VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE

Brno, 2016

Jméno autora

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

NÁZEV PRÁCE ČESKY

THESIS TITLE IN ENGLISH

SEMESTRÁLNÍ/DIPLOMOVÁ PRÁCE

SEMESTRAL/DIPLOMA THESIS

AUTOR PRÁCE
AUTHOR

Jméno autora

VEDOUCÍ PRÁCE
SUPERVISOR

Jméno vedoucího práce

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Jan Novák ***ID:*** 999999
Ročník: 2 ***Akademický rok:*** 20xx/xx

NÁZEV TÉMATU:

Úplný název zadání diplomové práce

POKYNY PRO VYPRACOVÁNÍ:

- Seznamte se s problematikou.
- Navrhněte metody řešení.
- Vybrané metody vyzkoušejte.
- Dosažené výsledky průběžně konzultujte s vedoucím práce.
- Porovnejte jednotlivé metody mezi sebou.
- Porovnejte výsledky simulace s výsledky naměřenými na reálných sy
- Dosažené výsledky zhodnoťte

DOPORUČENÁ LITERATURA:

[1] CHRISTIANSEN, Donald. *Electronics engineers' handbook*. 4th ed. New York: McGraw Hill, 1997. ISBN 0070210772.

Termín zadání: 19.9.201x ***Termín odevzdání:*** x.x.201x

Vedoucí práce: prof. Ing. Jiří Novotný, Ph.D.
Konzultant semestrální práce:

doc. Ing. Josef Nový, CSc., *předseda oborové rady*

PROHLÁŠENÍ

Prohlašuji, že svou semestrální práci na téma „Interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl“ jsem vypracoval samostatně pod vedením vedoucího semestrální práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené semestrální práce dále prohlašuji, že v souvislosti s vytvořením této semestrální práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

Obsah

Úvod **9**

1 Teorie webových aplikací **11**
1.1 Full-stack aplikace . . . 11
1.1.1 Editor kódu . . . 11
1.1.2 HTML . . . 11
1.1.3 CSS . . . 13
1.1.4 JavaScript . . . 13
1.1.5 DOM . . . 14
1.1.6 Knihovny rozšiřující JavaScript . . . 15
1.2 Back-end . . . 16
1.2.1 Node.js . . . 16
1.2.2 HTTP Protokol . . . 17
1.2.3 REST API . . . 17
1.2.4 Express . . . 18
1.2.5 AJAX . . . 18
1.2.6 CORS . . . 18
1.2.7 Objektově relační mapování . . . 18
1.3 Front-end . . . 19
1.3.1 React . . . 19
1.3.2 Virtual DOM . . . 19
1.3.3 JSX . . . 19
1.3.4 Axios . . . 20
1.3.5 Bootstrap a Reactstrap . . . 20
1.4 Databáze . . . 20
1.4.1 MariaDB . . . 20
1.4.2 Dostupné databáze s otagovanými obrázky a videi . . . 21
1.5 Možnosti ovládání aplikace pomocí interaktivních ovladačů . . . 21

2 Postup vytvoření vlastní aplikace **23**
2.1 Back-end a databáze . . . 23
2.1.1 Příprava pro vlastní implementaci . . . 23
2.1.2 Spuštění jednoduchého back-end serveru . . . 23
2.1.3 Databázové modely . . . 24
2.1.4 Zdrojová data pro databázi . . . 25
2.1.5 Úprava zdrojových dat . . . 26
2.1.6 Vytvoření Databáze . . . 27

2.1.7 Skript pro zpracování souborů do databáze 27
2.1.8 Funkce main() 28
2.2 Front-end 29
2.2.1 Vytvoření nové aplikace v knihovně React 29
2.2.2 Struktura hlavního souboru 29
2.2.3 Optimalizace 30
2.2.4 Komponenty 30
2.3 Grafika 32

Závěr **33**

Literatura **35**

Seznam příloh **39**

A Médium **41**

Úvod

Webové aplikace jsou součástí každodenního života většiny moderní populace a slouží k nejrůznějším účelům od přehrávání videí až po nakupování v e-shopech. Tento stále se rozvíjející obor se díky ohromné poptávce ve věku internetu stále rozčleňuje na další a další specializace, vytváří nové pracovní příležitosti a s tím i důvody, proč se oborem zajímat jakožto kariérní možností. Tato práce se povrchově věnuje všem oblastem vývoje webové aplikace pro účel zpracování obrazových a audiovizuálních děl do formy interaktivního archivu, ve kterém je možné dle různých kritérií filtrovat výsledky vyhledávání. Aby bylo možné tato díla převést do webové aplikace, je nutné je systematicky někam ukládat jako data, která jsou následně zpracována a vyobrazena uživateli. Celý tento proces je možné řešit mnoha způsoby, z nichž jeden je popsán právě v této semestrální práci.

1 Teorie webových aplikací

1.1 Full-stack aplikace

Pro úspěšné vytvoření webové aplikace je nutné pochopit celý proces jejího vývoje. Webová aplikace se všemi funkčními komponenty se označuje jako "Full-stack aplikace". Jedná se o rozsáhlé téma, které je možné rozdělit do několika kategorií popisujících určitou oblast jejího vývoje. Programátor, který rozumí všem částem procesu tvorby webových aplikací se označuje jako „Full-stack developer". Nejzákladnější rozdělení v praxi využívá čtyř oblastí:

- Back-end
- Front-end
- Databáze
- DevOps

Back-end označuje veškeré procesy probíhající na straně serveru. Předmětem back-endu je logika a zpracování dat. Front-end označuje veškeré procesy odehrávající se na straně klienta. Předmětem front-endu je vzhled, viditelný obsah a struktura webové aplikace. Zatímco front-end využívá standartně tři jazyky: HTML, CSS a JavaScript, pro řešení back-endu se nachází velké množství alternativ, ze kterých je nutné si vybrat. Jednou z možností je využití Node.js, umožňující psaní back-endu v jazyku JavaScript. V databázi jsou uschována data, které jsou využívány webovou aplikací. Oblast DevOps se poté zabývá nasazováním aplikace na server.

1.1.1 Editor kódu

Jako software pro zpracování kódu byl použit Visual Studio Code. Jeho výhodou je kromě atraktivního vzhledu také podpora velkého množství rozšíření pro nejrůznější programovací jazyky, jako jsou např. našeptávače a nástroje pro formátování zdrojového kódu. Kromě toho je v něm velmi pohodlná správa verzování systémem Git a za zmínku stojí i možnost uzpůsobení si vzhledu podle představ uživatele. Do editoru je možné nahrát velké množství nejrůznějších rozšíření pro prakticky jakýkoliv jazyk.

1.1.2 HTML

Hypertext Markup Language (HMTL) je značkovací (tagovací) jazyk, pomocí kterého se vytváří webové stránky. Označení "hypertext"znamená, že jednotlivé stránky jsou propojeny pomocí odkazů. Jedná se o standardní jazyk podporovaný všemi běžnými prohlížeči a vývojář je díky němu schopen říct, jakým způsobem má daný prohlížeč zpracovávat a formátovat text.

Pomocí elementů, které HTML poskytuje, je vývojáři umožněno pracovat s velikostí textu a rozhodnout, zdali bude vypsán tučně či kurzívou, importovat do dokumentu obrázky a animace ve formátu GIF, vytvářet formuláře, rámečky a tabulky. Na pozadí stránky a do tabulek je možné pomocí HTML přidat i barvu. Nevýhodou HMTL je, že se jedná o jazyk statický, a tudíž není možné v něm vytvořit např. vysouvací menu a další, běžně používané elementy. Tagy (značky) využívají v jazyce HTML syntax <tag></tag>, kde "/"označuje konec působiště daného tagu. Tyto značky lze podle jejich významu rozdělit na tři skupiny - strukturální, popisné a stylistické.[1]

Zdrojový kód HTML dokumentu je vždy obalený ve značce <html> - kořenovém elementu. Každý HTML dokument obsahuje alespoň dva elementy. Tím jsou elementy <head> a <body>. V elementu <head> - hlavičce souboru - jsou obsaženy informace o dokumentu, které se přímo v prohlížeči nezobrazují. V elementu <body> - těle dokumentu - se téměř vždy nachází většina kódu. Na začátku každého html souboru bývá standartně deklarován typ dokumentu pomocí komentáře <!DOCTYPE html>. V dokumentu je nutné dodržovat tzv. nesting, kdy vnořený element musí mít svůj začátek i konec mezi začátkem a koncem tagu vnějšího. HTML disponuje mnoha tagy, které plní různé funkce. Mezi ty nejzákladnější patří:

- <h1>(až <h6>) - velikost nadpisu
- <p> - odstavec
- - oddíl
- - úsek textu
- - tučný text
- - kurzíva
- - odrážkový seznam
- - číslovaný seznam
- - položka seznamu
- <a href> - hyperlinkový odkaz
- <img> - obrázek
- <table> - tabulka
- <form> - formulář
- <button> - tlačítko

Jazyk HTML má ve světe IT již dlouhou historii, neboť jeho oficiální vydání se datuje k roku 1990. Mezi další významné průlomy tohoto období lze zařadit protokol HTTP a WorldWideWeb. V současné době je nejnovější verzí HTML 5.3. Kromě HTML existuje i jazyk XHTML, kde význam písmena X je slovo "extensible"- rozšiřitelný. Mělo se jednat o nástupce jazyka HTML, reálně však svého předchůdce z trhu nevytlačil. [2]

1.1.3 CSS

Zkratka CSS znamená Cascading StyleSheets - kaskádové styly. Důvodem vytvoření CSS bylo oddělení správy vzhledu stránky od jejího obsahu (HTML). Sám o sobě je soubor s příponou css nepoužitelný, jeho funkce se projevují až v souboru HTML. Existují tři způsoby deklarace CSS atributů:

- přímý styl (inline) - využívá syntax style="..."
- stylesheet (internal) - pravidla pro formátování textu jsou zapsána do hlavičky dokumentu mezi tagy <style></style>
- externí stylesheet (external) - připojení souboru s příponou css, na který odkazuje tag <link>

Pomocí CSS je uživatel schopen upravovat styly prvků webové stránky či aplikace. Těmito prvky jsou např. fonty, barvy a velikost mezer mezi elementy. Pomocí CSS specifikujeme pravidla, podle kterých se bude daný prvek renderovat. Každý element se skládá ze dvou částí a jejich syntax je následující { atribut: hodnota } [3]

Příklady atributů CSS:

- Background-(color, image, size,...) - mění vlastnosti pozadí
- Border-(collapse, color, style, width...) - mění vlastnosti rámečků
- Color - mění barvu libovolného prvku
- Font-(family, size, style, weight,...) - mění vlastnosti písma
- Margin - šířka vnějšího okraje prvku
- Opacity - průhlednost
- Padding - šířka vnitřního okraje prvku
- Text-(align, indent, shadow,...) - mění vlastnosti textu

Jazyk CSS lze využívat od roku 1996. Od roku 2005 je standardní verzí CSS3.[4]

1.1.4 JavaScript

JavaScript je vysokoúrovňový, objektově orientovaný programovací jazyk. Obecně platí, že pomocí HTML se řeší obsah a informace, CSS slouží k úpravě vzhledu stránky a JavaScript určuje chování webové stránky či aplikace. V jeho definici se často objevuje i slovo "klientský", neboť běh programu napsaného v JavaScriptu se odehrává na straně klienta, a to ve webovém prohlížeči po stažení aplikace. JavaScript je ve výchozím stavu asynchronní, což znamená, že kód nečeká na dokončení kódu předchozího, a rovnou pokračuje v dalších úkonech. Také se jedná o jednovláknový jazyk, tudíž kód probíhá postupně za sebou, ze shora dolů.[5]

1.1.5 DOM

DOM je rozhraní, které využívají prohlížeče. Využívá k tomu uzly (node), což jsou v podstatě HTML elementy. DOM samotný reprezentuje stromovou strukturu těchto elementů. Lze jej využít také pro reprezentaci dokumentů XML a XHTML. Chová se jako API, tudíž umožňuje programům číst a manipulovat obsahem, strukturou a styly dané webové aplikace. U náročných aplikací mohou s DOM nastat problémy s rychlostí, neboť pracuje s reálnými komponenty webové aplikace.

ECMAScript

Aby bylo možné pochopit význam vzniku ECMAScriptu, je nutné uvést několik událostí v historii vývoje JavaScriptu. JavaScript byl vytvořen v roce 1995 společností Netscape a prošel si několika změnami jména. JavaScript se jako jméno ustálilo z marketingových důvodů, neboť Java byl v té době velmi populární jazyk. Netscape využíval JavaScript ve svém prohlížeči Netscape Navigator, který byl ve své době na trhu dominantním prohlížečem. Problém nastal, když se firma Microsoft rozhodla prorazit na trh s vlastním prohlížečem, který využíval obdobu JavaScriptu zvanou JScript. Tyto jazyky sice sdílely kompatibilní jádro, rozšířené schopnosti ale ucelené a sjednocené nebyly a tím vznikala spousta problémů jak na straně tvůrců stránek, tak na straně návštěvníků. Řešením byla standardizace asociací ECMA, jejíž výstupem byla první verze jazyka "ECMAScript". [6] S rozvojem webových aplikacích rostla také nutnost JavaScript postupně aktualizovat a přinášet nové nástroje. Od roku 2015 vychází každý rok nový update, přinášející nové funkce a vylepšení. Z těchto nových funkcí za zmínku stojí:

- ECMAScript 2015
  - Arrow funkce
  - Třídy (Class)
  - Import/Export
  - Konstanty (deklarace pomocí klíčového slova "const")
  - Promises
  - template literal
- ECMAScript 2016
  - Destrukturalizace
  - Klíčová slova async/await
- ECMAScript 2017
  - Třítečkový operátor

JSON

JavaScript Object Notation (JSON) je velmi populární formát reprezentující data, využívaný zejména pro API a konfigurační soubory. JSON soubor obsahuje "zprávu", jejíž "poslem"je API. Očekávaný výstup souboru JSON často bývá objekt, možností je však více. Klíče a hodnoty jsou v JSON souboru vždy v uvozovkách. Pro zpracování souborů ve formátu JSON je nutné jej prvně rozparsovat - provést syntaktickou analýzu. Syntax této metody je v jazyku JavaScript následující: "JSON.parse(názevSouboru)". JSON ve výchozím stavu podporuje datové typy:

- Řetězec (String)
- Číslo (Number)
- Boolean
- null
- Pole (Array)
- Objekt (Object)

1.1.6 Knihovny rozšiřující JavaScript

Existuje velké množství způsobů, jak si programátor může ulehčit svoji práci. K tomuto účelu slouží knihovny, které plní určité služby, jenž by bylo složitější nebo časově náročnější programovat ručně. Knihovna shromažďuje do jednoho či více souborů procedury a funkce, se kterými programátor dále pracuje ve svém zdrojovém kódu. Některé z těchto knihoven se staly díky svému využití velmi oblíbenými. Pro rozšíření nativního JavaScriptu v semestrální práci byly kromě jiných použity tyto knihovny:

Babel

V současné době není podpora všech schopností jazyka JavaScript na všech prohlížečích stejná. Aby nebylo nutné programy napsané v nejnovějších verzích JavaScriptu přepisovat ručně pro všechny verze všech prohlížečů, vznikl nástroj zvaný Babel. Kompilátor automaticky převádí kód tak, aby bylo možné jej spustit z každé starší verze prohlížeče. Tímto způsobem řešíme problém se zpětnou kompatibilitou. [7] Uveďme příklad:

```
//Zápis arrow funkce z ES5
[1, 2, 3].map((n) => n + 1);
//Převod na starší syntax ES
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

TypeScript

Vzhledem k tomu, že JavaScript není typovým jazykem (datové typy u něj není nutné deklarovat a ošetřovat), může se jevit jako jednodušší pro začátečníky. Reálně se však typové programování může velmi hodit pro kontrolu vlastního kódu. TypeScript je rozšíření JavaScriptu, které ovšem pracuje s datovými typy, což činí kód přehlednějším, protože je vždy možné zjistit, jaký datový typ je očekáván např. na vstupu funkce. Je vhodné jej využívat zejména na větších projektech s velkým množstvím řádků kódu, avšak i u menších projektů může možnost ověření správné implementace statických typů kódu zpříjemnit práci programátora. Při instalaci knihoven je nutné doplnit předponu @types, která značí, že program bude psán v TypeScriptu a konkrétní knihovna tomu musí být přizpůsobena. V současné době bylo komunitou kolem TypeScriptu otypováno přes 1000 knihoven jazyka JavaScript. Většina populárních knihoven pro JavaScript tudíž obsahuje i přizpůsobení pro TypeScript. TypeScript kromě své hlavní funkce otypování obsahuje i několik jiných rozšíření. Jedním z nich je metoda enum, která slouží jako výčet hodnot.[8]

ESLint

Jedná se o nástroj sloužící pro upozornění programátora na problémy a chyby, které činí kód nekonzistentním a mohly by se projevit v budoucnu. Jedná se o druh softwaru Lint, jehož obdoby lze využít pro mnoho programovacích jazyků. ESLint po zjištění chyby odkáže uživatele na možné místo či oblast vzniku této chyby. Jeho konfiguraci provádíme v souboru .eslintrc. V něm najdeme soubor pravidel, díky kterým si můžeme určit, zdali kompilátor bude hlásit chybu, varování, nebo jestli bude problém ignorovat. ESLint zamezí možnosti spustit kód, který by sice bylo možné zkompilovat, ale obsahuje programátorské či stylistické chyby. [9]

Prettier

Prettier je knihovna sloužící k formátování kódu a podporuje mnoho jazyků. Po jeho nainstalování se s každým uložením kód sám zformátuje pro optimální čitelnost, jednotnost a přehlednost. Lze jej jednoduše integrovat pomocí většiny editorů kódu. Konfigurace knihovny se provádí v souboru .prettierrc.

1.2 Back-end

1.2.1 Node.js

Node.js je softwarový systém/runtime prostředí umožňující běh JavaScriptu mimo webový prohlížeč. Tento systém využívá V8 JavaScript engine vyvinutý společností

Google a k němu několik knihoven, které skládá v jeden celek. Díky Node.js je nyní umožněno spouštění javascriptového programu mimo prostředí prohlížeče, typicky na straně serveru, tudíž jde v tomto jazyku psát i back-end, což nebylo dříve možné. Při využití systému node.js se v reálném JavaScriptovém programu vytvoří složka node-modules, do které se jednotlivé balíčky instalují. Výhodou node.js je rychlost a vysoká škálovatelnost. [10] [11]

NPM

Node package manager je výchozí správce Javascriptových balíčků Node.js. Používá se pomocí klíčového slova npm v příkazové řádce. Většina knihoven pro JavaScript se instaluje právě přes NPM příkaz. Oblíbenou alternativou k NPM je Yarn.

1.2.2 HTTP Protokol

HTTP Protokol je nejpoužívanějším protokolem na internetu. Zkratka HTTP znamená Hypertext Transfer Protocol a slouží ke komunikaci mezi World Wide Web servery. HTTP funguje na základě rodiny protokolů TCP/IP, umožňujících komunikaci v počítačové síti. Lze o něm smýšlet jako o poslu internetu. Přes HTTP je možné zaslat jakýkoliv druh dat, pokud jsou obě strany komunikace schopny tato data zpracovat. HTTP nevytváří mezi serverem a klientem stálou vazbu, tudíž po splnění své funkce zůstávají obě strany nepropojené, dokud nepřijde další dotaz. [12]Typická HTTP zpráva se skládá ze tří až čtyř částí:

- Start-Line - popisuje dotazy, které mají být vykonány a status o jejich provedení
- HTTP headers - specifikují dotazy a popisují obsah těla zprávy
- prázdný řádek
- body (tělo) - obsahuje veškerá data spojená s dotazem nebo dokument s odpovědí

1.2.3 REST API

API je zkratka pro Application Programming Interface (Rozhraní programovacích aplikací). API slouží k propojení nehomogenních systémů a zprovozňuje komunikaci mezi nimi. REST je zkratka pro Representational State Transfer a je to architektura, pomocí které lze přistupovat k datům přes metody protokolu HTTP. REST je datově orientovaný a k přístupu k datům či stavům aplikace používá zdroje (resources). Data přenesená touto architekturou jsou standardně ve formátu JSON.[13] [14] REST využívá čtyři metody komunikace pod označením CRUD, což znamená:

- Create (POST) - vytvoření dat

- Retrieve (GET) - získání požadovaných dat
- Update (PUT) - změna dat
- Delete (DELETE) - smazání dat

1.2.4 Express

Express je rozhraní pro Node.js, které zjednodušuje komunikaci se serverem. Můžeme pomocí něj definovat cesty (routes) a specifikace, jak se má program chovat, když na server přijde dotaz (request), který je schopen sám rozložit na jednotlivé elementy - rozparsovat jej . Express je také schopen převádět objekty a pole na JSON. Pomocí app.listen(port) potom na konkrétním portu Express "naslouchá"dotazům od klienta a vrací požadovanou odpověď. Celý tento proces je založený na dědičnosti z HTTP prototypů využívaných v systému Node. Zjednodušení spočívá v tom, že odpovědi (responses) ze serveru zasíláme jako celek a program si sám zpracuje, jak se má v dané situaci chovat. [15]

1.2.5 AJAX

je zkratka pro "Asynchronní JavaScript a XML". Jedná se o kombinaci technologií HTML (XHTML), JavaScriptu, XML a XMLHttpRequest. Pointa spočívá v možnosti odeslání a získání dat ze serveru, přičemž není nutné přenahrávat celou stránku. Typickým využitím AJAX jsou např. našeptávače, kdy se mění pouze samotná komponenta a celá stránka se nepřehrává. Název AJAX může být lehce zavádějící, neboť standardně se místo formátu XML využívá formát JSON. Díky tomu, že JSON není výchozím formátem technologie AJAX, je nutné využít metodu "parse". [16]

1.2.6 CORS

V prohlížečích je ve výchozím stavu zavedena zásada stejného původu - stránka vychází ze stejného serveru, z kterého čerpá data. Aby bylo možné komunikovat i s jinými servery, je nutné využít doplňkový protokol CORS (Cross-Origin Resource Sharing). Tento protokol uvádí standardní pravidla pro komunikaci mezi prohlížečem a serverem tak, aby ajaxové aplikace měly přístup i ke zdrojům dat z jiné domény. Při využití CORS bývají v komunikaci zasílány tzv. preflight requests, pomocí kterých se aplikace ptá serveru, zdali mu bude přístup k datům udělen. [17]

1.2.7 Objektově relační mapování

Objektově relační mapování (ORM) je způsob ukládání, získání, aktualizování a mazání dat z objektově orientovaného programu do relační databáze. Díky ORM je

možné automatizovat proces konverze objektů do údajů, které je relační databáze schopna zpracovat. Programátor je tedy odstíněn od nutnosti práce s databází přímo v relační databázi, ale může ji spravovat pomocí vybraného objektově orientovaného programovacího jazyka. Pro Node.js je jednou z nejpoužívanějších softwarů pro ORM knihovna Sequelize.

1.3 Front-end

1.3.1 React

React je open source knihovna pro vytváření uživatelských rozhraní pomocí webových komponent. Klade velký důraz na interaktivitu a dynamičnost. Filozofií Reactu je uchopení UI jako kompozici menších komponent, které se spojují v jeden celek. Velmi často se používá ke sloučení HTML a JavaScriptu do jednoho zdrojového kódu. Označuje se jako "V"v MVC modelu, což je zkratka pro Model-View-Controller, jakožto popis jeho atributů. [18]

- Model - spravuje data a pravidla aplikace
- View - výstup, využívá DOM (Document Object Model) prohlížeče
- Controller - vstup od uživatele, který je konvertován na příkazy pro Model a View[19]

V současné době se pro vytváření front-endu hojně využívají také knihovny Angular.js, Vue.js a Svelte.

1.3.2 Virtual DOM

Virtual DOM je s každou změnou vždy znovu vytvořený přímo Reactem a jeho výhodou je, že pomocí klíčů pouze zjišťuje změny mezi aktuální a předchozí verzí webové aplikace, jejíž stav si zaznamenal, a podle kterých potom upravuje aktuální stav. Když Virtual DOM zjistí změnu, upravuje v reálném DOM pouze tuto změnu. Algoritmy Virtual DOM jsou navrženy tak, aby probíhaly co nejrychleji. [20]

1.3.3 JSX

Je syntaktické rozšíření jazyka JavaScript. JSX se podobá jazyku HTML, proto je pro mnoho uživatelů přívětivý. Vytváříme pomocí něj elementy Reactu. Filozofií JSX je sloučení logiky programu a UI a vytváří jednotky zvané "komponenty". Pro převádění jazyka zpět do JavaScriptu, HTML a CSS je využíván kompilátor Babel. React explicitně nevyžaduje využití JSX, většina uživatelů ho však používá. Aby bylo možné použít JSX s TypeScriptem, používáme typovou verzi JSX s názvem TSX.[21]

1.3.4 Axios

Je knihovna pro vytváření HTTP requestů založená na tzv. "promises", díky čemuž lze používat funkce "async"a "await". Funguje v prohlížeči i systému node.js. Axios podporuje starší verze prohlížečů, automaticky provádí transformaci JSON dat a lze díky němu např. zrušit request či nastavit "response timeout". [22]

1.3.5 Bootstrap a Reactstrap

Jedná se o knihovny s velkým množstvím nástrojů pro stylizaci a grafickou úpravu webových aplikací. Konkrétně Bootstrap je nejpoužívanější knihovnou pro tvorbu front-endového UI vůbec. Bootstrap je vyhlášený mj. díky svému grid systému, který automaticky upravuje rozložení jednotlivých elementů na stránce. Je také přizpůsoben mobilním zařízením. Reactstrap je rozšířením Bootstrapu pro knihovnu React. [23]

1.4 Databáze

Databáze je systém pro ukládání dat, které jsou následně zpracovány. Tyto údaje jsou v databázi systematicky organizované a strukturované. Pomocí skriptů je možné s daty v databázi pracovat a upravovat je. Každá databáze má svoje pravidla pro práci s údaji.

Mezi databázemi mohou být tři typy vztahů, které jsou uvedeny na příkladech:

- 1:1 - V tradičním manželství může být žena provdaná za jednoho muže a muž se může oženit s jednou ženou
- 1:N - Dítě má pouze jednu biologickou matku, matka však může mít vícero dětí
- N:M - Student si může zapsat mnoho předmětů a v předmětu může být zapsáno mnoho studentů[24]

1.4.1 MariaDB

MariaDB je relační databáze odvozená z MySQL. Její výsostí je, že se jedná o open source program. Další výhodou je kompatibilita s většinou vlastností MySQL. MariaDB slouží k uchovávání dat v tabulkách, ve kterých jsou obsaženy záznamy tak, že na jedno pole připadá právě jedna reálie. Tabulky jsou organizovány do řádků a sloupců.[25]

1.4.2 Dostupné databáze s otagovanými obrázky a videi

Jako databáze pro testování byla zvolena pátá verze Open Images Dataset, dostupná z "https://storage.googleapis.com/openimages/web/factsfigures.html". Jedná se o největší databázi s obrázky s anotovanou lokací objektů. Pro videa byla zvolena databáze UFC101, dostupná z "http://www.thumos.info/download.html". Obě knihovny jsou odvozené z projektů sloužících ke strojovému učení rozeznávání jednotlivých objektů na obrázku/videu.

1.5 Možnosti ovládání aplikace pomocí interaktivních ovladačů

Praktická stránka ovládání aplikace pomocí interaktivních ovladačů byla po domluvě s vedoucím práce přesunuta jako jeden z úkolů v navazující bakalářské práci. Po nasazení na server bude aplikace testována na možnost ovládání dotykem. Tato funkcionalita by měla být řešena v rámci prohlížečů na mobilních zařízeních. Vzhledem k tomu, že reálná aplikace by měla být spouštěna právě jen v prohlížečích, nemělo by být nutné se problematikou dotyku dále zabývat. Pro ovládání stránky herním ovladačem lze využít Gamepad API, což je rozšíření pro HTML5 vytvořené přesně pro tento účel. Jednotlivým signálům vysílaným z ovladače lze potom přiřadit specifickou funkci. Gamepad API se skládá ze tří rozhraní. [26]

- Gamepad - reprezentuje ovladač připojený k počítači
- GamepadButton - představuje tlačítko ovladače
- GamepadEvent - je objekt reprezentující situace, které nastanou při používání herního ovladače

Podobně lze v prohlížeči využívat i akcelerometr, a to pomocí podtřídy Sensor API - accelerometer, což je rozhraní, které je schopné číst zrychlení probíhající na všech třech osách. Aby bylo možné s tímto API pracovat, je nutné jej povolit v Permissions API prohlížeče.[27]

2 Postup vytvoření vlastní aplikace

2.1 Back-end a databáze

2.1.1 Příprava pro vlastní implementaci

V této sekci je popsáno vytvoření serveru s REST API, který bude naslouchat frontendu a získávat data z databáze. Do složky Back-end byl pomocí příkazu npm init vytvořen repozitář package.json systému node.js. Poté byly nainstalovány knihovny ESLint a Prettier. Příkazy pro instalaci knihoven jsou velmi jednoduše dohledatelné na stránkách výrobců či na serveru github.com. V případě, že některé knihovny nejsou nainstalovány, ale jejich jména jsou součástí souboru package.json, lze příkazem „npm i“ tyto knihovny hromadně stáhnout. Některé knihovny ve výchozím stavu podporují datové typy, u jiných je potřeba instalaci zohlednit pro TypeScript (většinou pomocí předpony @types).

Ve složce Back-end byla vytvořena složka .vscode se souborem settings.json, ve kterém bylo nastaveno provedení funkcí knihovny ESLint a Prettier na každé uložení aplikace. Tato funkce zjednodušuje práci programátora, standardizuje a zpřehledňuje kód. Knihovny, které slouží vývojářům se označují jako "devDependencies"a ukládají se do vlastní sekce v souboru package.json. Knihovny nutné k běhu programu se označují jako "dependencies". Aby vše fungovalo, bylo nutné ještě vytvořit soubory .prettierrc.js a .eslintrc.js, ve kterém byly nakonfigurovány jednotlivé funkce těchto knihoven. Dále byl nakonfigurován TypeScript pomocí souboru tsconfig.json. Poté byla jako další knihovna pro vývojáře nainstalována knihovna ts-node-dev, sloužící ke spouštění zdrojových kódů v TypeScriptu při každém uložení. Tento postup byl ve složce front-end zduplikován.

Pro samotný back-end byly dále nainstalovány knihovny:

- core-ts
- cors
- express
- node

2.1.2 Spuštění jednoduchého back-end serveru

Následovně byla vytvořena složka "src"se souborem main.ts.

Pro spuštění serveru využijeme knihovnu Express. V souboru, kde chce vývojář danou knihovnu použít, lze využít dva způsoby připojení knihovny:

- klíčové slovo "import"- podporováno prohlížeči a jazykem TypeScript
- klíčové slovo "require"- podporováno systémem node.js

Pro spuštění back-endového serveru na námi zvoleném portu (adresa portu byla zvolena 3001) je tedy nutné využít v souboru main.ts následující příkazy:

```
import * as express from 'express'
const app = express()
const port = 3001
```

Vzhledem k tomu, že front-end poběží na jiném portu, než back-end, je nutné umožnit těmto různým doménám komunikaci. Jako další byla tedy nainstalována a naimportována knihovna CORS. Pomocí jednoduchého kódu:

```
app.use(cors())
```

byla umožněna serveru komunikace s jakýmkoliv jiným serverem. Pomocí systému whiteliste/blacklist je také možné přímo nastavit, kdo může data ze serveru přijímat a kdo ne. Nyní je back-end aplikace schopná komunikovat s front-endem.

2.1.3 Databázové modely

Je nutné, aby tabulky v databázi splňovaly zvolené schéma a také je potřeba je vzájemně propojit. K tomuto účelu byly vytvořeny databázové modely AssetsModel, TagsModel a TagsAssetsModel, ve kterých je definována struktura dané tabulky a také jejich asociace. Celý model se bude exportovat jakožto šablona pro vytváření reálné databáze. V této definici jsou obsaženy objekty s předdefinovaným datovým typem. Pro model "AssetsModel"vypadá soubor následovně:

```
//import celého obsahu knihovny Seqzelize
import * as Sequelize from 'sequelize'

//metoda jazyka TypeScript.
//výčet hodnot možných typů assetů
export enum AssetType {
  Video = 'Video',
  Image = 'Image',
}

 //parametr "sequelize" je objektem třídy
 //Sequelize z knihovny Sequelize
export default (sequelize: Sequelize.Sequelize, DataTypes: any) =>
//definuje název tabulky, počet a název sloupců, včetně datového typu
  const AssetsModel = sequelize.define('assets', {
    id: {
      type: DataTypes.STRING,
```

```
      primaryKey: true,
   },
    name: {
      type: DataTypes.STRING,
    },
    type: {
      type: DataTypes.STRING,
    },
    creation_year: {
      type: DataTypes.INTEGER,
    },
    src: {
      type: DataTypes.STRING,
    },
  })

 //metoda pro propojení tabulek
 AssetsModel.associate = models => {

 //1:N vazba na vazební tabulku
    models.Assets.hasMany(models.TagsAssets, {
      foreignKey: 'asset_id',
      sourceKey: 'id',
      constraints: false,
    })
 }
  return AssetsModel
}
```

2.1.4 Zdrojová data pro databázi

Data z knihoven pro obrázky a videa je potřeba převést do databáze MariaDB. Databáze Open Images Dataset disponuje i vymaskovanými objekty a označením objektů do obdélníků pro strojové učení. Pro účely semestrální práce je však nutné získat data jen pro tři tabulky v databázi. V sekci Download na stránkách

https://storage.googleapis.com/openimages/web budou tedy staženy ze sekce Validation, která obsahuje 41,620 obrázků, následující soubory:

- Image IDs - obsahuje kromě jiných informací ID obázků a URL adresy k jednotlivým obrázkům, tudíž bude sloužit jako tabulka "Assets"

- Image labels - vazební tabulka, obsahuje v M:N poměru položky z tabulky "Tags"a "Assets"
- Class names - obsahuje názvy jednotlivých tagů a jejich ID, tudíž bude sloužit jako tabulka "Tags"
- Set of boxable classes - JSON soubor obsahující hierarchii tagů

Pro videa z http://www.thumos.info ze sekce Download budou použity dva soubory. ID videí bude nutné uměle vytvořit, vazba je popsána v názvech souborů:

- UCF101 videos (individual files): [Link] - obsahuje URL adresy k jednotlivým videím, které budou přehrány pomocí prohlížeče, tudíž bude sloužit jako tabulka "Assets"
- Class-level attributes - obsahuje data pro vazební tabulku i pro tabulku "Tags"

2.1.5 Úprava zdrojových dat

Aby bylo možné data sjednotit do tří tabulek, je nutné prvně zpracovat zdrojové soubory, které jsou nekonzistentní. Z tohoto důvodu byl vytvořen soubor "loadAndParseData.ts", obsahující metody, kde které konvertují jakoukoliv podobu vstupních dat do formátu JSON, se kterým lze dále pracovat. V tomto souboru byla také vytvořena funkce getRandomYear, která vytváří falešný rok výroby v rozsahu 1900 až 2020. Tato funkce slouží k umělému vytvoření dat, ve kterých je logické využít výběr z rozsahu.

Obrázky

Pro zpracování souborů s příponou .csv, což jsou všechny soubory s údaji o obrázcích (kromě hierarchie, která je JSON), lze využít knihovnu "csvtojson/v2", která, jak již název napovídá, převádí soubory s příponou .csv na formát JSON. V této fázi tvorby je třeba si stále uvědomovat, jak daná data vypadají a jak je možné je ve výsledku propojit, aby jejich příprava dávala smysl pro připravené databázové modely. Soubory sloužící jako "Tags"a "Assets"disponují svými unikátními ID, které lze potom využít ve vazební tabulce. V každé metodě v poli headers, byly vypněny názvy kategorií všech reálných položek v daných .csv souborech. Výstupem těchto metod jsou však pouze data relevantní pro pozdější využití.

Videa

Pro videa je operace složitější, neboť jejich data nejsou rozčleněna do na první pohled jasně propojitelných sekcí. Při otevření souboru "Class-level attributes"se zobrazí tabulka, jejichž řádky jsou popsány jednotlivými názvy tags a jejich sloupce jednotlivými názvy assets. Celý vnitřek tabulky tvoří potom nuly a jedničky, které

vyjadřují, zda-li se daný tag v assetu nachází. Tudíž tabulka dohromady obsahuje názvy tags, assets i vazební tabulku, chybí v ní však url adresy jednotlivých videí.

Tyto adresy je možné získat z http://www.thumos.info ze sekce Download, konkrétně UCF101 videos (individual files): [Link], kde se nachází seznam odkazů pro stažení videí. Po rychlém prozkoumání chování stránky došlo k zjištění, že při doplnění názvu videa za lomítko URL adresy se dané video stáhne. Tudíž stačí celou stránku zkopírovat jako textový soubor, protože v něm jsou obsaženy veškeré nutné údaje pro účely semestrální práce. S textovými soubory je možné v JavaScriptu pracovat pomocí knihovny fs, která je schopna je přečíst. Poté už byly jen oba soubory zpracovány do vhodných formátů. Textový soubor byl zbaven přebitečných znaků metodou split, tabulka byla podle binárního kritéria profiltrována tak, aby k danému assetu připojila výčet tagů, které reálně má.

2.1.6 Vytvoření Databáze

Po stažení a instalaci databáze MariaDB byla vytvořena databáze pomocí grafického klienta HeidiSQL. Pro tento účel je možné zvolit také příkazovou řádku. Při tvorbě databáze je nutné vyplnit následující údaje:

- Hostitel/IP - 127.0.0.1 (localhost)
- Uživatel - root
- Heslo
- Port - 3306 (výchozí)
- Název databáze - semestralniPrace.db

Pro možnost přístupu do databáze skrz aplikaci pro back-end byl vytvořen soubor "env-config.ts", který uchovává přihlašovací údaje pro přístup k databázi. Tyto přístupové údaje jsou zpracovány v souboru core.ts. V tomto souboru se nachází konfigurace objektově relačního mapování prostřednictvím knihovny Sequelize. Při zavolání metody sequelize.authenticate() je možné vyzkoušet připojení k databázi. Objekt models uchovává obsah databázových modelů a konfiguraci objektu sequelize. Pomocí následující funkce jsou realizovány vazby mezi jednotlivými modely. Objekt models je ze souboru exportován.

2.1.7 Skript pro zpracování souborů do databáze

Skript dbLoadData je možné spustit příkazem "npm run db:load-data". Data, která jsou nyní ve vhodném formátu pro zpracování, rozdělí do tabulek a pomocí objektově relačního mapování jimi naplní databázi. V databázi budou tedy tři tabulky, kde v jedné jsou informace o obrázcích a videích, v druhé jsou jejch tagy a třetí obsahuje jejich vazbu. Všechny metody z předchozího souboru "loadAndParseData.ts"jsou do "dbLoadData"naimportovány přes klíčové slovo "import".

Zatímco obrázky i jejich tagy disponují svým vlastním unikátním ID, u videí tomu tak není, tudíž je potřeba jim ID uměle vytvořit. K tomuto účelu lze použít např. knihovnu shortid. Tyto ID byly v jednotlivých metodách přidány videím, i jejich tagům. Všechny metody v první části skriptu slouží k úpravě dat tak, aby byly vhodné pro nasazení do databáze, z čehož nejsložitější proces nastává u vazební taulky pro videa, kde byla data sloučena pomocí matice s objekty, která byla funkcí .flat() ochuzena o jednu dimenzi tak, aby z ní zůstaly pouze vazby uměle vytvořených ID videí a ID jejich tagů.

Celý tento skript má ve výsledku tři fáze. V první fázi dojde k restartování databáze pomocí metody knihovny Sequelize - sequelize.sync(force: true), která resetuje databázi a umožní programu přepisovat data. V druhé fázi jsou volány funkce skriptu loadAndParseData. Ve třetí fázi jsou použitelná data finalizována a zaslána do databáze. Nahrávání do databáze je možné optimalizovat metodou knihovny Sequelize bulkCreate, která data posílá ve velkém množství najednou, místo toho, aby byla data nahrávána po jedné položce. Vzledem k tomu, že program nebyl schopen zaslat celou databázi najednou, bylo nutné použít metodu slice, která umožnila zasílání po menších částech (po statisících položek).

2.1.8 Funkce main()

Nyní, když jsou data nahrána v databázi, je nutné definovat, jak se s nimi bude nakládat a co přesně budou vracet při přijmutí requestu. Aby aplikace naslouchala možným requestům, je nutné využít metodu app.listen(). Vzhledem k tomu, že aplikace data pouze zobrazuje, je jediným nutným typem requestu get. Tyto requesty jsou namířeny na určitou URL adresu zvanou "endpoint". Práci s těmito endpointy ulehčuje knihovna Express, díky které je jednoduše možné nadefinovat, jak se bude back-end při jakém requestu na jaký endpoint chovat. Vývoj této části back-endu reálně probíhal paralelně s vývojem front-endu, tudíž se zatím důvod volby chování back-endu na jednotlivých end-pointech mohou jevit jako neopodstatněné. Jejích propojení s front-endem je popsáno v kapitole "Struktura hlavního souboru". V aplikace jsou využívány endpointy:

- /tags - vrací seznam dat tabulky tags
- /assets/:assetId/tags - podle ID assetu vrátí seznam tagů
- /assets/fulltext - podle nastavených parametrů v requestu ovlivňuje zobrazení assetů
- /assets/tags/:tagId - podle ID tagu vrací assety

2.2 Front-end

2.2.1 Vytvoření nové aplikace v knihovně React

V nové složce s názvem front-end byla vytvořena nová aplikace knihovny 'React' pomocí příkazu npm create-react-app Front-end –typescript. Výchozí port takto vytvořené aplikace je 3000. Ve složce se objevily soubory:

- package.json - obsahuje metadata, skripty a názvy využitých knihoven
- package-lock.json - "zámek"pro verze knihoven využitých v aplikaci. Nové verze jednotlivých knihoven mohou ohrozit chod aplikace a tudíž je nutné jejich verze "uzamčít".
- gitignore - seznam souborů, které nemají být zaverzovány systémem git
- README.md - obsahuje informace ohledně aplikace knihovny React

a složky:

- public - soubory pro vytvoření výchozí aplikace knihovny React
- src - soubory pro samotný zdrojový kód
- node-modules - obsah používaných knihoven

Pomocí příkazu npm start se aplikace spustí. Každá webová aplikace či stránka, využívá HTML, které spouští soubor index.html, které je v projektu zastoupeno souborem index.tsx. V aplikaci React je v něm standartně jen spouštěna hlavní aplikace (App.tsx) metodou ReactDOM.render V souboru se nachází také metoda serviceWorker.unregister(), která umožňuje rychlejší práci v režimu offline. V hlavním souboru App.jsx je voláno vykonání souboru AssetsList, která je v případě tohoto projektu reálným mozkem celého front-edu.

2.2.2 Struktura hlavního souboru

Soubor AssetsList.tsx je jádrem celého programu, neboť je v něm reálně obsaženo vše, co se zobrazuje na straně klienta. V souboru, který plní takovou funkci by ideálně mělo být co nejméně kódu, prototože správná aplikace knihovny React je v podstatě kompozice komponent, a tyto komponenty by měly být rozdělené do jednotlivých menších souborů, které jsou v tomto jádru pouze zavolány. Úplně nahoře v kódu se tedy nachází seznam importů, což je typické.

Následuje definice globálních konstant, což je v případě této práce pouze PAGE_ OFFSET_SIZE , což určuje změnu strany vyhledávání assetů. Vzhledem k tomu, že je aplikace psána v jazyku TypeScript, je nutné pro jednotlivé objekty nadefinovat, o jaké datové typy se pro každou položku každého objektu jedná. Následuje deklarace stavových komponent třídy AssetsList, která udává, jaký je stav aplikace při jejím spuštění. Stav je kromě komponent tříd možné řešit v Reactu i pomocí tzv. Hooků, což jsou funkce, které se volají do jádra reactu a k dané informaci se

"zaháčkují". Funkce componentDidMount je funkcí z React Lifecycle, která se zavolá při připojení kompenent do DOM. Dále se zde nachází seznam funkcí, které jsou v zásadě dvou typů. Funkce měnící stav aplikace lokálně a funkce měnící stav aplikace využívající requesty. Pro práci s requesty je využívána knihovna axios, která umožňuje jednoduchými příkazy posílat na jednotlivé endpointy requesty.

2.2.3 Optimalizace

Při vytváření první verze aplikace nebyl kladen důraz na rychlost aplikace, tudíž nebylo překvapující, že po nasazení reálné databáze na server byla aplikace prakticky nefunkční skrz svoji pomalou odezvou. Aplikaci knihovny React lze optimalizovat několika způsoby. Pokud je aplikace knihovny React vytvořena za pomocí create-react-app, lze využít příkaz "npm run build". Tento příkaz přetvoří soubory tak, aby byly vhodné pro nasazení na produkci, čímž je oklešťuje o funkcionalitu, kterou není nutné v produkci používat.

Nejvyšší míry optimalizace front-endu dosáhneme při minimalizaci nutnosti přerenderování obsahu aplikace při manipulaci s jednotlivými komponenty. Potlačení nutnosti přerenderování provádí právě VirtualDOM, který porovnává stav reálného DOM s uloženým stavem, a až při detekci změny DOM přerenderuje.

Posílit tento efekt lze metodou React.memo(), která zapouzdří svůj obsah a zapamtuje si výsledky zapouzdřených funkcí při jejich volání. Pokud nedošlo v zapouzdřené funkci k žádné změně, posílá se jako její výstup pouze reference na položku s výsledkem v paměti, což je z principu méně náročný proces, než znovuprovedení dané zapouzdřené funkce. Tímto způsobem urchluje běh aplikace i metoda shouldComponentUpdate() třídy React.PureComponent.

Další možností jak zrychlit chod aplikace je optimalizace back-endu a databáze. Pro databáze lze využít např. indexování nebo klíčové slovo EXPLAIN.

2.2.4 Komponenty

Pro přehlednost a optimalizaci je důležité rozdělit hlavní soubor do vícero souborů tak, aby s nimi bylo možné odděleně manipulovat. Front-end tohoto projektu se skládá z několika komponent, které budou v následujícím tetu popsány. Na začátku každého komponentového soubory se nachází seznam importů a definice datových typů pro jazyk TypeScript.

Tlačítka pro obrázky

Pro splnění úkolu vyhledávání pomocí binárního kritéria byl zvolen seznam tlačítek s názvy tagů, které na kliknutí zobrazí v seznamu assetů ty assety, které daný tag ob-

sahují. Pro smysluplnější zobrazení je využito souboru bbox_label_600_hierarchy, který byl pro přehlednost přejmenován na TagsHierarchyTree. Tento soubor obsahuje stromovou strukturu tagů, které jsou rozřazeny do subkategorií. Jednotlivé položky této struktury obsahují tagID obrázků přímo ze zdrojové stránky databáze obrázků. Vzhledem k tomu, že tyto unikítní tagID byly zachovány, lze poměrně jednoduše jména tagů podle jejich ID na stromovou strukturu nasadit. Tato komponenta rozšiřuje React.PureComponent, díky čemuž dochází k optimalizaci. Pro namapování všech zanoření stromové struktury byl použit rekurzivní algoritmus. Při shodě tagID stromové struktury a tagID z databáze je tady tento tag vyobrazen jako tlačítko v rolovacím odrážkovém seznamu.

Tlačítka pro videa

Pro vyobrazení seznamu videí byla využita funkce getTagsByType, pomocí které lze tískat seznam tagů pro videa filtrací stromové struktury tagů z předchozího bodu. Tyto zbylé tagy jsou potom v render funkci namapovány a zobrazeny jako samostatný rolovací seznam.

Slidery

Slider, neboli jezdec, je grafický element, pomocí kterého lze nastavit určitou hodnotu pohybem indikátoru v určitém rozmezí. Pro elegantnější vzhled komponenety byla použita knihovna react-input-slider. Slidery v tomto programu ovlivňují rozsah filtru, který určuje, které položky se zobrazí, a to za pomocí creationYear - roku vytvoření. Komponenta vytvoří dva slidery, které reprezentují creationYearFrom (rok vytvoření od) a creationYearTo (rok vytvoření do). Dále je možné nastavit nejmenší možný krok, který pohyb slideru způsobí, což je v tomto případě jeden rok. V programu AssetsList jsou tyto slidery propojený s vyhledávačem, tudíž reálně omezují výdledky vyhledávání při kliknutí na tlačítko submit. Pro optimalizaci byla komponenta zaobalena metodou React.memo().Tato komponenta je řešením vyhledávání z rozsahu.

Input

Input, neboli vstup, je lišta, do které může uživatel psát text. Při stisknutí tlačítka submit se tento vstup zpracuje do requestu, který je poté odeslán na server. Server vrátí výsledky profiltrované podle zadaného parametru. Odpovídající výsledky jsou všechny assety obsahující vyhledávaný výraz v seznamu tagů. Tudíž při vyhledávání "x"se zobrazí např. assety s tagem "Saxophone", "Box"nebo "Taxi". Ve výchozím stavu obsahuje lišta prázdný string. Aby se při každé změně obsahu lišty stránka nepřerenderovala, je metoda ukládající stav lišty z funkce render() vyjmuta. Aby

bylo zabráněno zaslání velkého množství requestů najednou, byla vytvořena funkce loading, která tlačítko submit vypíná, dokud se výsledky vyhledávání nezobrazí.

Zobrazení seznamu tagů

Vzhledem k tomu, že u videí nejsou obsaženy miniatury a grafická prezentace je předmětem až navazující práce, byla zvolena možnost dvojího zobrazení výsledků pomocí jejich typů. Tyto typy jsou "video"a "image". Pokud se jedná o video, je zobrazen název a text "click to download"s odkazem ke stažení daného videa. Obrázky se ve vyhledávání zobrazují zmenšeně. Oba druhy zobrazení disponují ještě tlačítkem "show details", při jehož stlačení se zobrazí výčet tagů, které daný obrázek či vdeo obsahuje.

2.3 Grafika

Stylizace stránky byla provedena zejména využitím knihovny Reactstrap, odnože Bootstrap. Nativní značky jazyka HTML byly z velké části zaměněny za elegantnější a moderněji vypadající elementy knihovny Reactstrap. Celý zobrazený obsah byl zabalen do grid systému Bootstrapu, který funguje na základě jednoduchého rozložení stránky na řady a sloupce. Dále byla vyměněna všechna HTML tlačítka <button> za <Button> Reactstrapu, které nabízí nové možnosti jejich zobrazení. Načítání stránky nyní znázorňuje točící se kolečko - Spinner.

Závěr

V rámci semestrální práce byl vypracován základ programu, na který bude v bakalářské práci navázáno. Tento program disponuje funkčním uživatelským rozhraním s základní grafickou reprezentací výsledků a bez možnosti ovládání interaktivními ovladači, což bude řešeno v navazující práci. Front-end programu komunikuje s back-endem přes requesty, které jsou zprostředkovány knihovnami Axios a Express. Back-end komunikuje s databází pomocí objektově relačního mapování. Program byl otestován na databázi obsahující dohromady 54940 obrázků a videí, které jsou přes 715 klíčových slov provázány do 356707 vazeb. Data jsou zobrazena formou seznamu, ve kterém je možné filtrovat podle binárního kritéria, výběrem z množiny a z rozsahu. Kvůli většímu množství dat byl program základně optimalizován, míra optimalizace bude zvýšena v navzující práci.

Literatura

[1] PÍSEK, Slavoj. *HTML: začínáme programovat. 4., aktualiz. vyd.*. Praha: Grada, 2014. Průvodce (Grada). ISBN 978-80-247-5059-0.

[2] DUCKETT, Jon. *Beginning HTML, XHTML, CSS, and JavaScript*, Chichester: John Wiley distributor, c2010. ISBN 978-0-470-54070-1.

[3] JANOVSKÝ, Dušan. *CSS styly - úvod*. [online]. [cit. 2019-11-22] Dostupné z URL:
<https://www.jakpsatweb.cz/css/css-uvod.html .

[4] HAUSER, Marianne, Tobias HAUSER a Christian WENZ. *HTML a CSS: velká kniha řešení*. Brno: Computer Press, 2006. ISBN 80-251-1117-2.

[5] DEAN, John. *Web programming with HTML5, CSS, and JavaScript*. Burlington, Massachusetts: Jones & Bartlett Learning, 2019. ISBN 9781284091793.

[6] ŠKULTÉTY, Rastislav. *JavaScript: programujeme internetové aplikace*. Praha: Computer Press, 2001. Pro každého uživatele. ISBN 8072264575.

[7] MIKŠŮ, Vojtěch. *JavaScript? Babel.*. [online]. [cit. 2019-11-22] 2016. Dostupné z URL:
<https://www.dzejes.cz/babel.html>.

[8] DAJBYCH, Václav. *K čemu je dobrý TypeScript*. [online]. [cit. 2019-11-22] 2013. Dostupné z URL:
<https://www.zdrojak.cz/clanky/k-cemu-je-dobry-typescript/>.

[9] MACHAČ, Marek. *Automatická detekce potíží v JavaScriptu s ESLint*. [online]. [cit. 2019-11-22] 2015. Dostupné z URL:
<shorturl.at/fiqv8>.

[10] NGUYEN, Don. *Jump Start Node.js* Collingwood VIC: SitePoint, 2012. ISBN: 978-0-9873321-1-0

[11] TEIXEIRA, Pedro. *Professional node.js: building JavaScript-based scalable software*, Indianapolis, IN: John Wiley & Sons, 2013. ISBN 978-1-118-18546-9.

[12] AVIANI, Goran. *HTTP and everything you need to know about it*. [online] [cit. 2019-11-22] 2018. Dostupné z URL:
<shorturl.at/ozSUV>.

[13] DOGLIO, Fernando. *Pro rest api development with Node.js*. New York, NY: Apress, 2015. ISBN 978-1-4842-0918-9.

[14] MALÝ, Martin. *REST: architektura pro webové API*. [online]. [cit. 2019-11-22] 2009. Dostupné z URL: <https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>.

[15] SUBRAMANIAN V. *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node*, Berkeley, California: Apress, 2017. Books for professionals by professionals. ISBN 978-1-4842-2652-0.

[16] SALVET, Pavel. *Seriál o komunikaci mezi stránkou a serverem: Díl 1. AJAX v kostce*. [online] [cit. 2019-11-22] 2015. Dostupné z URL: <https://www.interval.cz/clanky/ajax-v-kostce/>.

[17] SALVET, Pavel *Seriál o komunikaci mezi stránkou a serverem: 2.AJAX: CORS,polling*. [online] [cit. 2019-11-22] 2015. Dostupné z URL: <https://www.interval.cz/clanky/ajax-cors-polling/>.

[18] FEDOSEJEV, Artemij. *React.js Essentials*. Birmingham, UK: Packt Publishing, 2015. ISBN 978-1-78355-162-0

[19] HAMEDANI, Mosh. *JavaScript for React Developers | Mosh*. In: Youtube [online] [cit. 2019-11-22] 2018, Dostupné z URL: <https://www.youtube.com/watch?v=NCwa_xi0Uuc .

[20] HEIS, Kevin. *Let's Build a Virtual DOM from Scratch*. In: Youtube [online] [cit. 2019-11-22] 2017, Dostupné z URL: <https://www.youtube.com/watch?v=l2Tu0NqH0qU .

[21] MIKŠŮ, Vojtěch. *React - JSX*. [online]. [cit. 2019-11-22] 2016. Dostupné z URL: <https://www.dzejes.cz/react-jsx.html>.

[22] BERNARDES, Marlon. *How to Use Axios as Your HTTP Client*. [online] [cit. 2019-11-22] 2015. Dostupné z URL: <http://codeheaven.io/how-to-use-axios-as-your-http-client/ .

[23] ČÁPKA, David. *Lekce 1 - Úvod do CSS frameworku Bootstrap*. [online] [cit. 2019-12-18] 2018. Dostupné z URL: <https://www.itnetwork.cz/html-css/bootstrap/kurz/uvod-do-css-frameworku-bootstrap>.

[24] BRUMM, Ben. *How to Handle a Many-to-Many Relationship in Database Design*. [online] [cit. 2019-11-22] 2017. Dostupné z URL: <shorturl.at/ftHMX .

[25] BARTHOLOMEW, Daniel. *MariaDB Cookbook* Birmingham, UK: Packt Publishing, 2014. ISBN 978-1-78328-439-9

[26] CAMDEN, Raymond. *Using the HTML5 Gamepad API to Add Controller Support to Browser Games*. [online] [cit. 2019-12-07] 2019. Dostupné z URL: <`shorturl.at/cejl6`>.

[27] BAR, Adam. *Device Motion* [online] [cit. 2019-12-07] Dostupné z URL: <`https://whatwebcando.today/device-motion.html`>.

Seznam příloh

A Médium **41**

A Médium

Médium obsahuje kompletní počítačový kód včetně návodu k jeho spuštění.