/GitHub>>(Navštíveno 9. 5. 2019)

React. Dostupné z URL: <[https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))> (Navštíveno 9. 5. 2019)

Životní cyklus komponenty. Dostupné z URL: <<https://code.likeagirl.io/understanding-react-component-life-cycle-49bf4b8674de>>(Navštíveno 9. 5. 2019)

Node Package Manager. Dostupné z URL:

<[https://en.wikipedia.org/wiki/Npm_\(software\)](https://en.wikipedia.org/wiki/Npm_(software))> (Navštíveno 9. 5. 2019)

Fontawesome. Dostupné z URL: <<https://origin.fontawesome.com/icons?d=gallery>> (navštíveno 9. 5. 2019)

Ob. 2: React lifecycle. Dostupné z URL:

<<https://programmingwithmosh.com/javascript/react-lifecycle-methods/>>