

**STREDNÁ PRIEMYSELNÁ ŠKOLA JOZEFA MURGAŠA,
HURBANOVA 6, BANSKÁ BYSTRICA**



TRIEDNA SOCIÁLNA SIET'

MATURITNÁ PRÁCA PČOZ forma b)

SPRIEVODNÁ DOKUMENTÁCIA

MARIÁN LORINC, IV. A

BANSKÁ BYSTRICA 2017/2018

**STREDNÁ PRIEMYSELNÁ ŠKOLA JOZEFA MURGAŠA,
HURBANOVA 6, BANSKÁ BYSTRICA**

Meno a priezvisko: Marián Lorinc

Trieda: IV. A

Školský rok: 2017/2018

Odbor: 2694 6 informačné a sieťové technológie

Názov práce: Triedna sociálna sieť

Zadanie práce: Vytvoriť webové rozhranie pre užívateľa

Nakonfigurovať MariaDB databázu

Naprogramovať REST API

Zabezpečiť dáta a prístup

Pripraviť stránku na produkciu

Napísať sprievodnú dokumentáciu

Dátum zadania: 21. 10. 2017

Konzultant: Ing. Ivan Košťál

Podpis konzultanta:

ČESTNÉ VYHLÁSENIE

Čestne vyhlasujem, že maturitnú prácu som vypracoval samostatne s použitím uvedenej odbornej literatúry.

Banská Bystrica 10. 2. 2018

.....

vlastnoručný podpis autora

OBSAH

ZOZNAM OBRÁZKOV	7
ÚVOD	8
1 TEORETICKÉ VÝCHODISKÁ	9
1.1 Spring – serverová časť, backend	9
1.1.1 Anotácie	9
1.1.2 Beany	9
1.1.2.1 Definícia	9
1.1.2.2 Kontrolér - @Controller	10
1.1.2.3 Služba - @Service	10
1.1.2.4 Repozitár - @Repository	10
1.1.2.5 Komponent - @Component	11
1.2 Klientska časť, frontend.....	11
1.2.1 React - javascript	11
1.2.1.1 Element, komponent a imutabilita objektu	11
2 CIELE PRÁCE	13
2.1 Serverová časť	13
2.2 Klientska časť	13
3 METODIKA PRÁCE	15
3.1 Úvod do projektu	15
3.2 Výber databázy	15
3.3 Doménový model.....	15
3.3.1 Validácia dát	15
3.4 Kompilácia a kompilačné nástroje na strane frontendu.....	15

3.5	Spravovanie knižníc.....	16
3.5.1	Maven	16
3.5.2	NPM – Node Package Manager.....	16
3.6	Výber renderovacej knižnice	16
3.7	Implementácia dynamického renderovania	16
3.7.1	Dynamické načítavanie dát zo servera.....	17
3.7.2	Problémy dynamického načítavania	18
3.7.2.1	Problém stránkovania	18
3.7.2.2	Problém so skrolovaním.....	20
3.7.3	CRUD metódy	20
3.7.3.1	Kolízie	21
3.8	Vzhľad stránky a veľká zmena v organizovaní súborov	21
3.8.1	Modifikácia Bootstrapu	21
3.8.2	Reštrukturalizácia projektu	21
3.8.3	Štýl stránky	22
3.8.4	Navigácia	22
3.9	Administrátorské rozhranie.....	22
3.9.1	Pravidlá	22
3.9.2	Rozhranie	22
3.10	Prehliadač skupín.....	23
3.10.1	Dynamické načítavanie pomocou React Virtualized.....	23
3.10.1.1	Obmedzenia.....	23

3.10.1.1.1	Pevne pridelená dĺžka.....	23
3.10.1.1.2	Dynamicky pridelená dĺžka.....	23
3.10.2	Meranie riadkov v liste	23
3.11	REST.....	25
3.11.1	ResourceAssembler	25
3.11.2	Objekty typu ResourceSupport	25
3.11.3	Kontroléry	25
3.11.4	Vlastná implementácia Pageable	26
3.12	Bezpečnosť	26
3.13	Nasadenie servera do produkcie	26
4	VÝSLEDKY PRÁCE	27
	ZÁVER	29
	ZOZNAM POUŽITEJ LITERATÚRY	30
	ZOZNAM PRÍLOH	31

ZOZNAM OBRÁZKOV

<i>Obrázok 1 Ukážka HATEOAS (Brisbin, a iní, 2018)</i>	17
<i>Obrázok 2 Popis stránkovania pri vymazaní elementu</i>	18
<i>Obrázok 3 Algoritmus pre výpočet stránkovania</i>	19
<i>Obrázok 4 Súčasná projektová štruktúra</i>	21
<i>Obrázok 5 Meranie výšky v GroupCarde</i>	24
<i>Obrázok 6 Metóda na prepočítavanie výšky v GroupBrowseri</i>	25

ÚVOD

Úlohou práce je vybudovať REST API, ktoré bude možné použiť na rôzne účely. Tento projekt sa presnejšie zameria na sociálnu sieť pre žiakov školy, ale stále si zachová generickú podobu.

Prezentačnú stránku bude tvoriť hlavná stránka, ktorá bude vstupným bodom po prihlásení užívateľa, a kde sa budú spravovať skupiny.

Ďalšou časťou bude skupina samotná, kde budú príspevky od členov a komentáre, teda hlavné miesto diania v skupine.

Nakoniec je tu administrátorské prostredie, kde sa bude spravovať skupina. Hlavnou úlohou ho bude spraviť autonómne, aby sa ľahšie spravovali skupiny globálnemu administrátorovi. Administrátorské prostredie bude jednoduché oproti dvom predchádzajúcim. Bude obsahovať tabuľku členov a tri polia: strana, veľkosť strany a vyhľadávacie pole.

Na konci práce by mala vzniknúť malá jednoduchá sociálna sieť, ktorá bude demonštrovať univerzálne API. Dôvody tohto postupu sú autorom vysvetlené na konci tejto práce.

1 TEORETICKÉ VÝCHODISKÁ

Webové aplikácie sú v súčasnosti najrozšírenejšími aplikáciami na svete. Ich nesmiernou výhodou je ľahké používanie, aktualizovanie a žiadna inštalácia.

Vyvíjanie webových aplikácií sa delí na dve roviny. Jedna je serverová, ktorá má na starosti uchovávanie dát, autentifikáciu a spracovávanie požiadaviek klientov. Druhá na strane klienta sa zaoberá zobrazovaním obsahu v internetovom prehliadači a získavaním dát zo servera.

1.1 Spring – serverová časť, backend

Serverovú časť má na starosti framework Spring. Uľahčuje prácu na projekte, tak že programátori sa nemusia plne venovať veciam ako je autentifikácia, autorizácia, práca s databázou a s manažovaním služieb. Táto práca presnejšie využíva Spring Boot, ktorý dopĺňa Spring o dodatočnú automatickú konfiguráciu, čo urýchlilo vývoj softvéru.

1.1.1 Anotácie

Zastupujú v Jave rolu metadát. Metadáta sú informácie o jednotlivom Java prvku a neovplyvňujú program priamo. V Jave sa označujú pomocou znaku @ a menom, napríklad @Component. Používajú sa v aspektovo orientovanom programovaní a pomocou nich autor môže s nimi narábať za behu programu. (Oracle)

Aspektovo orientované programovanie sa používa v prípadoch, keď nepoznáme presnú štruktúru a nemôžeme použiť objektovo orientované riešenie.

1.1.2 Beany

1.1.2.1 Definícia

Bean je objekt, ktorý má jediný konštruktor bez parametrov, obsahuje gettery a settery a má v sebe viaceré objekty, ktoré sú privátne. Beany musia spĺňať nasledujúce podmienky:

1. Introspekciu – framework musí vedieť beany analyzovať
2. Prispôsobivosť – užívateľ môže prispôbovať podobu beany pri používaní frameworku
3. Prepojitelnosť s ostatnými beanmi

4. Perzistentnosť, beany sa musia dať uložiť do súboru, a preto musia implementovať rozhranie Serializable

(Sun Microsystems Inc., 1997)

Spring používa beany na vkladanie závislostí (*injection alebo autowire*) do našich objektov. Väčšinou sa jedná o beany repozitárov, služieb alebo komponentov.

1.1.2.2 Kontrolér - @Controller

Označuje triedu ako prvok, ktorý bude spracovávať požiadavky klienta na určitej webovej lokalite. Obsahuje metódy, ktoré sú označené anotáciou alebo neoznačené, ktoré sú pomocné. Označené metódy poslúchajú na určité požiadavky a následne vracajú odpoveď klientom. Kontroléry majú na starosti zoskupovanie dát zo služieb a repozitárov a ich odoslanie klientom. (Pivotal Software, Inc., 2018)

Metóda v kontroléry môže byť označená pomocou anotácie RequestMapping, GetMapping, DeleteMapping, PostMapping, PutMapping a PatchMapping. Každá zastupuje jednu http metódu a určuje na aké požiadavky bude metóda reagovať. RequestMapping slúži na všeobecné mapovanie (*pomocou argumentu method*), kde môžeme pomocou argumentu method zadať viacero parametrov. Pomocou argumentu value, môžeme určiť webovú lokalitu. Argument môže nadobúdať hodnotu stringu alebo poľa stringov. Stringy môžu byť platné regulárne výrazy alebo môžu obsahovať aj URL premenné. (Pivotal Software, Inc., 2018)

1.1.2.3 Služba - @Service

Slúži ako vrstva pre doménovú logiku alebo ako facade pattern (*protokol*). Facade slúži na zjednodušenie rozhrania komplexnejšieho API. V tejto práci využíva tento protokol PermissionService.java, ktorý obaľuje bean repozitára GroupMembership. Poskytuje metódy na overenie členstva a postavenie člena v skupine.

1.1.2.4 Repozitár - @Repository

Používa sa pri objektoch, ktoré pristupujú k databáze a potrebujú špeciálne spracovávať databázové a iné programové chyby. (Johnson, a iní)

1.1.2.5 Komponent - @Component

Nakoniec je tu najvšeobecnejšia anotácia @Component. Každá, doteraz spomenutá anotácia je komponent. Líšia sa svojou špecializáciou, pričom komponent je najvšeobecnejší. Hoci anotácie je možné zameniť, v budúcnosti to tak byť nemusí. (Pivotal Software, Inc., 2018)

Používa sa, keď nemôžeme objekt označiť ako predchádzajúce anotácie. Napríklad objekt, ktorý konvertuje zložený primárny kľúč na string alebo jeho reprezentujúci objekt by sme označili ako komponent. Nie je to služba, lebo nesimuluje logiku medzi objektami. Nie je to ani kontrolér, lebo nereprezentuje dáta užívateľovi a repozitár v žiadnom prípade.

1.2 Klientska časť, frontend

Klientska rovina sa delí na kaskádovité štýly a Javascript (*ECMAScript*). Jednotlivé prvky si detailnejšie rozoberieme v metodike práce, ale ešte predtým je potrebné spomenúť pojmy, ktoré sa týkajú programovania v Javascripte.

1.2.1 React - javascript

Najhlavnejšou použitou technológiou v projekte je React. React je knižnica vyvíjaná Facebookom, ktorá je zameraná na renderovanie html komponentov. Na vytváranie komponentov sa používa JSX, čo je dodatok Javascriptu, a preto tento projekt používa transpiler Babel. React sa nemusí používať s JSX, ale syntax JSX je skoro totožný html, na rozdiel od čisto javascriptového spôsobu. (Facebook Inc.)

1.2.1.1 Element, komponent a imutabilita objektu

Element je najmenší prvkom v Reacte, často mýlený s komponentom. Element je imutabilný objekt, čo znamená, že element sa za behu programu už nezmení. Dalo by sa to prirovnať k jednému snímku z filmu. Objekt na snímkach sa môže javiť tak isto, ale snímky nie sú rovnaké, menia sa. Keď chceme pozmeniť imutabilný objekt, musíme skopírovať jeho obsah a následne až potom môžeme meniť stav. (Facebook Inc.)

Imutabilita sa môže na prvý pohľad javiť ako nevýhodná a pomalá. Nie je tomu tak, lebo existujú lenivé algoritmy, ktoré aktualizujú objekt, len keď je to potrebné. Navyše imutabilné nezanechávajú po sebe vedľajšie účinky, čo je pri viac vlákňovom programovaní užitočné. Taktiež sa dajú porovnávať predchádzajúce stavy, čo je

znázornené na oficiálnej stránke Reactu ako tutoriál. Tvorcovia využili fakt, že každý ťah piškvoriek je jedna imutabilná mriežka. Tú ukladali do stavu komponentu a nakoniec vyrenderovali podľa vstupu užívateľa. (Facebook Inc.)

Nakoniec, React dokáže pomocou immutability rozhodnúť, či treba určitý komponent aktualizovať. Komponent je prvok, ktorý pozostáva z viacerých elementov. (Facebook Inc.)

2 CIELE PRÁCE

Cieľom projektu je naprogramovať webovú aplikáciu, ktorá bude schopná poskytnúť študentom spôsob komunikácie a spolupráce. Prvoradým je spraviť jadro servera tak univerzálne, aby nebolo len limitované na školskú sociálnu sieť.

2.1 Serverová časť

1. Vybrať vhodný backendový framework na vývoj jadra servera
2. Vybrať vhodnú databázu (*engine*)
3. Navrhnuť databázovú schému
4. Zhotoviť doménový model
5. Naplánovať kaskádovanie akcií v Hibernate
6. Implementovať validáciu pomocou Hibernate a JPA anotácií
7. Sprístupniť základne endpointy (*kontroléry*)
8. Nainštalovať a nakonfigurovať Spring Data REST
9. Vytvoriť POJO (*objekt s gettermi a settermi*) objekty, ktoré implementujú ResourceSupport pre REST API
10. Ďalej vytvoriť assemblyery pre skonštruovanie HATEOAS formátu pre REST API a ich klientov
11. Dorobiť pár extra endpointov, ktoré nie sú vbudované v Spring Data REST pre našu stránku (*s javascriptom budeme sťahovať dáta*)
12. Upraviť springovské stránkovanie, aby sa podobali na limity a offsety v SQL (*štandardné podporujú len strany a počet elementov na stranu*)
13. Zabezpečiť repozitáre a kontroléry pomocou Spring Security
14. Uviesť server do produkcie

2.2 Klientska časť

1. Inicializovať git repozitár
2. Vybrať si vhodný responzívny css framework + css preprocesor
3. Vybrať si package manager
4. Nakonfigurovať build manager
5. Zvoliť si frontend renderovaciu javascript knižnicu/framework
6. Nainštalovať Babel – javascript transpiler, ktorý premení novší javascript na starší (*prehliadačova kompatibilita*)

7. Nastaviť javascript rest knižnicu
8. Naprogramovať renderovateľné komponenty
9. Skompilovať do produkčného vydania

3 METODIKA PRÁCE

3.1 Úvod do projektu

Počiatkový stav projektu obsahoval primitívnu projektovú štruktúru, ktorá bola už štandardne vygenerovaná. Prvá štruktúra obsahovala len Java súbory a css bolo hostené pomocou CDN poskytovateľa. Jednalo sa presnejšie o Bootstrap s beta verziou 4. Spočiatku všetko bolo statické, čo sa na stránku s takýmto cieľom nehodilo, tak vznikla potreba projekt vylepšiť. Kapitoly sú zoradené podľa štádií ako nasledovali.

3.2 Výber databázy

Výber databázy bol pomerne náročný keďže ich je celkom veľa. Najväčší hráči pre tento projekt boli PostgreSQL, MySQL, MariaDB a trochu MongoDB. Keď štruktúra databázy bola známa, MongoDB stratil možnosť stať sa projektovou databázou. Ostali už len PostgreSQL, MySQL a MariaDB. PostgreSQL ponúka lepšiu výbavu príkazov, ale z tých projekt nebude ťažiť vďaka Springu s Hibernatom, a preto padlo rozhodnutie na MySQL a MariaDB. Spraviť rozhodnutie medzi týmito dvomi databázami nebolo jednoduché, ale nakoniec zvíťazila MariaDB a jej open source politika.

3.3 Doménový model

Doménový model je skupina tried, ktoré simulujú reálny svet. Spočiatku tieto triedy boli navrhované detailne podľa normálových noriem. To však spôsobovalo problémy s databázou, hlavne pri kaskádovaní, keď sa závislosti rôzne krížili. Došlo ku zjednodušeniu modelu a práca sa uľahčila, modely sa stali stabilnejšie.

3.3.1 Validácia dát

Skoro všetky dáta, ktoré neslúžia ako primárny kľúč, sú validované. Validuje sa minimálne dĺžka stringu, taktiež či nie je prázdny alebo či nie je null.

3.4 Kompilácia a kompilačné nástroje na strane frontendu

Mozgom, ktorý organizuje celý frontend je webpack. Webpack pomocou konfiguračných súborov napísaných v čistom javascripte dokáže manažovať projekt. Konfigurácia musí exportovať json objekt, kde je konfigurácia. V konfigurácii môžu byť definované rôzne parametre ako napríklad, ktoré súbory sa budú kompilovať, ako sa budú volať, generovanie máp, hlavne loadery a mnoho iných parametrov. Medzi loadery použité v projekte patria Babel (*premieňa novší javascript a JSX do staršieho*) a css

loadery. Webpack podporuje aj pluginy, projekt využíva zatiaľ len dva pluginy. CopyWebpackPlugin, ktorý prekopíruje obrázky a Bootstrap súbory do serverového priečinka a CommonsChunkPlugin, ktorý spoločné časti scriptov balí do jedného súboru.

Na kompilovanie scss sa používa sass kompilátor.

3.5 Spravovanie knižníc

O projektové závislosti sa starajú dvaja balíčkoví manažéri.

3.5.1 Maven

Maven funguje na strane Javy. Jeho konfigurácia sa nachádza v súbore pom.xml. Drží projekt pokope tým, že ak nejaký balík napísaný v konfigurácii chýba, tak ho stiahne. Maven ponúka taktiež operácie ako sú napríklad build, clean, install a ich kombinácie. Dokonca je možné vytvoriť si vlastné príkazy.

3.5.2 „npm“ – node package manager

Plní takú istú úlohu ako Maven. Konfiguráciu si ukladá vo formáte json, väčšinou v súbore package.json. V npm existujú dve závislosti:

1. Developerske – pre programátorov
2. Bežné – pre programátora a pre vchod projektu

V npm taktiež môžeme vytvárať vlastné príkazy a potom ich spúšťať pomocou *npm run prikaz*.

3.6 Výber renderovacej knižnice

Rozhodnutie o výbere podnietilo cieľ vytvoriť nekonečné skrolovanie príspevkov, čo by s Thymeleafom nebolo možné, čo je knižnica na renderovanie na serveri. Tu prichádza javascript a jeho schopnosť dynamicky manipulovať dokument, teda stránku. V tejto dobe je na výber príliš veľa možností ako sú napríklad Backbone.js, Vue.js, Angular a React. React sa svojou komunitou a dokumentáciou ukázal ako správna voľba.

3.7 Implementácia dynamického renderovania

Migrácia zo serverového renderovania na klientske renderovanie nebola príliš náročná. React so svojou dokumentáciou a rozvinutou open source komunitou pokrýl

všetky technológie, ktoré stránka potrebovala. Pomocou JSX sa okresal kód na renderovanie zo serverovej časti, ktorý mal zložitejšiu syntax.

3.7.1 Dynamické načítavanie dát zo servera

Po migrácii bolo potrebné implementovať funkcionality, ktorá sa bude zaoberať sťahovaním príspevkov a komentárov, ich vytváraním, mazaním a aktualizovaním. Tieto operácie sú bežné v programovaní, že Spring podporuje CRUD repozitáre, ktoré podporujú základné operácie. Projekt však potreboval tieto operácie poskytnúť vo formáte jsonu, čo je formát, ktorý sa ľahko spracováva programátorsky, ale je aj čitateľný pre človeka. Preto projekt, implementuje Spring Data REST, ktorý sprístupní API a automaticky nám umožní pristupovať k našim modelom cez Ajax.

Dátový formát, ktorý pri prenose projekt používa, je hal+json. Viaz sa na HATEOAS, čo znamená Hypermedia as the Engine of Application State. Dopĺňa klasický json o linky ku dátam API, pri kolekciách obsahuje metadata o strane (*aktuálna strana, veľkosť strany, posledná/prvá strana*) a nakoniec linky ku vnoreným objektom nadradeného objektu. Formát hal+json môže vyzeráť ako je na obrázku nižšie.



Obrázok 1 Ukážka HATEOAS (Brisbin, a iní, 2018)

1. Odkaz na ďalší prvok v kolekcii
2. Odkaz na predchádzajúci prvok v kolekcii
3. Údaj o aktuálnej strane (*čísluje sa od nuly*)

3.7.2 Problémy dynamického načítavania

Po čase, keď sa začali robiť prvé testy na stránke, sa zistilo pár nedostatkov.

3.7.2.1 Problém stránkovania

Problém stránkovania sa dá najlepšie vysvetliť na príklade. Máme dvoch používateľov, používateľ A a B. Používateľ A si v pohode doma prezerá príspevky, zatiaľ čo užívateľ B zmaže príspevok, ktorý si užívateľ A prezerá. Nastane situácia, ktorá je znázornená na diagrame nižšie.



Obrázok 2 Popis stránkovania pri vymazaní elementu

Pred vymazaním všetko funguje správne. Akonáhle užívateľ B vymaže príspevok, čo je znázornené v druhej fáze, nastane posun elementov. Po vymazaní vyzerá stránkovanie ako je znázornené v tretej fáze, štvrtý element je súčasťou prvej strany. V štvrtej fáze dochádza k nahratiu ďalšej strany (*príspevkov*), kde sa ukáže problém. Kvôli posunu sa nenahrá štvrtý element, lebo už nie je súčasťou druhej strany, ale prvej, ktorá nie je aktualizovaná.

Riešením je stránkovací algoritmus nižšie, ktorý spraví prepočty.

```
/**
 * Vypocitaj parametre pre aktualizáciu strany
 * @param {number} currentPage aktualna strana
 */
calculateRefetchPage(currentPage) {
  if (this.deletedItemsCount === 0) {
    throw new Error('Tato metoda by sa mala iba volat, ak doslo k zmazaniu elementov');
  }
  // Vypocitaj kolko elementov sa presunie na predchadzajucu stranu
  const presah = this.deletedItemsCount % this.pageSize;
  // Vypocitaj pocet vymazaných strán
  const removedPages = this.calculateDeletedPages();
  // Vypocitaj, kolko elementov bude aktualizacia ignorovat (elementy, ktore su uz stiahnute)
  const ignoreCount = this.pageSize - presah;
  // Sprav poslednu kalkuláciu čísla aktualizacnej stránky a vrat vysledok
  return new PageResult(currentPage - removedPages, this.pageSize, ignoreCount);
}
```

Obrázok 3 Algoritmus pre výpočet stránkovania

Jedná sa len o podpornú metódu, ktorá je z triedy Paging. Prijíma argument aktuálnej strany a využíva vnútornú premennú `deletedItemsCount`, ktorý sa zvyšuje po vymazaní elementu. Táto hodnota sa dá manipulovať pomocou dvoch metód, ktorý je používateľ knižnice povinný volať sám. Po aktualizácii by mal používateľ resetovať počítadlo vymazaných elementov.

Algoritmus pracuje nasledovne:

1. Vypočíta presah, čo spraví pomocou delenia so zvyškom
2. Vypočíta počet vymazaných strán pomocou vzťahu
$$\text{vymazané strany} = \frac{\text{vymazané elementy}}{\text{veľkosť strany}}$$
3. Zistí počet elementov, ktoré bude ignorovať (pôvodné elementy strany)
4. Vráti výsledok a odstráni strany, ktoré boli vymazané

Maturitná práca používa túto funkciu, keď potrebuje načítať ďalšiu stranu. Počet vymazaných elementov počíta, keď dôjde správa od websocketu.

WebSocket je technológia, ktorá umožňuje serveru posielat' klientovi správy, bez potreby klienta poslať žiadosť. V aplikácii sa zatiaľ využíva len v skupinovej časti, kde sa zobrazujú príspevky. Server posiela len tri druhy správ o akciách ako sú:

1. Delete – vymazanie objektu
2. Update – aktualizovanie objektu
3. New – vytvorenie nového objektu

3.7.2.2 Problém so skrolovaním

Skrolovanie sa zaznamenáva pomocou poslucháča naviazaného na objekt window v javascripte. Problém však nastáva, keď sa používateľ stránky dostane na spodnú časť a začne sa načítavanie ďalších príspevkov. Výkonovo to nie je náročná operácia, ale vo svete webových stránok platí, že malé veci sa nabaľujú. Poslucháč sa volá príliš často pri skrolovaní, a preto bolo potrebné vymyslieť optimálne riešenie.

Prvé riešenie bolo monitorovať výšku, v ktorej prebehla aktualizácia strany. Tolerancia bola okolo 300 pixelov. Avšak toto riešenie sa neujalo, lebo nepočítalo s používateľom, ktorý rýchlo skroluje.

Druhé, oveľa premyslenejšie riešenie sa podobá prvému s rozdielom, že malo vbudovaný časovač a používalo relatívne jednotky dĺžky. K načítavaniu nového obsahu došlo, keď sa užívateľ dostal na dolnú štvrtinu stránky. Nastavila sa nová hodnota časovača a v komponente sa zmenil stav na načítavanie.

Avšak ani druhá metóda nie je perfektná. Vo finálnej fáze vývoja sa použila iná technika a tá bude vysvetlená v kapitole 3.10.1.

3.7.3 CRUD metódy

Serverové API, ktoré sa nachádza na endpointe /api, nám umožňuje vytvárať, aktualizovať a vymazávať elementy z databázy. Ak chceme vykonať nejakú z vymenovaných akcií, musíme správne určiť http metódu a cestu k zdroju v API. Tých ciest je príliš mnoho, pomocou HAL browsera sa dajú preskúmať, ale vymenujeme si aspoň http metódy a ako ich použiť.

1. GET – vyžiadame si dáta od API (*napríklad /api/articles/1*)
2. POST – vytvoríme nový objekt, musíme priložiť aj telo s parametrami
3. PUT – nahradíme určitý objekt, ktorý určíme v tele
4. PATCH – aktualizujeme zvolený objekt
5. DELETE – zmažeme objekt v API

Maturitná práca využíva všetky akcie okrem PATCH.

3.7.3.1 Kolízie

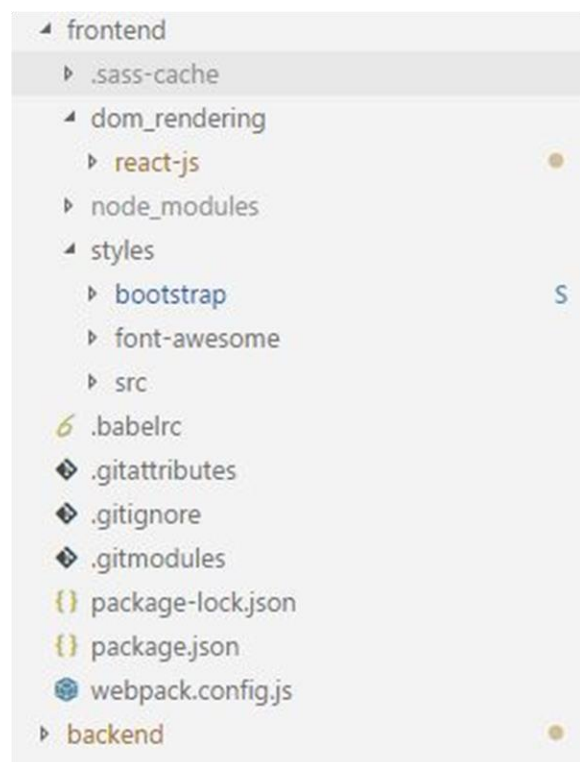
Pri manipulovaní s dátami môže dôjsť ku kolíziám. Práca zatiaľ tieto kolízie nekontroluje, keďže iba autor príspevkov môže meniť dáta. Jediný problém môže nastať pri administrácii skupín. V nasledujúcej dobe sa nechystá update, ale problém by sa dal vyriešiť pomocou http hlavičky E-Tag, ktorá slúži podobne ako hash. Ak by niekedy došlo k aktualizácii v tejto oblasti, aktualizácia by bola zabalená aj s knižnicou.

3.8 Vzhľad stránky a veľká zmena v organizovaní súborov

Stránka bez vlastného css vyzerala ako klasická bootstrapovská stránka. Preto sa začala vyvíjať vlastná verzia bootstrapu pomocou git submodulu. Git je program, ktorý pomáha tímom spolupracovať na projekte a uchováva zmeny vo verziách programu. Kedykoľvek sa developer môže vrátiť ku commitu (verzii) a upraviť ju. Git nesúvisí s touto prácou, a preto nebude ďalej rozoberaný.

3.8.1 Modifikácia Bootstrapu

Aktuálna verzia projektového Bootstrapu je 4 a nelíši sa veľmi od jadra. Boli len pridané pravidlá pre tabuľky, zmenené premeny farieb, aby stránka vyzerala krajšie a nakoniec boli upravené aj formuláre.



Obrázok 4 Súčasná projektová štruktúra

3.8.2 Reštrukturalizácia projektu

Po integrovaní Bootstrapu do projektu nastali veľké problémy pri kompilácii. Jednak, kompilácia Bootstrap je pomerne zložitá, ktorú je možné nájsť na ich oficiálnej stránke, a keď sa k tomu pridá kompilácia vlastného scss a javascriptu, nastane chaos. Výsledkom týchto komplikácií bolo rozdelenie celkového projektu na tri podprojekty. Serverová časť, teda Spring tvoril samostatnú časť. Javascript, presnejšie React, taktiež

tvorí samostatný celok. Nakoniec sú tu kaskádovité štýly, vlastné scss spolu s Bootstrapom. Súborová štruktúra je znázornená na **Obrázok 4**.

3.8.3 Štýl stránky

Bootstrap dopĺňa css, ktoré mierne mení štandardné správanie Bootstrapu. Jedná sa hlavne o responzívny dizajn, kde sa mení pozadie kvôli lepšej viditeľnosti. Ďalej definuje štýl pre niektoré React komponenty, ako sú napríklad modálne formuláre, potvrdzovacie okná a komponent SplitDiv, ktorý rozdeľuje prvky na pravú a ľavú stranu.

3.8.4 Navigácia

Pôvodne navigácia obsahovala vyhľadávacie pole, ale to bolo presunuté na spodok stránky. Logo je vektorový obrázok vo formáte svg.

3.9 Administrátorské rozhranie

Globálne administrátorské rozhranie zatiaľ nebude dostupné z dôvodu, že nemá zmysel vytvárať také rozhranie s takým modelom. Miesto toho sa odporúča použiť rozhranie pre skupiny, ktoré je dostupné na lokalite `/group/{groupName}/admin`.

3.9.1 Pravidlá

Každá skupina má vlastný systém, pričom globálny admin má prístup do každej skupiny bez toho, aby to administrátori skupiny vedeli. Majiteľ skupiny môže byť len jeden, ak by nastal problém, že majiteľ nebude naďalej aktívny, globálny admin má právo zvoliť nového majiteľa. Administrátori skupiny môžu prideliť len práva čítania a zapisovania. Majiteľ môže určovať administrátorov.

3.9.2 Rozhranie

Administrátorské rozhranie podporuje vyhľadávanie používateľov podľa mena, priezviska a privilégií. Má to však jednu vadu, že globálne vyhľadávanie ešte nie je podporované.

Dáta sú stránkované podobne ako v kapitole 3.7.1 o dynamickom nahrávaní dát, avšak s tým rozdielom, že tentokrát sa dáta stránku až po zadaní jedného z parametrov. Tentokrát sa dáta neudržiavajú. To znamená, že pri každom nahrávaní inej stránky, pôvodné dáta sa zmažú z pamäte. Takto sa eliminoval problém stránkovania a nechal sa priestor pre potenciálnu implementáciu E-Tagov.

Parametre aktuálnej strany sa menia pomocou dvoch polí, ktoré nastavujú veľkosť strany a číslo strany.

3.10 Prehliadač skupín

Prehliadač skupín umožňuje užívateľovi prehliadať jeho členstvá skupinách a prijímať alebo odmietat' pozvánky do skupín.

3.10.1 Dynamické načítavanie pomocou React Virtualized

Oproti ostatným riešeniam dynamického nahrávania spomenutým v kapitole 3.7.2.2, riešenie tejto knižnice je najefektívnejšie. Najhlavnejším rozdielom je, že knižnica React Virtualized renderuje iba dáta, ktoré sú potrebné a viditeľné na stránke. Taktiež ich aj načítavajú dopredu, aby sa čo najviac zmenšil efekt načítavania stránky (*po vizuálnej stránke*). Dáta pri prehliadači skupín na rozdiel od administrátorského rozhrania v pamäti zostávajú a nemažú sa. Jedinou nevýhodou oproti predchádzajúcim riešeniam je obmedzená možnosť konfigurovať štýly elementov v liste, čo je objekt knižnice React Virtualized.

3.10.1.1 Obmedzenia

Obmedzení nie je veľa, ale sú ovplyvnené tie najhlavnejšie. Medzi najhlavnejšie patria rozmery elementu (*šírka a výška*). React virtualized ponúka dve možnosti ako sa vysporiadať s týmto problémom.

3.10.1.1.1 Pevne pridelená dĺžka

Pevne pridelená dĺžka je najľahšou, ale súčasne najmenej praktickou možnosťou. Táto možnosť by sa napríklad hodila pri liste, kde riadky sú rovnaké a presne definované. Táto možnosť v programe nebola využitá.

3.10.1.1.2 Dynamicky pridelená dĺžka

Pri tvorbe web stránok, skoro nikdy nepoznáme presné rozmery elementu. React Virtualized prichádza s riešením v podobe špeciálneho komponentu, ktorý sa volá CellMeasurer. Komponent vyrenderuje náš prvok, ktorý chceme zmerať mimo dohľadu užívateľa. Na tomto je postavený prehliadač skupín.

3.10.2 Meranie riadkov v liste

Ako bolo spomenuté v kapitole 3.10.1.1.2, prehliadač skupín využíva CellMeasurer. CellMeasurer ako samostatný merací komponent nestačí, keďže riadok

môže obsahovať viacero elementov, ktoré pravdepodobne nemajú rovnakú výšku. Preto skupinový prehliadač používa ešte jeden komponent, ktorý vyrovnáva výšky. Je umiestnený ako prostredník medzi riadkom a elementami v riadku. Označuje sa ako kontajner. Radšej sa presuňme ku schéme v **Príloha A**, ktorá to vysvetlí graficky lepšie.

Na rade je vysvetliť algoritmus, ktorý je na obrázku nižšie. Začneme zdola, teda v GroupCard.

```
// Zavola sa po vyrenderovaní komponentu
componentDidMount() {
  // Posli vysku GroupListu
  this.props.updateParent(this.el.offsetHeight, this);
}

// Vola sa pri každej aktualizácii
componentDidUpdate() {
  // Ak nahodou nemame referenciu na element, nevykonaj žiadnu akciu
  if (!this.el) {
    return;
  }
  // Ak referencia existuje, element aktualizuje GroupList s aktuálnou výškou
  this.props.updateParent(this.el.offsetHeight, this);
}
```

Obrázok 5 Meranie výšky v GroupCard

Začneme s prvou metódou, ktorá sa volá `componentDidMount`. Zavolá sa po úspešnom vyrenderovaní komponentu a jej úlohou je poslať výšku nadradenému komponentu, ktorý sa volá `GroupList`. Pod ňou sa nachádza metóda `componentDidUpdate`, ktorá zavolá po úspešnom aktualizovaní komponentu. Aktualizácia v Reacte sa vykoná iba, ak sa zmenia vlastnosti komponentu (*props*) alebo sa zmení stav (*state*). Špecificky v tejto metóde sa taktiež aktualizuje nadradený komponent, spolu s tým prebehne aj kontrola, či je to možné.

Komponent `GroupCard` poslal svoju výšku nadradenému komponentu, takže sa presunieme teraz k nadradenému komponentu. Kód pre jeho veľkosť sa nachádza v **Príloha B**.

Celý princíp kódu spočíva v tom, že komponent poslúcha na aktualizácie `GroupCard` komponentov, a keď zaznamená taký počet odlišných aktualizácií, koľko je

stĺpcov, potom prebehne kontrola. Kontrola rozhodne, či sa bude opäť prepočítavať výška riadka, teda kontajnera, alebo sa nič nevykoná.

Na poslednej vrstve sa nachádza najmenej sofistikovaná metóda, ktorá je znázornená na obrázku nižšie. Je to najhlavnejšia metóda, keďže udáva veci do pohybu. Jej jedinou úlohou je vyčistiť pamäť cache a znova prepočítať list.

```
/**
 * Prepocita list a vycisti cache pamat
 * @param {number} index index riadka
 */
this.recomputeHeight = (index) => {
  return () => {
    cache.clear(index);
    this.list.recomputeRowHeights(index);
  }
}
```

Obrázok 6 Metóda na prepočítavanie výšky v *GroupBrowseri*

3.11 REST

Štandardná konfigurácia REST rozhrania od Spring Data REST nestačila, tak bolo potrebné dorobiť pár endpointov.

3.11.1 ResourceAssembler

Abstraktná trieda, ktorá sa podieľa na vytvorení json podľa HATEOAS. Projekt obsahuje modifikovanú abstraktnú triedu, ktorá zjednodušuje vytváranie odkazov k prostriedkom. Hlavným cieľom assemblera je premeniť ResourceSupport objekt do hal+json formátu.

3.11.2 Objekty typu ResourceSupport

Reprezentujú dáta v jsone a nejedná sa o nič iné ako objekt s atribútmi, gettermi a settermi. Takéto objekty sa označujú tiež ako aj POJO.

3.11.3 Kontroléry

Doplňujú Spring Data REST o ďalšie endpointy. Oproti normálnym kontrolérom sa nelíšia.

3.11.4 Vlastná implementácia Pageable

Pageable je Java interface, ktorý limituje počet prvkov, ktoré sa vyberú z databázy. Avšak jeho štandardná implementácia dobre nespôlupracuje s knižnicou React Virtualized, vznikla potreba implementovať vlastnú verziu. Vlastná verzia nestránkuje, ale slúži ako náhrada limitu a offsetu v MySQL dialekte.

3.12 Bezpečnosť

Spring Security poskytuje možnosť zabezpečiť metódu pred neautorizovaným užívateľom. Medzi najviac zabezpečené objekty patria repozitáre, ktoré slúžia na prístup k databáze a niektoré kontroléry ako napríklad kontrolér Administrátorského rozhrania. Bezpečnosť sa aplikuje pomocou anotácií PreAuthorize, PostAuthorize, PreFilter, PostFilter, kde najbežnejšou anotáciou je PreAuthorize, ktorý kontroluje prístup pred vyvolaním metódy. Anotácia prijíma jeden argument value, ktorý musí byť typu string a musí byť platným SpEL výrazom. SpEL výraz je interpretovaný skript a musí vracať true alebo false.

3.13 Nasadenie servera do produkcie

Skompilovanie klientskej časti je najľahšie. Treba pozmeniť len parameter, ktorý prepne mód na produkčný. Na strane servera je potrebných viac akcií k dosiahnutiu cieľa. Treba si vybrať web server a operačný systém, na ktorom bude spozajzdený. Taktiež je možné použiť virtualizáciu alebo kontajner. Záleží na prostredí, na ktorom bude server bežať a od potrieb projektu.

4 VÝSLEDKY PRÁCE

Jadro servera spĺňa všetky ciele, ktoré boli stanovené. Najhlavnejším cieľom bolo spraviť čo najviac generické jadro, čo v praxi znamená, že server bude môcť byť použitý na akúkoľvek jednoduchú sociálnu sieť. Jedine, čo by sa dalo podotknúť, sú chýbajúce unit testy, ktoré sú užitočné tým, že garantujú správnosť fungovania servera a tým potenciálne skracujú čas pri debugovaní.

Frontend splnil základné požiadavky na prezentáciu projektu, ale má určite vysoký potenciál. Práca bola primárne zameraná na REST API a jej univerzálne použitie. Json patrí medzi formáty, ktoré sa ľahko spracovávajú a spolu s API dokážu poskytovať dáta akejkolvek aplikácií, nie len napísanej v javascripte. To je tá sila projektu, poskytnúť rozhranie širokej škále aplikácií, ktoré potom môžu byť naprogramované pre Android, iOS, Windows, Linux Desktop a MacOS.

Na prezentáciu API sa použil React z dôvodu, že to bola knižnica s bohatou dokumentáciou. Po dobe, keď sa projekt rozrástol som zistil, že javascript sa stáva menej prehľadným a začína to byť chaotické. Preto by som v budúcnosti namiesto Babelu radšej použil Typescript, ktorý pridáva dátové typy do javascriptu. Ďalším zistením je, že nie je dobré vytvárať React komponenty, ktoré modifikujú svoje podradené komponenty rekurzívne. Sťažuje to knižnici rozhodovanie, kedy sa má komponent aktualizovať, čím sa znižuje aj výkon aplikácie.

Čo sa týka zistení na strane Springu, databázy by nemali byť príliš komplikované, lebo to spôsobuje dve veci: znižuje efektivitu dotazov a spôsobuje príliš náročnú konfiguráciu kaskádovania entít. Ďalším zistením je nepoužívať Stream API s repozitármi pri limitovaných dotazoch. Domnienka, že sa jedná o lenivú operáciu, nebola správna.

Pri definovaní nového REST API endpointu treba uviesť celú cestu pre kontrolér, aby sa renderovali odkazy na metódu kontroléra správne. Spring Data REST nedokáže stránkovať doplnkové funkcie prostriedku, preto bolo nutné vytvoriť nový kontrolér.

Ak chceme premenovať meno objektu („*resource*“), musíme definovať anotáciu Relation. Táto anotácia nie je súčasťou Spring Data REST, ale Spring HATEOAS.

Na záver prikladám pár ukážok stránky a jej vývoja v prílohách **Príloha C**, **Príloha D**, **Príloha F** a **Príloha E**.

ZÁVER

Vytvorením programu v rámci tejto zadanej práce boli splnené všetky ciele definované v kapitole **CIELE PRÁCE**.

Poznatky získané autorom budú využité v praxi i v ďalšom štúdiu za predpokladu, že nenastane nejaká veľká zmena v IT svete.

Práca by sa dala využiť s úpravami aj ako blog (*príspevky podporujú markdown*), menšia sociálna sieť, pri menšej zmene vzhľadu by to mohlo byť aj fórum. Ďalej sa dá pristupovať aj k dátam členom skupiny, takže sa dá vytvoriť aj appka na určenie týždenníkov. API je otvorené pre všetkých členov. Jediným limitom sú len dáta v API.

ZOZNAM POUŽITEJ LITERATÚRY

Brisbin, Jon, Gierke, Oliver a Turnquist, Greg. 2018. Spring Data REST - Reference Documentation. *spring.io*. [Online] 19. Február 2018. [Dátum: 27. Február 2018.] <https://docs.spring.io/spring-data/rest/docs/current/reference/html/>.

Facebook Inc. React - A JavaScript library for building user interfaces. *React*. [Online] [Dátum: 26. Február 2018.] <https://reactjs.org/>.

—. Tutorial: Intro To React - React. *React*. [Online] [Dátum: 26. Február 2018.] <https://reactjs.org/tutorial/tutorial.html>.

Johnson, Rod a Hoeller, Juergen. Annotation Type Repository. *spring.io*. [Online] [Dátum: 26. Február 2018.] <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Repository.html>.

Oracle. Lesson: Annotations (The Java™ Tutorials > Learning the Java Language). *Oracle*. [Online] 1.8. [Dátum: 25. Február 2018.] <https://docs.oracle.com/javase/tutorial/java/annotations/index.html>.

Pivotal Software, Inc. 2018. Core Technologies. *spring.io*. [Online] 19. Február 2018. [Dátum: 26. Február 2018.] <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-stereotype-annotations>.

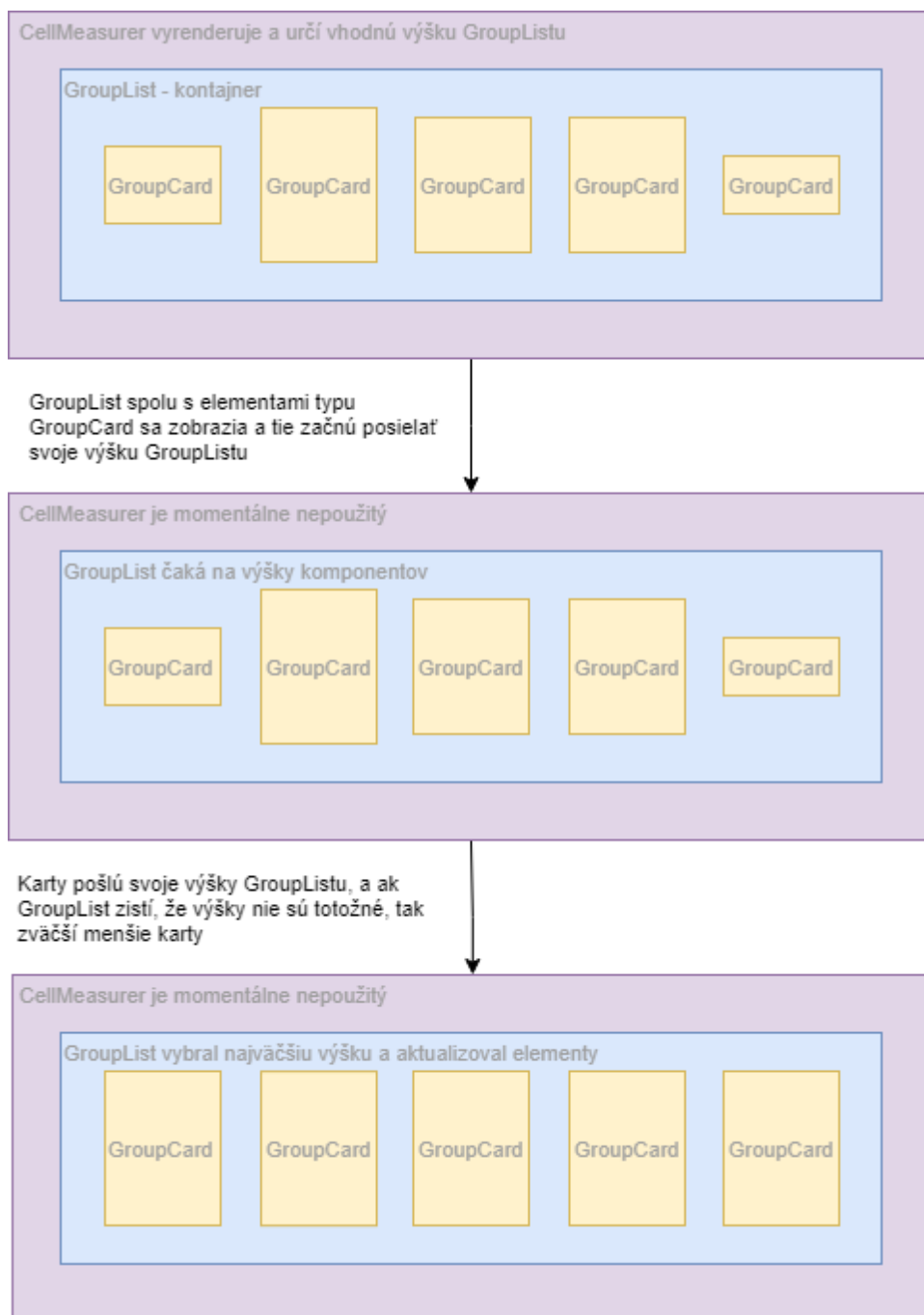
—. **2018.** Web on Servlet Stack. *spring.io*. [Online] 19. Február 2018. [Dátum: 26. Február 2018.] <https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-controller>.

Sun Microsystems Inc. 1997. JavaBeans(TM) Specification 1.01 Final Release. *Oracle*. [Online] 1.01, 8. August 1997. [Dátum: 25. Február 2018.] <http://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/>.

ZOZNAM PRÍLOH

<i>Príloha A</i> Schéma dynamického merania	32
<i>Príloha B</i> Zdrojový kód GroupListu, pre výpočet výšok	32
<i>Príloha C</i> Stránka ešte renderovaná pomocou Thymeleafu (17.12.2017)	32
<i>Príloha D</i> Renderovanie pomocou Reactu (28.1.2018)	32
<i>Príloha E</i> Prehliadač skupín 1.3.2018	32
<i>Príloha F</i> Stránka pred odovzdaním 1.3.2018	32

Príloha A Schéma dynamického merania



Príloha B Zdrojový kód GroupListu, pre výpočet výšok

```
/**
 * Vratí handler pre aktualizovanie vysok
 * @param {number} index index stĺpca v riadku
 */
updateHeight(index) {
  // Vytvorí anonymnú metodu
  return (height) => {
    // Vyberieme si určité hodnoty z vlastností komponentu
    const { recomputeHeight, memberships } = this.props;


    // Porovná a vyberie väčšiu výšku
    this.maxHeight = Math.max(this.maxHeight, height);
    // Zaznamená výšku
    this.updates[index] = height;


    // Pomocné premenné
    const updateCount = Object.keys(this.updates).length;
    const heights = Object.values(this.updates);

    // Podmienka, ktorá sa splní:
    // ak doslo k počtu aktualizácii koľko je stĺpcov
    // a
    // ak nejaka výška komponentu sa nezhoduje s najväčšou
    if (updateCount === memberships.size && heights.some(height => height !== this.maxHeight)) {
      // Nastaví štýl, presnejšie výšku pre komponenty
      this.style = { height: this.maxHeight };
      // Spustí prepocítavanie výšky v rodičovi
      this.props.recomputeHeight();
      // Vycistí záznam aktualizácii
      this.updates = {};
    }
  }
}
```

SPŠ J. Murgaša

Search





4A

This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.

Clanok cislo 0.

sample subheader od [Superuser](#)

#####Lorem ipsum dolor

sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit. Lorem ipsum dolor *sit amet*, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit.#####Lorem ipsum dolor

sit amet, consectetur adipiscing elit. Praes

Delete

Edit

Like

Citat dalej...

Clanok cislo 1.

sample subheader od [Superuser](#)

#####Lorem ipsum dolor

sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit. Lorem ipsum dolor *sit amet*, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit.#####Lorem ipsum dolor

sit amet, consectetur adipiscing elit. Praes

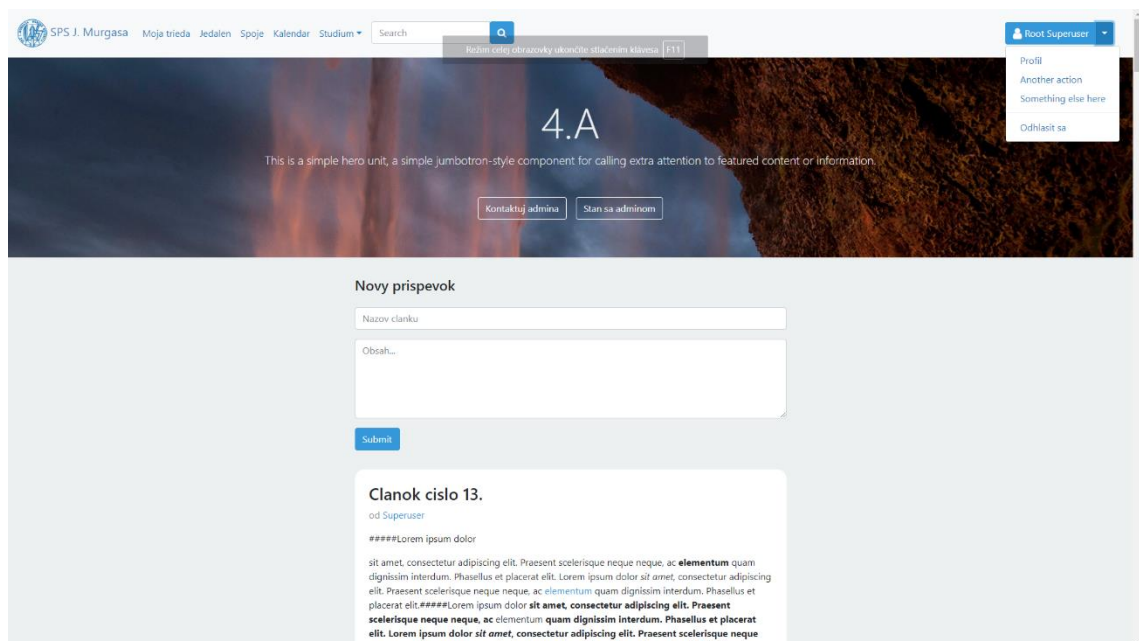
Delete

Edit

Like

Citat dalej...


Príloha D *Renderovanie pomocou Reactu (28.1.2018)*



Príloha E Prehliadač skupín 1.3.2018

SPS J. Murgasa Moja trieda Skupiny					Root Superuser	
SPS/JM	Skupina 1 Test description number 1	Skupina 2 Test description number 2	Skupina 3 Test description number 3	Skupina 4 Test description number 4		
Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť		
Skupina 6 Test description number 6	Skupina 7 Test description number 7	Skupina 8 Test description number 8	Skupina 9 Test description number 9	Skupina 10 Test description number 10		
Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť		
Skupina 11 Test description number 11	Skupina 12 Test description number 12	Skupina 13 Test description number 13	Skupina 14 Test description number 14	Skupina 15 Test description number 15		
Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť		
Skupina 16 Test description number 16	Skupina 17 Test description number 17	Skupina 18 Test description number 18	Skupina 19 Test description number 19	Skupina 20 Test description number 20		
Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť		
Skupina 21 Test description number 21	Skupina 22 Test description number 22	Skupina 23 Test description number 23	Skupina 24 Test description number 24	Skupina 25 Test description number 25		
Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť	Pozrieť Opustiť		

Príloha F Stránka pred odovzdaním 1.3.2018

 SPS J. Murgasa

Moja trieda

Skupiny

Admin

Root Superuser

4.A

Testovacia moja trieda

Názov článku *

minimálne 5 znakov

Obsah *

minimálne 5 znakov

Odoslať

Zrušiť

Článok číslo 12.

od Superuser

####Lorem ipsum dolor
sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit.####Lorem ipsum dolor
sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent scelerisque neque neque, ac **elementum** quam dignissim interdum. Phasellus et placerat elit.####Lorem ipsum dolor