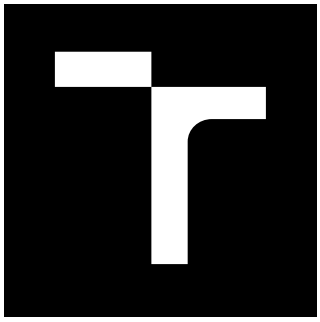
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Dominik Kuře



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

INTERAKTIVNÍ VYHLEDÁVÁNÍ V ON-LINE ARCHIVU OBRAZOVÝCH A AUDIOVIZUÁLNÍCH DĚL
INTERACTIVE SEARCHING IN ONLINE ARCHIVE OF VISUAL AND AUDIOVISUAL ARTWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Dominik Kuře**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jiří Schimmel, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Dominik Kuře ***ID:*** 203736
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl

POKYNY PRO VYPRACOVÁNÍ:

Pomocí systému node.js, knihovny React a databáze MariaDB vytvořte webovou aplikaci, která umožní interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl. Předpokládá se, že všechna díla v archivu jsou opatřena klíčovými slovy z různých skupin, a cílem práce je vytvořit uživatelské rozhraní, které bude graficky reprezentovat vyhledávání pomocí těchto klíčových slov a to jako binární kritérium, výběr z množiny a výběr z rozsahu. Při realizaci je potřeba dbát na možnost ovládání dotykem a interaktivní ovládání, např. herními ovladači, akcelerometrem a dalšími.

DOPORUČENÁ LITERATURA:

[1] CONNOLLY, Randy a Ricardo HOAR. Fundamentals of Web Development. 2. Pearson, 2017. ISBN 978-0134481265.

[2] Nette Docomentation [online]. Nette Foundation, 2019 [cit. 2019-09-16]. Dostupné z: https://doc.nette.org/cs/3.0/

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce je zaměřena na vývoj webové aplikace, jenž slouží jako archiv pro audiovizuální díla, která jsou opatřena klíčovými slovy. Aplikace pracuje s již vytvořenou databází a serverem, na kterém se tato díla nachází. Databáze byla autorovi předána vedoucím práce. V archivu lze vyhledávat konkrétní díla pomocí vyhledávací lišty a několika druhů filtrů, které jsou vázány na informace o každém díle. Výsledky vyhledávání i databáze jako celek jsou ve vytvořené aplikaci reprezentovány grafy, které se podle stavu aplikace dynamicky mění. Jednotlivá díla lze na stránce přehrávat a na základě shod v klíčových slovech je pro každé dílo vytvořen seznam podobných videí. Aplikace byla realizována pomocí systému Node.js, knihovny React, databáze MariaDB a dalších technologií. Jazykem programu je převážně JavaScript, rozšířený o nemalé množství knihoven, díky kterým je možné ve všech oblastech vývoje tento jazyk používat jako primární. Text bakalářské práce je rozdělen na dvě části. V teoretické části jsou veškeré potřebné technologie a knihovny popsány. Důraz byl kladen zejména na ty části daných knihoven, které jsou v aplikaci opravdu používané. Kromě technologií nutných pro realizaci aplikace jsou také zmíněny knihovny, systémy a rozšíření, které práci programátora ulehčí, např. automatickým formátováním a zálohováním kódu, ukládáním kódu do verzí, přidáním statických datových typů do JavaScriptu pomocí TypeScriptu atd. Přečtení teoretické práce by mělo umožnit čtenáři nahlédnout do základů programování aplikací pro webové prohlížeče. V praktické části je potom rozebrán reálný vývoj takové aplikace od úplného začátku. Aplikace zprostředkovává komunikaci mezi čtyřmi servery - front-end, back-end, databázový server a server obsahující audiovizuální díla. Bakalářská práce se také zabývá možností ovládání prohlížeče pomocí interaktivních ovladačů.

KLÍČOVÁ SLOVA

Aplikace, Axios, Bootstrap, CORS, CSS, Express, HTML, HTTP, JavaScript, MariaDB, Node.js, React, Reactstrap, Recharts, Sequelize, TypeScript, Web

ABSTRACT

This bachelor thesis focuses on development of a web application, which serves as an archive for videos. Each video has a certain amount of keywords. The application uses an already created database and a preinstalled server on which the videos are uploaded. The database was given to the author by his supervisor. Searching through the archive can be done by inputting an expression into a search bar or through a variation of filters that are based on information about each video. The results and the database as a whole are graphically represented by charts, which change their form based on given data. Videos can be played in a video player and a list of similar videos is generated. The list is based on keywords which the videos have in common. The main technologies used in the application are Node.js, React and MariaDB. A good amount of libraries are used for this application, allowing JavaScript to be the primary programming language in all phases of development. The text of this bachelor thesis can be dividen in two parts - theoretical and practical. The first part describes all the technologies and libraries used in the application. An in depth approach was taken especially on those parts of each library which are actually being used in the practical. Apart from the necessary technologies, the reader will be also introduced to libraries and systems which help a programmer with his work such as automatic formatting of code and it's backup, saving different versions of the code or adding static datatypes into JavaScript through TypeScript. The theoretical part should give the reader a summary of how browser applications work and communicate with each other. In the practical part an entire application will be built from scratch. The application will connect four different servers - front-end, back-end, database server and a server storing the videos - and allow them to communicate accordingly. The bachelor thesis also contains information about using different controllers for manipulating the browser.

KEYWORDS

App, Axios, Bootstrap, CORS, CSS, Express, HTML, HTTP, JavaScript, MariaDB, Node.js, React, Reactstrap, Recharts, Sequelize, TypeScript, Web

KUŘE, Dominik. *Interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl*. Brno, 2020, 66 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jiří Schimmel, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce panu doc. Ing. Jiřímu Schimmelovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

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 zabývat jakožto kariérní možností. Tato práce se povrchově věnuje všem základním oblastem vývoje webové aplikace pro účel zpracování audiovizuálních děl do formy interaktivního archivu, ve kterém je možné dle různých kritérií filtrovat výsledky vyhledávání. Aby bylo možné díla systematicky zobrazovat ve webové aplikaci, je nutné je v digitální podobě ukládat spolu s jejich metadaty, která jsou následně využita pro vyhledávání, filtrování výsledků hledání a vizualizaci dat. Celý tento proces je možné řešit mnoha způsoby z nichž jeden je popsán právě v této bakalářské práci. Pro plnohodnotné pochopení textu praktické části bakalářské práce silně doporučuji číst i zdrojový kód.

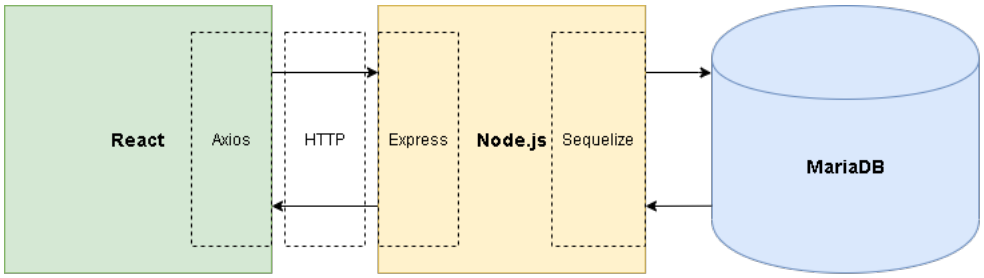
1 Teoretický úvod do webových aplikací

1.1 Full-stack aplikace

Pro úspěšné vytvoření webové aplikace je nutné pochopit celý proces jejího vývoje. Kompletní webová aplikace, která zprostředkovává komunikaci mezi uživatelem a serverem, 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, struktura uživatelského rozhraní webové aplikace a komunikace se serverem. Zatímco front-end využívá standardně 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 realizaci back-end části aplikace v jazyce JavaScript. Alternativou k Node.js mohou být např. jazyky Python, Java, PHP, C# nebo Ruby. V databázi jsou uchovávána data, která jsou využívána webovou aplikací. Oblast DevOps se poté zabývá testováním, nasazováním aplikace na server a jejím provozem. Následující kapitoly stručně popisují jazyky, technologie, knihovny a programovací paradigmata využívané v bakalářské práci.



Obr. 1.1: Technologie, na kterých je praktická část práce založena

Editor kódu

Jako software pro psaní a 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. Visual Studio Code při detekci typu souboru a syntaxe automaticky zjišťuje, o který jazyk se jedná a nabízí rozšíření, která je vhodné nainstalovat. Díky integraci s různými vývojářskými nástroji Visual Studio Code jakožto editor nabízí srovnatelné možnosti s programy IDE (integrated development environment). Dalšími populárními IDE a editory jsou např. Visual Studio, Sublime Text, Atom nebo Eclipse.

Systém Git

Jedná se o mezi programátory hojně využívaný systém ukládání a verzování dat. Nejčastěji se s ním (spíše s jeho rozšířením) lze setkat v podobě webových aplikací poskytujících uživatelům možnost ukládat a spravovat svůj kód. Mezi nejznámnější z nich patří GitHub, GitLab a Bitbucket. Git umožňuje uložit vlastní kód do tzv. repozitáře, čímž je vytvořen časový snímek - snapshot - stavu programu. Kód je zároveň zálohován a programátor má vždy možnost porovnat nový kód s jeho staršími verzemi. Při práci s Gitem je možné vytvořit vícero větví - branches - ve kterých lze na kódu samostatně pracovat bez ovlivňování ostatních větví, hlavní větev se označuje jako „master“. Pomocí systému Git lze ukládat kromě kódu v podstatě jakýkoliv textový soubor.

1.1.1 Jazyk HTML

Hypertext Markup Language (HMTL) je značkovací jazyk, pomocí kterého se vytváří struktura dokumentů webových stránek. Označení „hypertext“ značí, že jednotlivé stránky jsou propojeny pomocí odkazů. Jedná se o standardní jazyk podporovaný všemi běžnými prohlížeči a vývojář je díky němu schopen prohlížeči předat informaci o tom, jaký obsah má zobrazovat a provést základní formátování textu.

Pomocí některých prvků, které HTML poskytuje, je vývojáři umožněno pracovat s velikostí textu a rozhodnout, zda-li bude vypsán tučně či kurzívou, importovat do dokumentu obrázky, animace ve formátu GIF, video a audio, 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 mnohé další dnes běžně používané součásti moderního webu. HTML prvky jsou v kódu reprezentovány pomocí tagů (značek), jazyk HTML využívá syntax párovou <tag></tag>, kde „/“ označuje konec působiště daného tagu, a nepárovou <tag />. Tyto značky lze podle jejich významu rozdělit na tři skupiny - strukturální, popisné a stylistické. Jednotlivé prvky přijímají atributy v rámci otevírací značky (seznam atributů každého prvku je možno dohledat v dokumentaci). [1]

Zdrojový kód HTML dokumentu je zaobalený ve značkách <html> - kořenovém prvku. HTML dokument standardně obsahuje alespoň dva prvky. Těmi jsou <head> a <body>. V prvku <head> - hlavičce souboru - jsou obsaženy informace o dokumentu, které se přímo v obsahu stránky v prohlížeči nezobrazují.

V elementu <body> - těle dokumentu - se téměř vždy nachází většina HTML kódu. Validní HTML soubor by měl na začátku obsahovat deklaraci typu souboru <!DOCTYPE html>. V dokumentu je nutné dodržovat tzv. nesting, kdy značky vnořeného prvku musí mít svůj začátek i konec mezi značkami prvku vnějšího. HTML disponuje mnoha značkami, které plní různé funkce. Mezi ty základní 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 světě IT dlouhou historii, jeho oficiální vydání se datuje k roku 1990. Mezi další významné průlomy tohoto období lze zařadit protokol HTTP a World-WideWeb. 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 XHTML svého předchůdce z trhu nevytlačilo. [2]

1.1.2 CSS styly

Zkratka CSS znamená Cascading Style Sheets - kaskádové styly. Důvodem pro vytvoření CSS bylo rozšíření možností správy vzhledu stránky a její oddělení od obsahu (HTML). Sám o sobě je soubor s příponou .css nepoužitelný, jeho funkce se projevují až v HTML dokumentu. Existují tři způsoby deklarace CSS atributů:

- přímý styl (inline) - využívá zápis pomocí atributu style="..."v rámci otevírací značky prvku
- stylesheet (internal) - pravidla pro formátování textu jsou zapsána do hlavičky dokumentu mezi značky <style></style>
- externí stylesheet (external) - připojení souboru s příponou css, na který odkazuje značka <link>

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

{ atribut: hodnota }

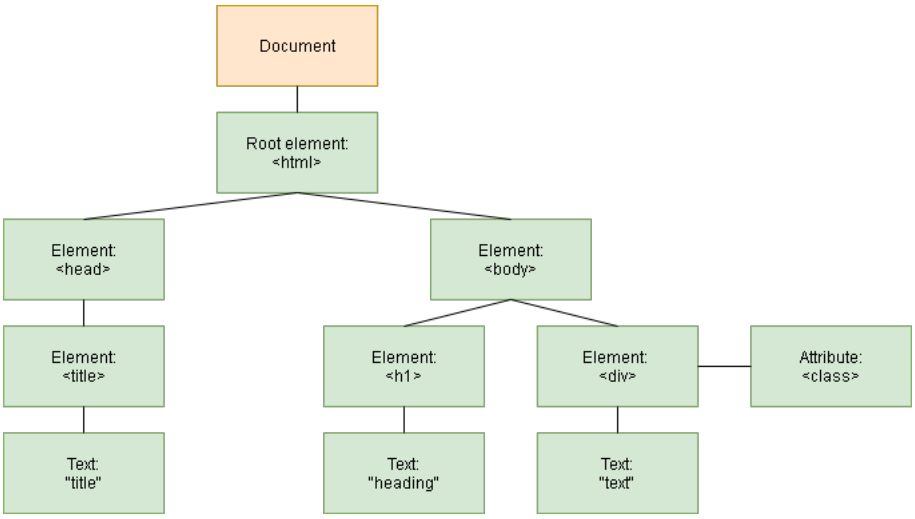
Příklady atributů CSS:

- 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]

1.1.3 Rozhraní DOM

DOM (document object model) je reprezentací dokumentu využívanou prohlížečem, reprezentuje strukturu prvků obsažených v dokumentu v podobě stromu - každý uzel (node) reprezentuje HTML prvek. Lze jej využít také pro reprezentaci dokumentů XML a XHTML. Prohlížeče poskytují API - aplikační programovací rozhraní, tudíž umožňují programům a skriptům čtení a manipulaci s obsahem, strukturou a styly dané webové aplikace.



Obr. 1.2: HTML DOM

1.1.4 Imperativní a procedurální programování

Imperativní programování popisuje průběh programu pomocí posloupnosti příkazů s přesným postupem. Program se skládá ze stavu (proměnných) a příkazů, které stav mění. Procedurální programování spadá pod imperativní paradigma, program je sestaven z procedur (podprogramů), jejichž vnitřní běh probíhá postupně po jednotlivých příkazech. Tento přístup již částečně nabízí možnost abstrakce. Při čtení kódu může být programátorovi zřejmé dle názvu procedury, jakou funkci zastává, nemusí tuto informaci zjišťovat ze samotné implementace algoritmu.

Základními stavebními bloky imperativního programování jsou cykly (for, while, do while), podmínky (if, else if, else) a v případě procedurálního programování procedury (také nazývány funkce, podprogramy). [5]

1.1.5 Objektově orientované programování

Objektově orientované programování je způsob návrhu softwaru, při kterém je kladen důraz na zapouzdření jednotlivých součástí programu do objektů a tříd. Jako objekt označujeme datovou strukturu, která obsahuje potřebná data, ale také všechny operace, které je nad touto datovou strukturou možno provádět. Třídu je možno si představit jako formu, dle které můžeme objekty vytvářet (všechny objekty odvozené od třídy musí splňovat standardizovanou strukturu). V objektově orientovaném programování se snažíme zapouzdřit data a související funkce (v kontextu třídy je nazýváme metodami) za účelem větší strukturovanosti a možnosti opakovatelného

použití kódu. Dalším důležitým principem OOP je omezení přístupu, kdy u jednotlivých členských proměnných a metod objektu můžeme volit, zda je možno k nim přistupovat zvnějška (tedy z metod nenáležících tomuto objektu).

Díky třídám a objektům jsme schopni v kódu snáze popisovat situace a interakce v reálném světě. Toto je možno demonstrovat na zjednodušeném příkladu nákupního košíku. Nákupní košík reprezentovaný jako objekt bude přinejmenším obsahovat list produktů jako členskou proměnnou a metody pro vrácení obsahu, přidání produktu do košíku, odstranění produktu z košíku.

Mezi další důležité vlastnosti paradigmatu OOP patří dědičnost, tedy možnost opětovného využití kódu a jeho rozšíření, dále polymorfismus, schopnost jazyka zacházet s různými objekty, které vykazují podobosti, pomocí stejného kódu. [6]

1.1.6 Deklarativní a funkcionální programování

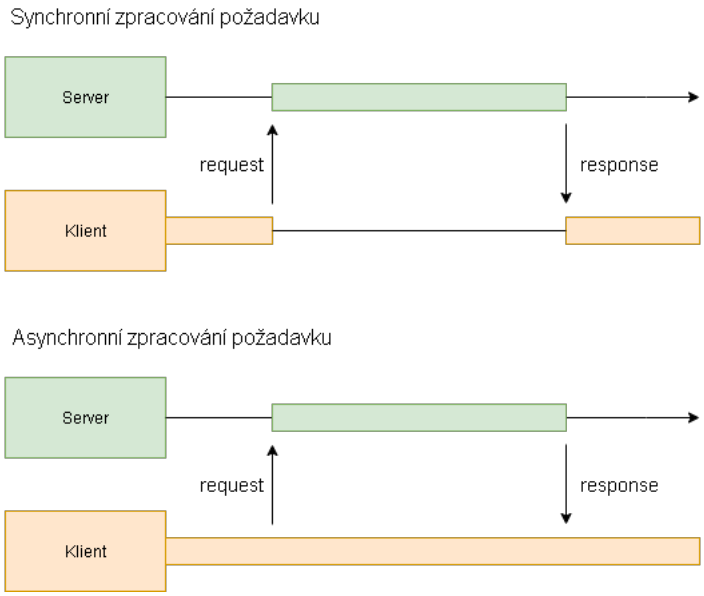
Deklarativní programování spočívá v zápisu kódu takovém, že vyjadřuje, co má program provést, ale nikoli specifickou implementaci všech částí. Jazyky, které dovolují takovýto styl programování, jsou např. Rust, JavaScript, Haskell či Python. Tyto jazyky tedy poskytují větší míru abstrakce, není zde potřeba explicitně implementovat algoritmy například pro filtrování, mapování či seřazení kolekcí, tato funkcionalita je součástí samotného jazyka a obvykle je snadno použitelná pomocí jednoduchého zápisu (funkce, klíčové slovo, operátor, ...). [7]

Dalším znakem obvyklým pro jazyky podporující deklarativní a funkcionální programování je možnost ukládat funkce do proměnných a tvorba anonymních funkcí. Toto dovoluje výrazně rozšířit míru abstrakce v kódu pomocí předávání funkcí jako vstupů jiných funkcí. Příkladem této abstrakce v JavaScriptu je například funkce map, která přijímá funkci (i anonymní), která deklaruje, co se má s jednotlivými členy kolekce stát, tedy co bude výstupem funkce map. Jazyk JavaScript dovoluje využívat funkcionálního programování, v kontextu této práce budou některé tyto vlastnosti využity spolu s principy objektově orientovaného programování. [8]

1.1.7 Jazyk JavaScript

JavaScript je vysokoúrovňový programovací jazyk založený na prototypech, který využívá principy imperativního, funkcionálního i objektově orientovaného programování. Obecně platí, že HTML zastává předání obsahu a informací v podobě srozumitelné pro prohlížeč, CSS slouží k úpravě vzhledu stránky a JavaScript upravuje chování a interakce webové stránky či aplikace. [9]V jeho definici se často objevuje i slovo „klientský“, neboť je nejběžněji využíván ve webových prohlížečích na straně klienta. JavaScript je ve výchozím stavu synchronní, což znamená, že program čeká na dokončení kódu předchozího.

To může být nevýhodou např. při zasílání requestu, který jako odpověď získá velké množství dat. Asynchronního chování však je množné v Javascriptu dosáhnout třemi způsoby, prvním je callback (funkce předaná jako argument jiné funkci). Další způsoby - promise a async/await - jsou popsány v části „Axios a promise“.



Obr. 1.3: Práce programu při zaslání požadavku na server synchronně a asynchronně

JavaScript využívá dynamické datové typy, což je u skriptovacích jazyků obvyklé. Jedná se o „weakly-typed“ programovací jazyk, což znamená, že datový typ proměnné se automaticky přizpůsobuje dané situaci. V určitých situacích (např. při matematických operacích) se tedy k řetězci bude chovat jako k číslu nebo naopak, zatímco u „strongly-typed“ jazyka by kompilátor nahlásil chybu. Pokud se chce programátor naučit JavaScript a vychází z objektově orientovaných jazyků typu C++, Java nebo C#, nalezne mnoho podobností, které mu práci ulehčí. V JavaScriptu je možné využívat základní cykly typu for, while, if/else atd., standardní matematické i binární operátory, celková syntax je podobná, liší se však zásadně v sémantice. [10]

1.1.8 Jazyk 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 několika změnami názvu. JavaScript se jako název ustálil 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, jejímž výstupem byla první verze jazyka „ECMAScript“. [11] S rozvojem webových aplikací rostla také nutnost JavaScript postupně aktualizovat a přinášet nové nástroje. Od roku 2015 vychází každý rok update přinášející nové funkce a vylepšení. [12] Z těchto nových funkcí stojí za zmínku:

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

1.1.9 Formát JSON

JavaScript Object Notation (JSON) je velmi populární formát pro reprezentaci dat, využívaný zejména pro webová API a konfigurační soubory. JSON soubor obsahuje „zprávu“, jejíž „poslem“ je API. Očekávaným obsahem 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 nejdříve parsovat - provést syntaktickou analýzu. Použití převodu z JSON zápisu na nativní objekt je v jazyce JavaScript následující: „JSON.parse(názevSouboru)“. [13] JSON ve výchozím stavu podporuje datové typy:

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

1.1.10 Knihovny rozšiřující JavaScript

Existuje velké množství způsobů, jak si programátor může ulehčit svoji práci. K tomuto účelu slouží knihovny, jenž plní určité služby, pro které by bylo složité nebo časově náročné tvořit vlastní implementaci. Knihovna shromažďuje do jednoho či více souborů třídy, 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 byly při realizaci této bakalářské práce mimo jiné použity tyto knihovny:

Nástroj 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 je vyřešen problém se zpětnou kompatibilitou. [14] Příklad překladu z ES5 na starší verzi JavaScriptu:

```
//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; });
```

Jazyk 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. Typové programování však může podpořit čitelnost a zjednodušit hledání chyb ve vlastním kódu. TypeScript je rozšíření JavaScriptu, které pracuje s datovými typy a 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 někdy nutné doplnit předponu „@types/“, čímž lze stáhnout balíček rozšíření pro TypeScript, pokud jej knihovna nepodporuje již ve výchozím stavu. V současné době bylo komunitou kolem TypeScriptu otypováno přes 1000 knihoven jazyka JavaScript. Velké množství 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. [15]

Knihovna ESLint

ESLint je 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 ve složce projektu. V něm najdeme soubor pravidel, díky kterým si můžeme určit, zda nástroj 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. [16]

Knihovna Prettier

Knihovna Prettier slouží k formátování kódu a podporuje mnoho jazyků. Po jejím nainstalování a vhodném nastavení se s každým uložením kód sám zformátuje pro optimální čitelnost, jednotnost a přehlednost. Lze ji jednoduše integrovat do populárních editorů kódu pomocí rozšíření. Konfigurace knihovny se provádí v souboru .prettierrc.

1.2 Back-end

1.2.1 Systém 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 ve složce JavaScriptového programu vytvoří podsložka node_modules, do které se jednotlivé balíčky instalují. Výhodou Node.js je rychlost, vysoká škálovatelnost a snadný přístup k open-source balíčkům. [17] [18]

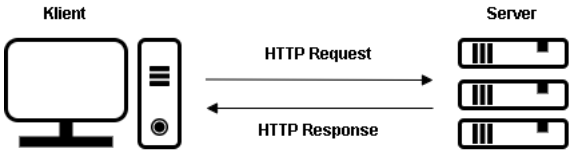
Správce NPM

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 program NPM. 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. [19] 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

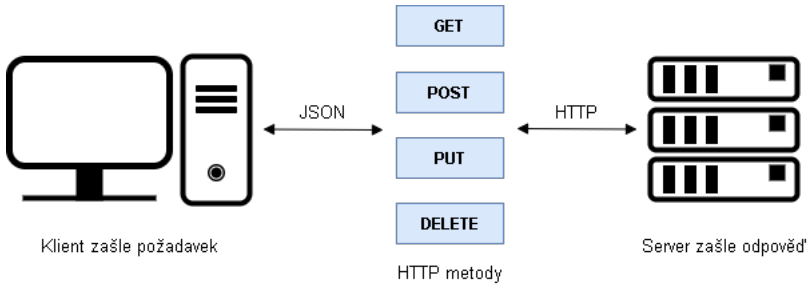


Obr. 1.4: Základní komunikace pomocí HTTP protokolu

1.2.3 Aplikační programovací rozhraní a architektura REST

API je zkratka pro Application Programming Interface. 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 nebo XML.[20] [21] 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



Obr. 1.5: Komunikace mezi klientem a serverem pomocí REST API

1.2.4 Rozhraní Express

Express je rozhraní pro Node.js, které zjednodušuje komunikaci se serverem. Můžeme pomocí něj definovat cesty (routes), které určují adresu, na jejíž požadavky má daná část programu reagovat, a specifikace, jak se má program chovat, když na server přijde dotaz (request). 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.js. Zjednodušení spočívá v tom, že odpovědi (responses) ze serveru zasíláme jako celek, program si je zpracuje a vyhodnotí, jak se má v dané situaci chovat. [22]

1.2.5 Technologie AJAX

AJAX je zkratka pro „Asynchronní JavaScript a XML“. Jedná se o kombinaci technologií HTML (XHTML), JavaScriptu, XML a XMLHttpRequest. Pointa spočívá v možnosti odesílání a získání dat ze serveru, přičemž není nutné přenahrávat celou stránku. Typickým využitím AJAX jsou např. našeptávače, kdy se mění pouze samotná komponenta a není nutná obnova celé stránky. Název AJAX může být lehce zavádějící, neboť v dnešní době se místo formátu XML hojně využívá formát JSON. Vzhledem k tomu, že JSON není výchozím formátem technologie AJAX, je nutné při použití v JavaScriptu využít metodu *parse*. [23]

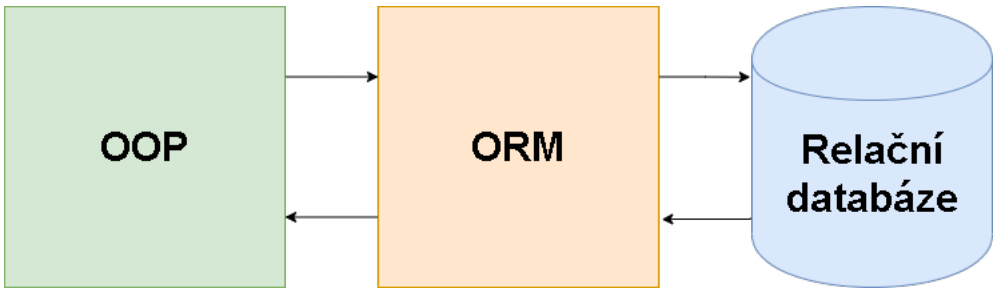
1.2.6 Protokol CORS

V prohlížečích je ve výchozím stavu zavedena zásada stejného původu - stránka či aplikace pochází 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, zda-li jí bude přístup k datům udělen. Pomocí CORS lze díky systému „whitelist“ omezit přístup všem doménám, které nejsou součástí tohoto listu.[24]

1.2.7 Objektově relační mapování

Objektově relační mapování (ORM) je způsob realizace ukládání, získání, aktualizování a mazání dat mezi objektově orientovaným programem a relační databází. Díky ORM je možné automatizovat proces konverze objektů do údajů, které je relační databáze schopna zpracovat. Programátor tedy pracuje s abstrahovaným rozhraním bez nutnosti práce přímo s relační databází, může ji spravovat pomocí vybraného objektově orientovaného programovacího jazyka. Pro Node.js je jedním z nejpoužívanějších softwarů pro ORM knihovna Sequelize. [25]



Obr. 1.6: Princip ORM

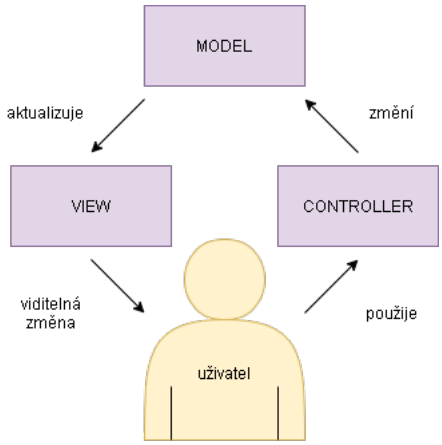
1.3 Front-end

1.3.1 Knihovna React

React je open source knihovna vyvinutá společností Facebook pro vytváření uživatelských rozhraní pomocí webových komponent. Klade velký důraz na interaktivitu a dynamičnost. Jeho hlavní využití lze nalézt zejména ve webových aplikacích. Filozofií Reactu je uchopení UI jako kompozice menších komponent, které se spojují v jeden celek, naproti tomu potlačuje nutnost využití dědičnosti. V současné době se pro vytváření front-endu hojně využívají také knihovny Angular.js, Vue.js a Svelte. Oblíbenou schopností knihovny React je možnost sloučení HTML a JavaScriptu do zdrojového kódu označovaného JSX. React běžně zastává pozici „V“ v MVC modelu, což je zkratka pro Model-View-Controller, React tedy obstarává finální formátování a zobrazení dat uživateli. [26]

Součásti modelu MVC:

- 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



Obr. 1.7: Model MVC

Popsat detailně celou knihovnu React v rámci bakalářské práce není vzhledem k rozsahu této knihovny možné ani vhodné, v náledujících sekcích budou však přiblíženy alespoň ty koncepty, které jsou v rámci praktické části - webové aplikace - používány nejčastěji.

Syntax JSX

JSX je syntaktické rozšíření jazyka JavaScript. Podobá se jazyku HTML, proto je pro mnoho uživatelů přívětivý. Filozofií JSX je sloučení logiky programu a UI. Pomocí JSX jsou vytvářeny jednotky zvané komponenty, což jsou větší celky kódu sloučené do jednoho souboru. Díky JSX je možné vkládat do komponent po boku HTML prvků i jiné React komponenty. Pro převádění jazyka zpět do JavaScriptu, HTML a CSS je využíván kompilátor Babel. React explicitně nevyžaduje využití JSX, většina uživatelů ho však používá. Pro sloučení JSX s TypeScriptem lze využít typovou verzi JSX s názvem TSX. [27]

Komponenty

Pomocí komponent je možné rozčlenit uživatelské rozhraní do libovolného množství menších částí, které lze využívat opakovaně. Pro zobrazení webové aplikace je využívána statická metoda *ReactDom.render*. Jedná se o jedinou metodu podtřídy

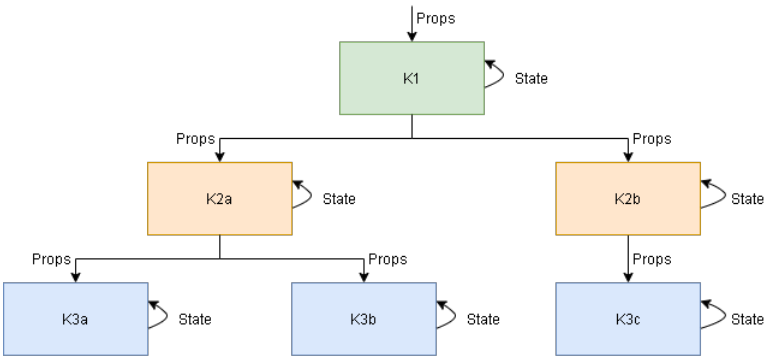
React.Component, která musí být definována. Každá komponenta má svůj životní cyklus, jednotlivé části jeho průběhu lze ošetřit pomocí základních metod životního cyklu *componentDidMount* (komponent se „připojil“), *componentDidUpdate* (komponent byl aktualizován) a *componentWillUnmount* (komponent bude „odpojen“). [28] Pod pojmem připojit (mount) se v tomto kontextu rozumí, že instance komponenty byla vytvořena a vložena do DOM. Aktualizace komponenty znamená změnu jeho vlastností či stavu. Komponenta je odpojena, pokud je z DOM odstraněna. [29]

Vlastnosti a stav

Props (zkratka pro properties - vlastnosti) poskytují způsob, jak předávat data dynamicky z jedné komponenty do druhé. Při každé změně *props* je daná komponenta přerenderována. Pomocí syntaxe obdobné jako u HTML atributů je možné v JSX tyto *props* přidat do zápisu komponenty. Standardně jsou předávány pomocí klíčového slova *this*, značící instanci dané třídy.

V rámci stromové struktury komponent lze předávat data vždy pouze z menší míry zanoření do větší. *Props* je možné pouze číst, není možné je přenastavit. Interaktivitu lze tedy zajistit pomocí *state* (stavu), který lze měnit. Ať už se jedná o *state* či *props*, komponenta, která je potomkem jiné komponenty, neví, zda přijímá *props* nebo *state*, data přijímá jako *props*. V případě změny oproti předchozímu stavu se přerenderovává. Protože není možné měnit stav ze zanořenější komponenty do méně zanořené, využívá se stavu jako samostatně deklarovaných globálních proměnných. Pomocí funkcí z méně zanořených komponent, ke kterým zanořenější komponenty mají přístup, stav již měnit lze. Všechny komponenty, kterých se daná změna týká, budou přerenderovány.

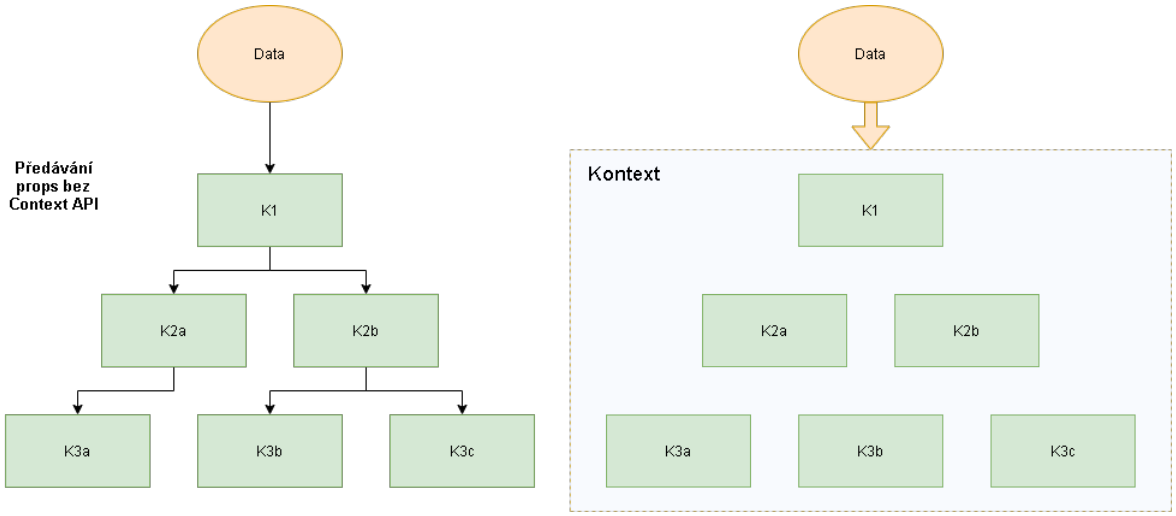
Je-li potřeba získat state z props, lze využít metodu *getDerivedStateFromProps(props, state)* stav se mění podle změny props, užívání této metody se však obecně nedoporučuje. [30]



Obr. 1.8: Předávání props přes komponenty a změna jejich stavu

Kontext

Context API řeší problém velkého množství zanoření komponent, kdy - pokud je potřeba dostat props z nejméně zanořené do n-tého zanoření - je potřeba props přeposlat přes každé jednotlivé zanoření a tím se přerenderovávají i komponenty, které nejsou nutně ovlivněny. Pomocí kontextu lze předávat props tak, že je zpracovávají pouze ty komponenty, které dané props zajímají/ovlivňují je. Takovému typu předávání dat se říká implicitní, kdežto předešlý případ předává data explicitně. Pro předávání kontextu je využíván hook (viz dále v texu) *useContext*. [28]



Obr. 1.9: Předávání props v Context API

Hooks

Hooks, nebo počeštělé „hooky“, jsou možností knihovny React, kterou lze nahradit třídy. Hooky umožňují funkcionálním komponentám se „zaháčkovat“ na globální stav Reactu a jeho životní cyklus. [28] Hooky zpravidla začínají slovem *use*. V rámci bakalářské práce jsou používány tři hooky:

- useState - nejčastěji ve tvaru:

  ```
  const [X, setX] = setState(0)
  ```

  lze pomocí něj definovat konstantu X, jejíž stav je upravován funkcí *setX*. Hodnota v kulatých závorkách nastavuje výchozí hodnotu X

- useContext - umožňuje získání dat z kontextu, ve kterém je stav dané proměnné uložen

  ```
  const a = useContext(kontext)
  ```

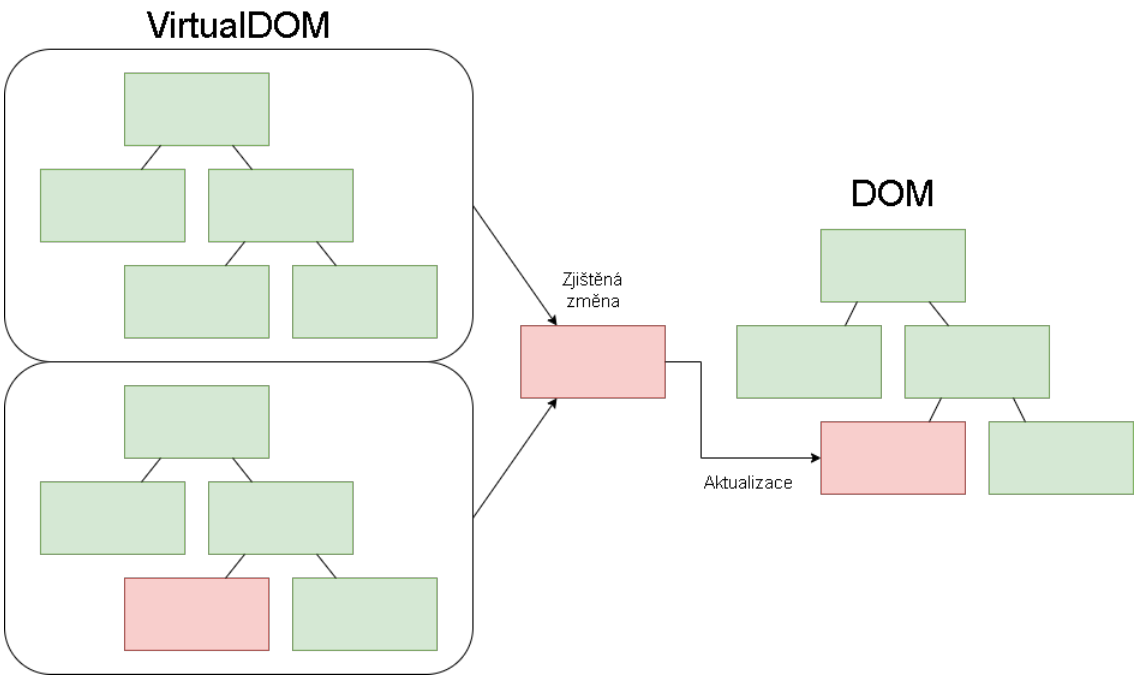
- useEffect - umožňuje funkcionálním komponentám provádět „efekty“, pokud nastane změna uvnitř „podmínky“ hooku. Příkladem z praktické části je:

  ```
  useEffect(() => {
      setPage(0)
    }, [filteredVideos.length])
  ```

  kdy se při změně délky pole *filteredVideos* nastaví aktuální stránka pomocí *setPage* na 0

Virtual DOM

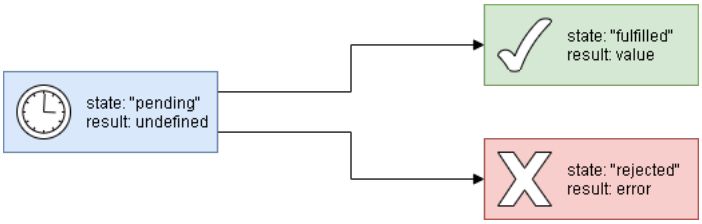
Jde o koncept, kdy je umělý (virtuální) DOM reprezentující UI uložen v paměti a synchronizován s reálným DOM. S každou změnou stavu je vždy znovu vytvořený přímo Reactem a jeho výhodou je, že pomocí klíčů pouze zjišťuje změny mezi aktuální a předchozí verzí webové aplikace, jejíž stav si zaznamenal, 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 zajistily rychlejší průběh než při práci pouze s DOM browseru. [31]



Obr. 1.10: Princip VirtualDOM

1.3.2 Knihovna Axios a objekt Promise

Je knihovna pro vytváření HTTP požadavků (requestů), založená na tzv. *Promise* - návratová hodnota (objekt) využívaná při asynchronních operacích, která může nabývat stavů *pending* (výchozí stav), *fullfilled* (operace proběhla úspěšně) a *rejected* (operace neproběhla). *Promise* se používá typicky s metodami *then*, *catch* a *finally*. S metodou *Promise.all* se lze také běžně setkat - výsledek je vyhodnocen až po úspěchu nebo neúspěchu všech *Promise*. Jinou možností, jak využívat v JavaScriptu asynchronní operace, je pomocí klíčových slov *async* a *await*. Axios umoňuje zasílat požadavky jak z browseru, tak pomocí Node.js. Axios podporuje starší verze prohlížečů, automaticky provádí transformaci JSON dat a lze díky němu např. zrušit požadavek či nastavit *response timeout*. [32]

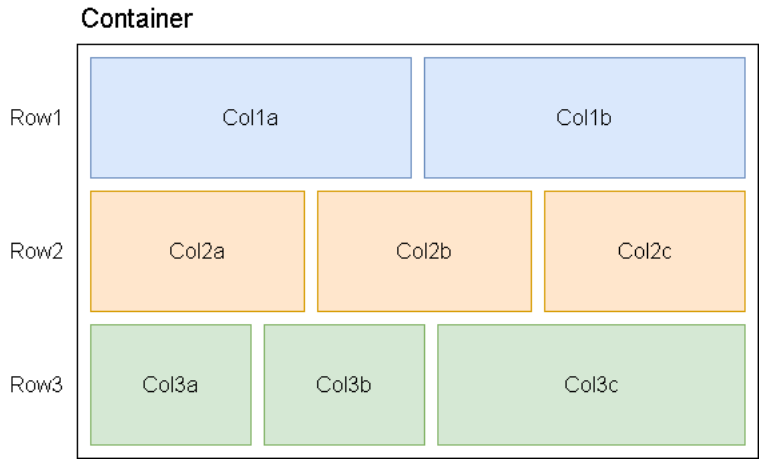


Obr. 1.11: Návratové hodnoty asynchronních operací při využití promise

1.3.3 Knihovny 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. [33] V rámci aplikace byly z této knihovny využity komponenty:

- Buttons - tlačítka
- Card - karta - grafický útvar zaobalující danou komponentu
- Layout - rozvržení stránky
- Navbar - záhlaví
- Pagination - stránkování (u výsledků vyhledávání)
- Spinners - rotující kolečko (při nahrávání)
- Tables - tabulky



Obr. 1.12: Grid systém knihovny Bootstrap

1.3.4 Knihovna Recharts

Pro grafické zobrazení statistik webové aplikace lze využít mnoho knihoven. Jedna z nejpopulárnějších pro React je knihovna Recharts, která umožňuje vytvořit velké množství typů grafů, které jsou jednoduché na implementaci. Některé z možností jsou liniový, sloupcový, kruhový aj. Vytvořené grafy jsou ve vektorovém formátu SVG, je tedy možné jednoduše a bezztrátově měnit jejich velikost.

1.3.5 Knihovna Styled Components

Jedná se o knihovnu, umožňující určitý způsob psaní kódu jazyka CSS v moderním JavaScriptu. Jednoduše umožňuje popsat CSS vlastnosti v samostatných komponentách, které lze bez nutnosti kopírování využívat na více místech. Výhodou této knihovny je celková konzistence stylu webové aplikace a možnost změnit jednoduše jakoukoliv vlastnost v celém kódu pomocí jedné komponenty.

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. [34]

1.4.1 Relační databáze

Relační databáze využívá organizace dat do tabulek, které mohou být propojené na základě společných dat. Tato propojení mezi tabulkami se nazývají relace a díky nim je možné pomocí dotazů na databázi z atomizovaných dat získat novou tabulku, která obsahuje všechna potřebná data. Relace jsou tvořeny pomocí tzv. cizích klíčů, tedy sloupců v tabulkách, které obsahují primární klíče tabulek jiných. U relačních databází je kladen důraz na validitu a konzistenci dat. Obsah těchto sloupců je při zápisu a změnách validován, zda neobsahuje klíč, který v cizí tabulce neexistuje a je pro něj definováno chování pro případ smazání záznamu z cizí tabulky (kaskádové smazání záznamů s relací, blokace smazání záznamu, který má relace, ...). [35]

Software, který uživateli dovoluje definovat, vytvářet, udržovat databázi a ovládat přístup k ní, nazýváme RDBMS - Relational database management system. Do této skupiny software spadá například MySQL, MariaDB, Oracle, PostgreSQL a další. [36]

1.4.2 Databáze 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. [37] Příkazy jsou v tomto jazyce standardně zadávány velkými písmeny. MariaDB využívá mnoho stejných příkazů, jako MySQL, jsou jimi např:

- CREATE - vytvoří nový objekt
- DROP - odstraní objekt
- SELECT - vybere z databáze
- DELETE - odstraní data z databáze
- INSERT - vloží nová data do databáze

1.4.3 Možnosti ovládání interaktivními ovladači

Herní ovladače

Pro ovládání webové aplikace herním ovladačem lze využít Gamepad API, což je rozšíření pro HTML5 vytvořené přesně pro tento účel. Toto API je podporováno všemi významnějšími prohlížeči s výjimkami Internet Explorer a WebView Android. Jednotlivým signálům vysílaným z ovladače lze přiřadit specifickou funkci. Gamepad API se skládá ze tří rozhraní. [38]

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

Gamepad API využívá objekt Gamepad a pomocí metody *navigator.getGamepads()* lze získat pole připojených Gamepadů. Pokud je na gamepadu zmáčknuto tlačítko popř. je gamepad připojen během chodu aplikace, lze tuto skutečnost zjistit pomocí eventu gamepadconnected. Každému Gamepadu je přiřazeno unikátní ID, které je zjistitelné pomocí property *GamepadEvent.gamepad*. Dalším eventem je např. *gamepaddisconnected*, reprezentující událost odpojení gamepadu. [39]

Gamepad objekt

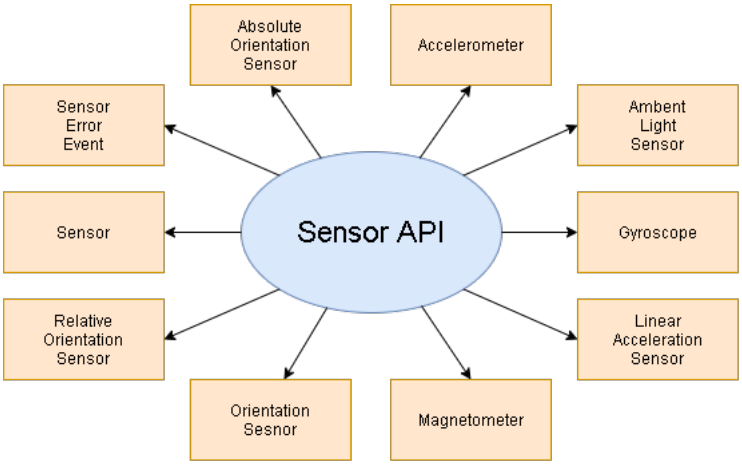
Objekt Gamepad disponuje množstvím vlastností se kterými lze dále pracovat:

- id - liší se podle prohlížeče, jedná se o řetězec obsahující informace ohledně controlleru např. id produktu, jméno přiřazené driverem
- index - autoinkrementovaný unikátní index (integer) pro identifikaci kontroleru
- connected - boolean indikující, zdali je gamepad stále připojen
- mapping - string indikující, jedná-li se o standardní gamepad
- buttons - pole GamepadButton objektů reprezentující tlačítka na daném zařízení
- axes - pole floatů sloužící k reprezentaci plynule přechodných hodnot např. u joysticku
- timestamp - reprezentuje poslední čas aktualizace gamepadu

Pro testování gamepadu byl použit Sony Dual Shock 4 - výchozí ovladač herní konzole PlayStation 4.

Akcelerometr a další ovladače

Rozhraní Accelerometer spadající pod Sensor API prohlížeče není podporováno prohlížeči Firefox, Internet Explorer a Safari, lze jím však ovládat prohlížeče Chrome, Edge a Opera. Při zavolání konstruktoru pomocí *Accelerometer.Accelerometer()* je vytvořen nový objekt *Accelerometer*. Tento objekt má tři properties (vlastnosti), které vrací double reprezentující zrychlení pro každou ze tří os [40]. Sensor API umožňuje ovládání prohlížeče pomocí vícero nestandardních zařízení, pro něž jsou vytvořena samostatná rozhraní, např. Magnetometer, Gyroscope, AmbientLightSensor a další. Pro tyto ovladače je také nutné povolení v Permissions API. Jednotlivé ovladače nejsou všeobecně podporovány a pro jejich testování je potřeba vybrat prohlížeč, který konkrétní rozhraní podporuje. [41]



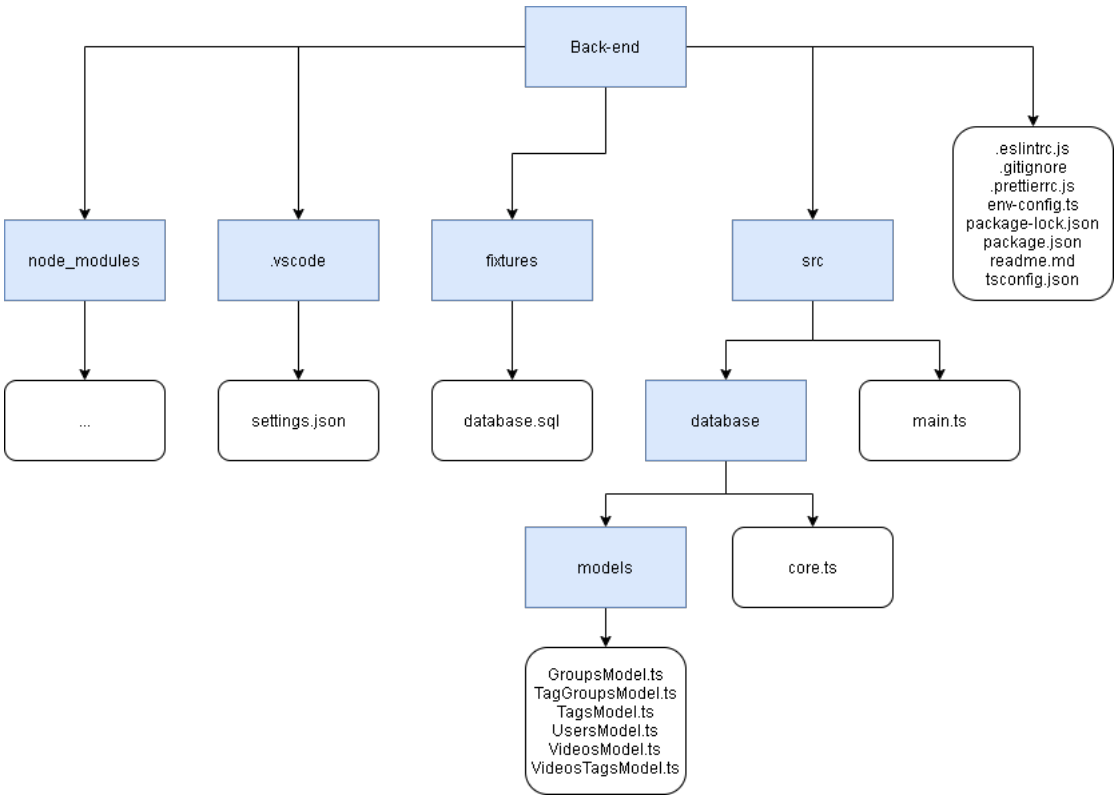
Obr. 1.13: Rozhraní spadající pod Sensor API

2 Postup vytvoření vlastní aplikace

Praktická část bakalářské práce podrobně popisuje postup vytvoření výsledného programu. V první části je chronologicky popsáno vytvoření back-endového serveru. Druhá část popisuje soubory nacházející se na front-endu. Tato část již jako celek chronologicky popsaná není, neboť paralelní práce v několika komponentách zároveň by vedla k nepřehlednostem v textu. Kapitola je tedy popisována po souborech a složkách, a to z menšího zanoření do většího.

2.1 Back-end a databáze

Tato část popisuje sestrojení back-endové části od vytvoření prázdné složky až po plnohodnotný server plnící svoji funkci. Na obr. 2.1. je po přehlednost struktura jeho souborů uvedena, aby bylo jasné, kterého souboru ve struktuře se daná část textu týká.



Obr. 2.1: Struktura Back-endu

2.1.1 Příprava pro vlastní implementaci

V této sekci je popsáno vytvoření serveru s REST API, který bude naslouchat front-endu 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 a s nimi i stejnojmenná rozšíření pro Visual Studio Code. 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 závislostí 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/). Při stažení těchto knihoven je vytvořen soubor package-lock.json, jehož účelem je zabránit nekonzistentnosti při závislosti knihoven na dalších knihovnách tím, že všechny tyto závislosti popíše a tím vytvoří časový snapshot - zámek přesně těch daných verzí knihoven, které jsou závislé přesně na těchto knihovnách v těchto verzích, pro libovolné zanoření závislotí.

Ve složce Back-end byla vytvořena složka .vscode se souborem settings.json, ve kterém bylo nastaveno provedení funkcí knihoven ESLint a Prettier na každé uložení aplikace (*formatOnSave*). Tato funkce zjednodušuje práci programátora, standardizuje a zpřehledňuje kód. Knihovny, které slouží vývojářům, ale nejsou nutné pro běh programu 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 kterých byly nakonfigurovány jednotlivé funkce těchto knihoven. Poté byla nainstalována další knihovna pro vývojáře, a to ts-node-dev, která slouží ke spouštění zdrojových kódů v TypeScriptu při každém uložení. Posledním souborem fáze příprav je soubor tsconfig.json - konfigurátor jazyka TypeScript. Tento postup byl ve složce front-end zduplikován.

Pro samotný back-end byly dále nainstalovány knihovny:

- 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

Back-endový server byl spuštěn na vlastně zvoleném portu (adresa portu byla zvolena 3001) a v souboru main.ts bylo zapsáno:

```
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 whitelist/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 Zdrojová data pro databázi

Téma bakalářské práce je součástí výzkumného projektu, který se zabývá obsahově založeným vyhledáváním v databázích obrazových a audiovizuálních děl. V rámci tohoto projektu byla vytvořena klíčová slova k existující databázi videí. Zdrojovými daty byl potom „dump“ - soubor obsahující veškerá data z této databáze. Aby bylo možné s databází pracovat, je nutné zanalyzovat a pochopit její strukturu. Databáze obsahuje šest tabulek, z čehož dvě jsou vazební (viz. poměr M:N). Struktura těchto tabulek vypadá následovně:

- Groups - ID, Name
- Users - UserId, UserName, Password, attempt, staff, administrator
- Tags - tagID, Name
- Videos - VideoID, Name, NameFromUser, Author, Year, Coment, Completed, PathID
- TagGroups (vazební) - VideoTagId, GroupID, TagID
- VideosTags (vazební) - ID, VideoID, TagID, UserID, StartTime, StopTime

Pro každou tabulku byl ve složce „models“ vytvořen databázový model, který umožňuje definovat obsah každé tabulky a datové typy jednotlivých sloupců. Každý model je exportován. Vzhledem k tomu, že data tabulek *Groups, Users* a *TagGroups* byly vyhodnoceny jako nepřidávající významnou hodnotu pro využití v rámci bakalářské práce, byly asociovány pouze tři z těchto tabulek ve standardní M:N vazbě. Tyto tabulky jsou *Tags, Videos* a *TagsVideos*. Jako příklad databázového modelu lze uvést soubor *TagsModel*.

Nejdříve je naimportován celý obsah knihovny Sequelize. V JavaScriptu existují dvě možnosti exportu - výchozí a pojmenovaný (default a named). Výhoda výchozího exportu je, že jeho import není třeba zapisovat se závorkami, nevýhodou je, že lze použít pouze jeden tento export na modul. V tomto případě je výchozím způsobem exportována funkce, která na vstupu bere instanci třídy *Sequelize* a any. Konstanta TagsModel je inicializována pomocí *sequelize.define* a pro jméno modelu 'tags' definuje jeho zobrazení jako tabulky v databázi. Sequelize automaticky k tabulce přidává atributy *createdAt* a *updatedAt*. *Name* a *tagID* jsou tedy sloupce, kterým lze nastavit jejich vlastnosti např. datový typ, primární klíč, automatickou inkrementaci nebo výchozí hodnotu. V druhé části souboru se nachází popis vytváření asociací na tabulku *VideosTags*, kde bylo pomocí metody *associate* určeno, že každý tag má k vazební tabulce vztah 1:N, k čemuž byla využita metoda *hasMany*.

```
import * as Sequelize from 'sequelize'
export default (sequelize: Sequelize.Sequelize,
 DataTypes: any) => {
  const TagsModel = sequelize.define('tags', {
    tagID: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    Name: {
      type: Sequelize.TEXT,
      defaultValue: null,
    }
  })
  TagsModel.associate = models => {
    models.Tags.hasMany(models.VideosTags, {
      sourceKey: 'tagID',
      foreignKey: 'TagID',
      constraints: false,
    })
  }
  return TagsModel
}
```

Stejný vztah na tabulku *VideosTags* má i tabulka *Videos*. Každá vazba vazební tabulky je asociována videu či tagu právě jednou (vztah 1:1), k čemuž byla využita metoda *belongsTo*. Pomocí výše zmíněných metod byl každému propojení nastaven

sourceKey a *foreignKey*, sloužící k odkazování na daná data buď z té samé tabulky (*sourceKey*) či z jiné tabulky (*foreignKey*). Pomocí *constraints* jsou nastavována omezení pro foreignKey a slouží k zabránění cyklické závislosti na sobě samotném kvůli nevhodnému návrhu asociací. Návratovou hodnotou celé funkce je *TagsModel*.

2.1.4 Vytvoření Databáze MariaDB

Celý dump této databáze byl umístěn do složky Back-end a v ní do složky fixtures. Po stažení a instalaci databáze MariaDB byla vytvořena databáze pomocí klienta příkazové řádky bash příkazem „mysql create database bakalarska_prace_db“. Pokud se jedná o první připojení do programu MySQL, je nutné nastavit heslo pro uživatele „root“. Problém, který může při tomto příkazu nastat je nenalezení programu MySQL. Pro operační systém Windows je nejprve nutné přidat cestu k programu MySQL do globální proměnné *path*. Nastavení lze provést pomocí menu „Upravit proměnné prostředí systému“ kde přes „Proměnné prostředí“ lze upravit obsah proměnné *path*. Pomocí příkazu:

„mysql -u root -p bakalarska_prace_db < databaze.sql“ byl naplněn obsah databáze daty ze souboru „databaze.sql“.

ID	VideoID	TagID	UserID	StartTime	StopTime
182	1	74	1	7	244
184	1	76	1	62	172
187	1	76	1	181	182
188	1	76	1	198	200
189	1	76	1	216	217
190	1	76	1	234	235
193	21	80	2	0	51
194	21	81	2	54	65
195	21	78	2	(NULL)	(NULL)
196	21	80	2	54	65
198	21	82	2	67	70
199	21	83	2	(NULL)	(NULL)
201	21	85	2	63	70
202	21	84	2	66	70
203	339	86	2	0	133
204	339	87	2	0	133
205	339	89	2	0	133
206	339	77	2	134	194
207	339	130	2	196	344
209	339	91	2	0	25
210	339	91	2	50	67

Obr. 2.2: Ukázka dat ve vazební tabulce pomocí grafického klienta HeidiSQL

Pro možnost přístupu do databáze skrz aplikaci byl pro back-end vytvořen soubor env-config.ts, který uchovává přihlašovací údaje pro přístup k databázi. Tyto přístupové údaje jsou zpracovány v souboru core.ts.

Jeho obsah je následující:

```
const config = {
  DB_HOST: '127.0.0.1', // localhost
  DB_USER: 'root',
  DB_PASSWORD: 'heslo',
  DB_DATABASE: 'bakalarska_prace_db',
  DB_PORT: 3306,
}

export default config
```

2.1.5 Soubor Core

V tomto souboru se nachází konfigurace objektově relačního mapování prostřednictvím knihovny Sequelize. Při vytvoření nové instance třídy *Sequelize* je nejprve nutné vložit přihlašovací údaje k databázi - jméno databáze, jméno uživatele, heslo, host a port. Tyto informace jsou získány ze souboru env-config.ts. Další nastavení zahrnuje:

- dialect - nastavení databázového jazyka, v tomto případě 'mariadb'
- logging - zobrazení každého vykonaného SQL dotazu do konzole
- operatorAliases - vytváření specifických řetězců jako aliasů pro operátory
- pool - způsob, jak zachovat připojení aktivní pro zamezení vytváření nových připojení
- (define) charset - sada znaků
- (define) collate - způsob řazení informací
- (define) timestamps - vytváření časových atributů createdAt a updatedAt

Při zavolání metody *sequelize.authenticate* je možné vyzkoušet připojení k databázi. Při použití metody *then* je vrácena Promise - objekt reprezentující splnění či nesplnění asynchronní operace. Při nesplnění je daný error zachycen metodou *catch*. Objekt models uchovává obsah databázových modelů pomocí jejich importu a také instanci třídy *Sequelize*, jež naplnila konstantu *sequelize*. Díky následující metodě *Object.keys* jsou realizovány vazby mezi jednotlivými modely. Objekt *models* je ze souboru exportován.

2.1.6 Soubor Main

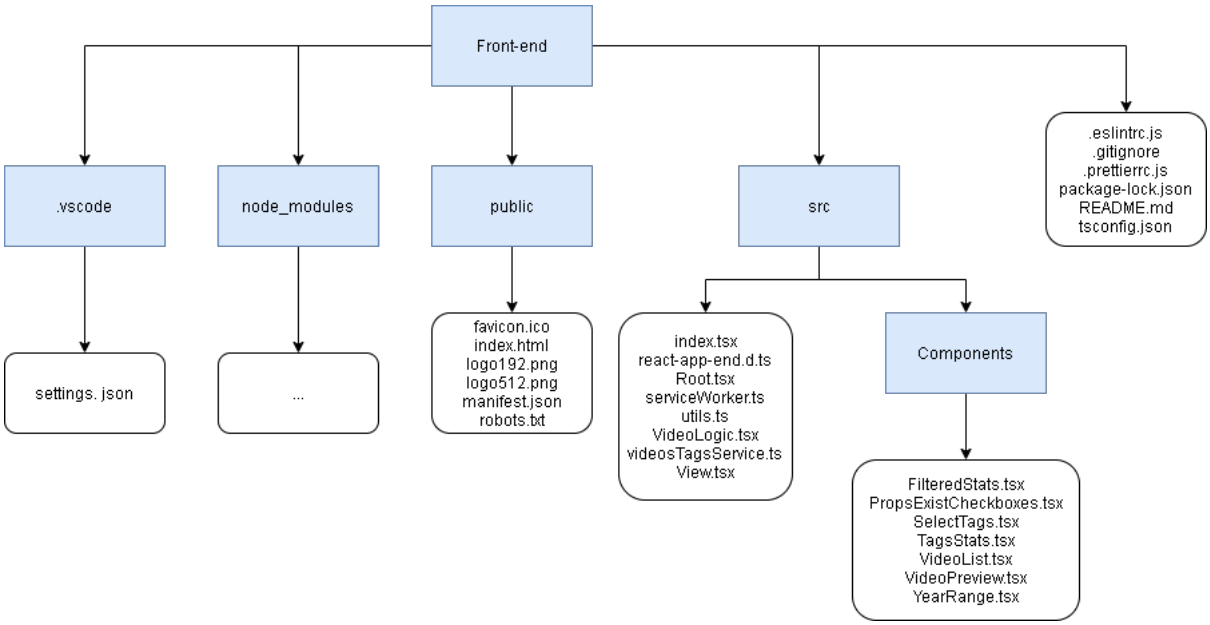
Nyní, když jsou data nahrána v databázi, je potřeba definovat, jak se s nimi bude nakládat a co přesně budou vracet při přijmutí requestu (požadavku). Vzhledem k tomu, že aplikace data pouze zobrazuje, je jediným nutným typem requestu „get“, který byl realizován metodou *app.get*. Tyto requesty jsou 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.

V databázi určené pro bakalářskou práci se nenachází velké množství dat, je tedy bez problému možné poslat celý obsah databáze na front-end jako jeden JSON soubor. Tento endpoint byl pojmenován „all-data“. Pokud by se v aplikaci nacházelo větší množství dat, je možné využít více endpointů v rámci optimalizace filtrováním na straně serveru. Při získání requestu se Express pokusí odeslat jako response (odpověď) JSON soubor, kdy v prvním zanoření se nachází videa a jejich data a pomocí vazeb v druhém zanoření pro každé video jejich tagy a data těchto tagů.

Je-li operace úspěšná, odpověď v podobě tohoto JSON souboru se odešle s potvrzujícím statusem 200 (OK). Pokud se operace nezdaří, chybové hlášení je vypsáno do konzole na back-endu a následně zaslána na front-end se statusem 500 (Internal Server Error). Aby aplikace naslouchala možným requestům, je nutné využít metodu *app.listen()* a nastavit jí správný port. Pokud operace proběhla úspěšně, vypíše se v konzoli pomocí metody *console.log()*: „App listening on port 3306!“ Správné fungování back-endu lze ověřit při zadání „localhost:3001/all-data“ do vyhledávače v prohlížeči. Pokud je back-end spuštěný, zobrazí se na této adrese JSON soubor obsahující veškerá data z databáze. Pro přehlednost a příjemnější testování lze v prohlížeči Google Chrome stáhnout rozšíření JSON formatter, které daný JSON zobrazí v přehledné stromové struktuře.

2.2 Front-end

Podobně jako na back-endu je i v této kapitole pro přehlednost přidán diagram struktury celé složky Front-end již na úvod.

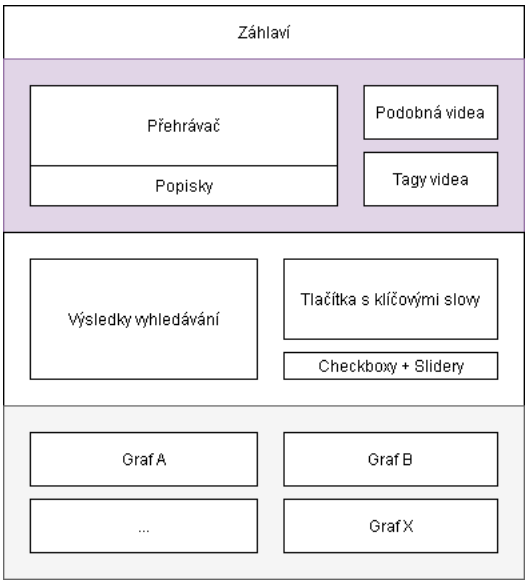


Obr. 2.3: Struktura Front-endu

2.2.1 Struktura zobrazené části webové aplikace

Než bude uvedeno, jak byly jednotlivé části front-endu webové stránky implementovány, je vhodné pochopit širší kontext/design - rozdělit si viditelnou část aplikace na určité úseky. Aplikace zobrazená uživateli se skládá ze tří hlavních sekcí - zobrazení videa a souvisejících informací, filtrování výsledků vyhledávání a zobrazování statistik.

V první části se nachází přehrávač videa, výčet jeho klíčových slov a navrhovaná videa na základě počtu shodných tagů. Ve filtrační sekci je možné počet zobrazovaných výsledků omezit vstupem pro vyhledávání výrazů v názvu videa či jeho autora, jejichž seznamem lze procházet pomocí stránkování. Dále lze filtrovat podle tabulky klíčových slov a pomocí checkboxů "ifExists", které vyfiltrují videa, která neobsahují danou informaci. Při zakliknutí checkboxu year se objeví dva slidery omezující rozsah výsledků podle roku vytvoření. Třetí sekce využívá pro grafickou reprezentaci databáze grafy knihovny recharts, které lze rozdělit na dynamické a statické podle toho, zdali jsou jejich data získávána z vyfiltrovaných videí nebo ze všech videí. Jednotlivé sekce jsou pro přehlednost a příjemnější vzhled odděleny barvou pozadí.



Obr. 2.4: Struktura zobrazované části webové aplikace

2.2.2 Vytvoření nové aplikace v knihovně React

V nové složce s názvem Front-end byla vytvořena nová aplikace knihovny React pomocí příkazu „npx create-react-app Front-end –template typescript“. Výchozí port takto vytvořené aplikace je 3000. Ve složce se objevily nové soubory, z nichž některé nesou stejný název a plní stejnou funkci jako na back-endu:

- 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
- robots.txt - instrukce pro vyhledávače

a složky:

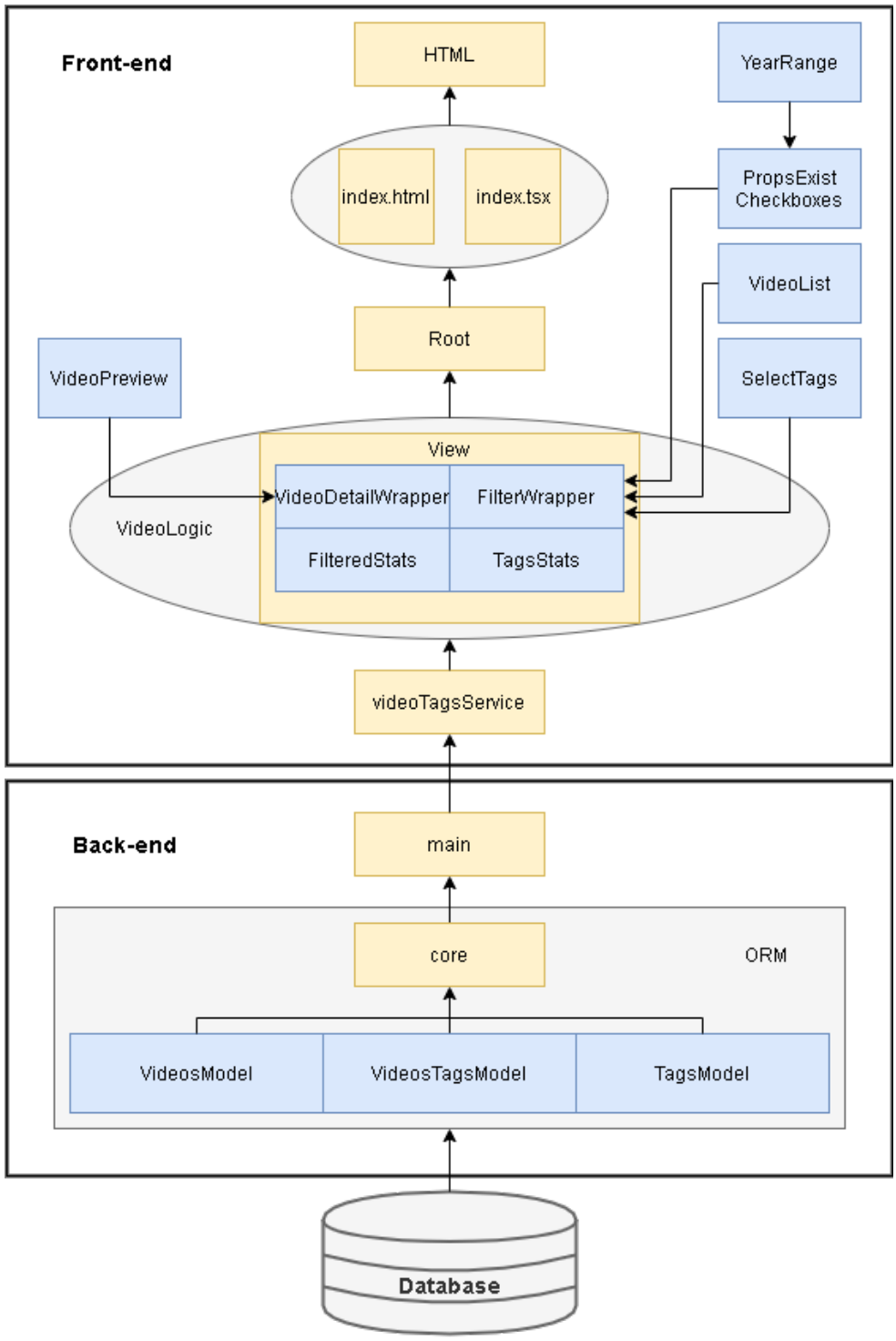
- public - soubory pro vytvoření výchozí aplikace knihovny React. Obsahuje základní soubor index.html, který musí nutně v každé webové aplikaci být. Soubor, jehož obsah je kromě deklarce typu dokumentu <!DOCTYPE html> celý zaobalen do značek <html>, obsahuje kromě metadat obalených značkami <head> také upozornění zobrazené při zákazu JavaScriptu na dané stránce a také s id hlavního souboru aplikace (ve výchozím stavu App - v této aplikaci root), obojí zaobaleno značkami <body>. V této složce je možné nastavit ikonku „favicon.ico“ zobrazující se na kartě prohlížeče nalevo před jejím jménem, dvě loga a manifest knihovny React.

- src - soubory pro samotný zdrojový kód
- node_modules - obsah používaných knihoven

Pomocí příkazu „npm run start“ se aplikace spustí. Každá webová aplikace či stránka využívá jazyk HTML, který spouští soubor index.html, který je v tomto případě exportován jako modul do prohlížeče knihovnou React. V aplikaci knihovna React zároveň do prohlížeče exportuje korespondující soubor index.tsx, který spouští metodu ReactDOM.render - zobrazení hlavního souboru aplikace. Hlavní soubor byl v této aplikaci pojmenován Root (ve výchozím stavu App). V souboru se nachází také metoda serviceWorker.unregister(), která umožňuje rychlejší práci v režimu offline. Z hlavního souboru Root.tsx je exportována funkce „Root“, která vrací komponenty VideoLogic (logika - popis chování) a View (styl - vzhled).

```
<VideoLogic>
  <View />
</VideoLogic>
```

V následující části budou popsány jednotlivé soubory, nacházející se na front-endu.



Obr. 2.5: Cesta dat přes jednotlivé soubory z databáze až k uživateli

2.2.3 Soubor videoTagsService

Tento soubor slouží k zaslání requestu na back-end pomocí knihovny axios a také k otypování získaných objektů *Tag*, *VideoTagLink* (vazby) a *Video*. Konstanta *BE_URL* uchovává adresu back-endového serveru, exportovaná funkce *getVideosTags* poté slouží k asynchronnímu vyslání právě get requestu na endpoint */all-data* back-endového serveru. Do konstanty *res* je uložena odpověď back-endového serveru, která je datového typu *AxiosResponse<Video[]>*. Návratovou hodnotou funkce je pole objektů *Video*, které bylo získáno metodou *res.data*.

2.2.4 Struktura souboru VideoLogic.tsx

Pro přehlednost kódu je vhodné oddělit logiku a vzhled do dvou souborů, kterými jsou v tomto případě soubory VideoLogic.tsx a View.tsx. V souboru Videologic.tsx se pod importy potřebných knihoven či jejich částí nachází definice datových typů pro stavové proměnné. Typ *VideoCtxApi* slouží k udání datových typů proměnných nenastavených v rámci stavové komponenty a také vstupů a výstupů funkcí. Obecně jsou ve stavu deklarovány ty proměnné, které mají být ve výchozím stavu nebo při přijetí dat naplněny nějakou hodnotou a v kontextu ty, které budou data získávat právě až ze stavových proměnných. Vlastnost knihovny React zvaná „Context“ slouží k poskytnutí cesty pro předání dat přes stromovou strukturu komponent bez nutnosti procházení každé úrovně stromové struktury. Tímto je aplikaci ulehčeno hledání dat.

Funkce getSimilarVideosByTags

Funkce *getSimilarVideosByTags* slouží k získání seznamu videí, která obsahují alespoň jeden společný tag s videem v přehrávači a jsou seřazena podle počtu shodných tagů. Na vstupu bere *selectedVideoId* (video vybrané v přehrávači) a *allVideos* (všechna videa). Pokud žádné video není vybrané, funkce vrátí prázdný řetězec. Konstanta *computedVideos* získá pole objektů, ve kterém se nachází ID všech videí a výčet jejich unikátních tagů. Do konstanty *currentVideo* je uloženo právě přehrávané/vybrané video přes shodu *VideoID* vstupního *selectedVideoId* a *computedVideos*. Návratovou hodnotou je potom konstanta *videoMatches*, která nejprve vyfiltruje ze seznamu vybrané/přehrávané video, dále jej namapuje a do konstanty *matchCount* ukládá počet shod unikátních tagů, vyfiltruje všechny bez shody a seřadí je podle počtu shod.

Metoda componentDidMount

Jedná se o jednu z metod cyklu React Lifecycle a volá se při „připojení“ dané komponenty. Význam metody *tick()* je hlouběji popsán v částech textu týkajících se možností ovládání herním ovladačem - gamepadem. Nejdřívě se stav booleanu *loading* pomocí *setState* nastaví na true - aplikace se načítá. Dojdou-li úspěšně data ze serveru, proměnné se naplní relevantními daty, která jsou uložena do vhodných stavů. Do přehrávače se nahraje první video seznamu a *loading* se změní na false. Funkce *onChangeFilterInput* nastavuje hodnotu ze vstupu uživatele do stavu proměnné sloužící k filtraci výsledků.

2.2.5 Metoda render

Metoda render obsahuje veškerý zobrazovaný obsah a při každé změně se znovu volá. V této aplikaci ji lze rozdělit na dvě části - naplnění proměnných, jejichž změna se musí projevit na webové aplikaci a návratová hodnota.

Naplnění proměnných při změně stavu

Zde se nacházejí filtry a výsledky filtrování, které se dynamicky mění. Obsahuje následující funkce:

- filterNameValue - ze stavu proměnné *filterNameValue* odstraňuje diakritiku pomocí knihovny „diacritics“ a velká písmena mění na malá metodou *toLowerCase()*
- filteredVideos - mění stav výsledků vyhledávání na základě jednotlivých filtrů - z inputu uživatele podle jména a tagů, podle sliderů pro filtraci na základě roku vytvoření a podle *ifExists* checkboxů. Využívá funkci *isNilOrEmpty* z pomocného souboru utils.ts, která řeší možnost nevhodně zpracovaného booleanu, kdy nula je v daném kontextu validní, ale ostatní falsy hodnoty (undefined, NaN, null, false,...) nejsou.
- selectedVideos - do stavu ukládá právě vybrané/přehrávané video
- allTags - upravuje seznam tagů pro filtrování
- allUniqTags - ze seznamu tagů filtruje duplicity
- similarVideosByTags - do stavu ukládá změny seznamu videí obsahujících stejné tagy jako vybrané video

Návratová hodnota metody render

Provider daného kontextu je komponenta umožňující jiným komponentám provádět změny v tomto kontextu. Do vlastnosti value je nejprve uložen celý obsah stavové

komponenty pomocí třítečkového spread operátoru, následně proměnné z předešlé části metody render a poté funkce reprezentující události:

- onChangeFilterInput - změna obsahu vstupu od uživatele
- onAddTag - zakliknutí tlačítka tagu a jeho přidání do proměnné selectedTagIds
- onRemoveTag - při zakliknutí zakliknutého tlačítka - odstranění z proměnné selectedTagIds
- deleteAll - smazání obsahu proměnné selectedTagIds - nahrazení prázdným stringem
- onCheckUniqByName - změna stavu booleanu ovládajícího checkboxy
- onChangeYearFrom - změna stavu slideru yearFrom
- onChangeYearTo - změna stavu slideru yearTo
- changeServer - změna stavu booleanu server

Uvnitř značek <VideoContext> se nachází krátký kód - *this.props.children* - který umožňuje veškeré atributy značky *VideoContext.Provider* vyrenderovat. Na konci je celá třída *VideoLogic* exportována výchozím způsobem.

2.2.6 Struktura souboru View.tsx

Tento soubor slouží zejména pro nastavení vzhledu aplikace. Pomocí styled-components jsou nastaveny styly jednotlivých komponent:

- LoadingWrapper - „obal“ pro nahrávací kolečko (spinner), nastavující jeho pozici na střed webové aplikace
- VideoDetailWrapper - „obal“ pro informace o videu zobrazené pod ním
- CenterSpinner - nastavení velikosti nahrávacího kolečka
- FilterWrapper - nastavení střední části zobrazované části aplikace - fixní výška a odsazení od ostatních částí

Ve stejnojmenné funkci souboru View se nachází proměnná *videoLogicData*, která slouží k získání dat z kontextu *VideoContext*, na který se „zaháčkuje“ pomocí hooku *useContext*. Pomocí něj má konstanta *loading* přístup ke stavu loading z kontextu. Dále se zde nachází implementace stylování nahrávacího kolečka na základě booleanu *loading*. V návratové hodnotě této funkce je použit React fragment (značky <></>), kterým je možné obejít omezení Reactu, kdy jedna komponenta musí vracet pouze jeden element. Uvnitř fragmentu se nachází struktura zobrazované části aplikace:

- Navbar - jednoduchá komponenta nastavující záhlaví aplikace
- VideoDetailWrapper - první část aplikace zobrazující přehrávač videa, seznam jeho tagů, podobná videa a informace o videu. Obashuje komponentu *VideoPreview*

- FilterWrapper - druhá část aplikace sloužící pro filtraci a zobrazení vyfiltrovaných výsledků vyhledávání. Značky <Row> a <Col> upravují rozložení komponent v aplikaci. Obsahuje komponenty *VideoList*, *SelectTags* a *PropsExistCheckboxes*
- FilteredStats a TagsStats - třetí část aplikace s grafickou reprezentací databáze a výsledků vyhledávání

Obr. 2.6: Struktura stránky podle komponenty View

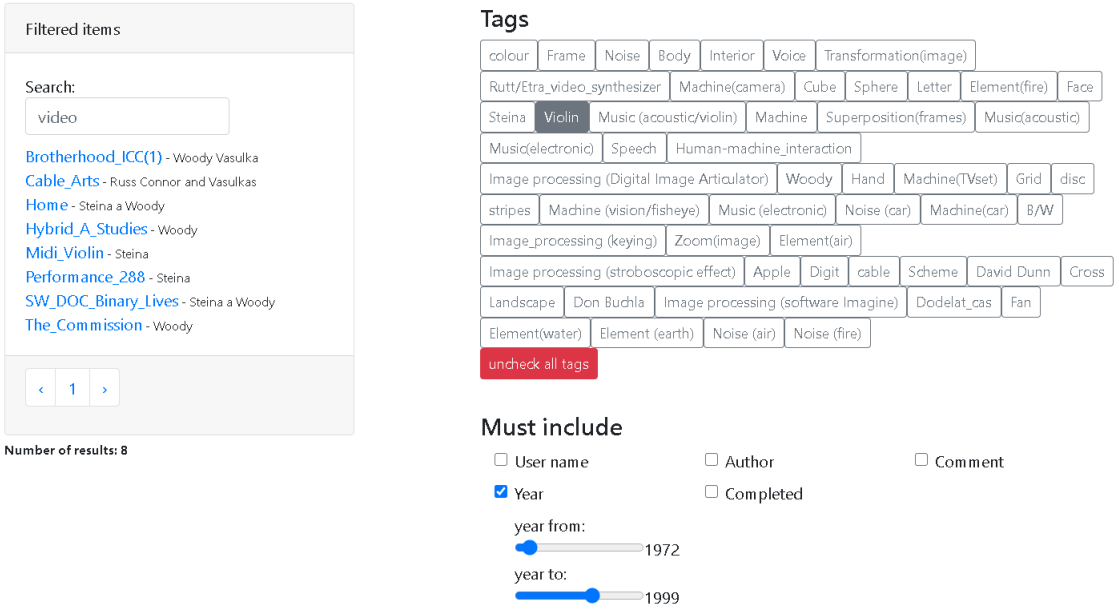
2.2.7 Komponenty

Pro přehlednost a optimalizaci je vhodné rozdělit větší soubory do vícero souborů tak, aby s nimi bylo možné odděleně manipulovat. Front-end tohoto projektu se skládá z několika komponent, které budou v následujícím textu popsány. Na začátku každého komponentového souboru se nachází seznam importů a na konci je stejnojmenná funkce souboru exportována. Každá komponenta typicky získává alespoň nějaká data z kontextu.

Tlačítka pro videa - SelectTags

V souboru se nachází jedno nastavení styled-components, a to pro "Wrapper„ - obal. V něm je pouze nastaveno odsazení zespodu. Funkce *SelectTags* umožňuje vybraným funkcím (*onAddTag, onRemoveTag, deleteAll*) a proměnným (*allUniqTags, selectedTagIds*) pomocí hooku *useContext* využít kontext *VideoContext*. Návratovou hodnotou je *Wrapper* obsahující text *Tags* a dále seznam tagů. Každý jednotlivý unikátní tag z proměnné *allUniqTags* je po namapování zobrazen jako tlačítko.

Tato tlačítka se zobrazují jako menší (*size = 'sm'*) a mají šedou barvu (v Bootstrapu standardně *color = 'secondary'*). Podle toho, jestli jsou tlačítka zakliklá se buď zobrazuje barva jen na okrajích tlačítka, nebo jsou zabarvená celá. Při kliknutí na tlačítko se také tag podle svého ID vloží do proměnné *selectedTagIds* reprezentující aktivní filtry. Tlačítko „uncheck all tags“ se zobrazí, pokud je právě proměnná *selectedTagIds* větší než 0, a umožňuje vymazat celý její obsah a s ním i aktivní filtrování pomocí tagů.



Obr. 2.7: Výsledky vyhledávání a filtry

Posuvníky - YearRange

Slider, neboli posuvník, je grafický element, pomocí kterého lze nastavit určitou hodnotu pohybem indikátoru v určitém rozmezí. 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 yearFrom (rok vytvoření od) a yearTo (rok vytvoření do). Dále je možné nastavit nejmenší možný krok, který pohyb slideru způsobí, což je v tomto případě jeden rok.

Soubor obsahuje jednu funkci, která nejprve získá pro proměnné *yearFrom, videos, yearTo* a funkce *onChangeYearFrom, onChangeYearTo* data z kontextu *VideoContext*. Konstanty *minYear* a *maxYear* hledají minimální a maximální hodnotu roku z pole čísel *videoYears*. Těmito daty jsou poté naplněny potřebné parametry vstupů. Vzhledem k tomu, že *e.target.value* vrací řetězec, je potřeba jej převést na číslo metodou *parseInt*.

Zatrhávací pole - PropsExistCheckboxes

Checkboxy (zatrhávací pole) jsou typ vstupu, kterým lze nastavit booleanovou hodnotu. Pod importy se nachází komponenty styled-components - *Wrapper*, zajišťující vnější odsazení zespodu a *CheckboxRow*, zajišťující vnější odsazení zleva.

Funkce zaškrtávacích polí je filtrování podle existence dané vlastnosti pro video. Vyfiltrována jsou ta videa, která nejsou naplněna žádnou hodnotou ve sloupcích *NameFromUser, Author, Comment, Year* a *Completed* (filtruje i nulu). Jejich chování je popsáno v souboru VideoLogic.tsx. Ve vlastní řadě se nachází nadpis ve značkách <h4> - „Must include“. Proměnné *existNameFromUser, existAuthor, existComment, existCompleted, existYear* a *onCheckUniqByName* jsou naplněny daty z kontextu. Každý jednotlivý checkbox je ve svém vlastsním sloupci <Col> a má vlastní popis <Label>. Při zaškrtnutí checkboxu year se objeví slidery pro výběr z rozsahu.

Vyfiltrovaná videa - VideoList

Tato komponenta slouží k zobrazení výsledků vyhledávání a filtrování, k jejich rozložení do více stran, jejichž offset je měněn pomocí elementu Pagination.

Proměnné *filterNameValue, filteredVideos a funkce onChangeFilterInput, onVideoClick* jsou získány z kontextu. Proměnná page je pomocí hooku *useState* deklarována a inicializována s hodnotou 0, tato proměnná je měněna funkcí *setPage*. Hook *useEffect* je volán při každé změně délky pole *filteredVideos* - což při změně seznamu filtrů nastaví proměnné *page* hodnotu 0. Tím lze zabránit situaci, kdy by se při nenulovém počtu výsledků zobrazil prázdný seznam. Dvě funkce - *getNextPage a getPrevPage* - slouží k posouvání stránek na další a předchozí pozici. Funkce *getNextPage* nastavuje hodnotu aktuální stránky na menší z hodnot „aktuální strana + 1“ a „délka seznamu vyfiltrovaných videí/offset stránky, to celé zaokrouhlené dolů“. Funkce *getPrevPage* nastavuje hodnotu aktuální stránky na větší z hodnot 0 a „aktuální stránka - 1“. Návratová hodnota je zaobalena elementem React fragment.

Element Card slouží k zobrazení svého obsahu v "kartě", která je vizuálně oddělena od ostatních komponent. V záhlaví této karty se nachází nadpis „Filtered items“. V těle karty se poté nachází input pro uživatele. Input, neboli vstup, je ve výchozím stavu lišta, do které může uživatel psát text. Tato komponenta slouží v rámci bakalářské práce k výběru z množiny. Při zadání výrazu se v seznamu výsledků vyfiltrují všechny, které daný výraz neobsahují ve svém názvu či ve jméně autora.

Pomocí vlastnosti „placeholder“ lze nastavit výchozí zašedlý text, který se zobrazí předtím, než uživatel něco do vstupu vyplní. Atribut *onChange* mění stav proměnné *onChangeFilterInput*. Hodnota zadaná uživatelem se ukládá do vlastnosti value.

Pomocí kódu:

```
{[...filteredVideos]
 .splice(page * PAGE_OFFSET, PAGE_OFFSET)
 .map(video => (
    <div key={video.VideoID}>
   <a href={‘#‘}
   onClick={() => onVideoClick(video.VideoID)}>
          {video.Name}
       </a>
       <small>
   - {video.Author ?? ’anonymní’}
   </small>
  </div>
  ))}
```

se obsah pole zobrazených vyfiltrovaných videí získaný spread operátorem metodou splice posune o hodnotu *PAGE_OFFSET*, výsledek je namapován a pro každé video je vytvořen s odkazem na adresu videa, který lze spustit kliknutím na jeho jméno. Dále je menší velikostí písma (značka <small>) vypsáno jméno autora - pokud není uvedeno, je vypsáno „anonymní“. V zápatí karty se nachází výše zmíněný pagination, který zobrazuje aktuální stránku a pomocí *getPrevPage* a *getNextPage* se dostává na další stránky kliknutím. Pod kartou je zobrazen počet vyfiltrovaných videí.

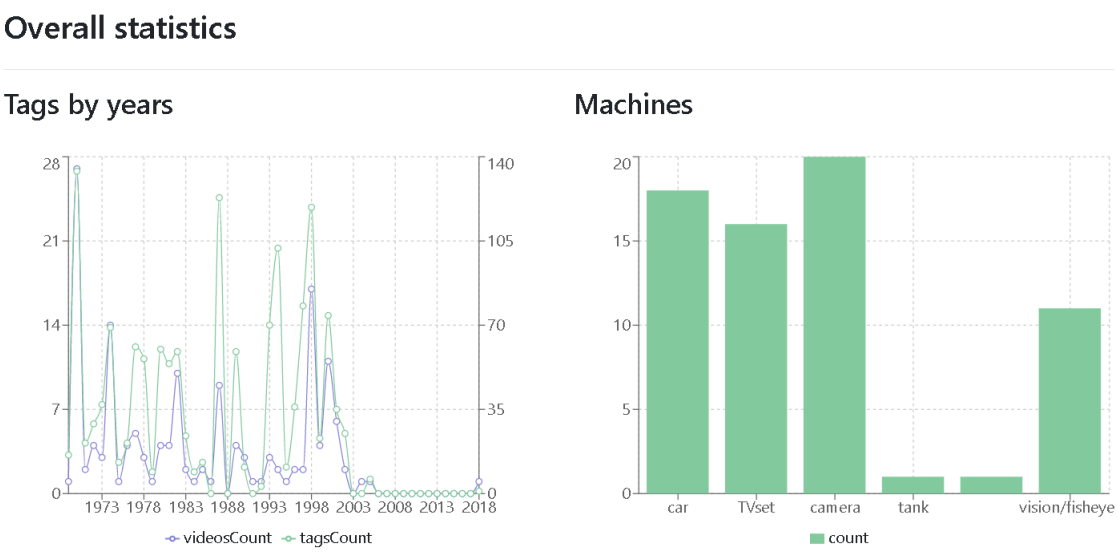
TagsStats - Statistiky databáze

Databázi nebo její části je možné také pomocí grafů knihovny recharts reprezentovat. V databázi se nachází několik tagů, které jsou vyjádřeny formátem kategorie(specifikace), kde se těchto specifikací nachází 2 - 9. Pokud se specifikace objevily pro kategorii jen dvě, byly vyhodnoceny jako nevyhovující pro grafickou reprezentaci, tudíž se v každém grafu z tagů nachází 4-9 položek. Tyto kategorie a specifikace byly manuálně zapsány.

ContainerWrapper je komponenta typu styled-components, která slouží k vnitřnímu odsazení celého úseku zeshora a pomocí background je nastavena barva pozadí. Pro optimální zobrazení grafů byl vytvořen element *ChartRow* - přestylovaný element bootstrapu - s odsazením zespodu, <Col> v těchto <Row> jsou zarovnány na střed. Následně je pro odsazení zespodu využito i přestylování nadpisu <h3>. Z pole manuálně naplněných řetězců lze získat obsah závorky dvojicí metod split, což je realizováno funkcí *getNameInBrackets*. Funkce *getTagsCount* získá na vstupu pole videí a string. Toto pole je namapováno a zbaveno dimenze funkcí *flatMap* a pomocí

anonymní funkce jsou pro každé video vyfiltrovány tagy nesplňující podmínku, že se rovnají zadanému stringu. Protože jde o počet položek, návratovou hodnotou je délka pole, získaná metodou *length*.

Proměnná videos je získána z kontextu, hookem *useState* je výchozí hodnota proměnné *windowWidth* nastavena na nulu a ovládána pomocí *setWindowWidth*. Hook *useEffect* nastavuje šířku *windowWidth* podle reálného smršťování a roztahování okna browseru. Tento údaj pomáhá k práci s grafy, jejichž zobrazování na mobilních zařízeních je potřeba manuálně nastavovat. Jednotlivé proměnné (kategorie)Count slouží k naplnění daných objektů a jejich klíčů (name, count) hodnotami. Pro graf vývoje tagů a videí podle letopočtů je potřeba naplnit tři klíče: *year, videosCount* a *tagsCount*. Následují grafy samotné, jež byly zasazeny do layoutu knihovny react-strap. Každý graf je naplněn správnými daty. V této čáti lze provést jejich stylizaci, zobrazení apod.

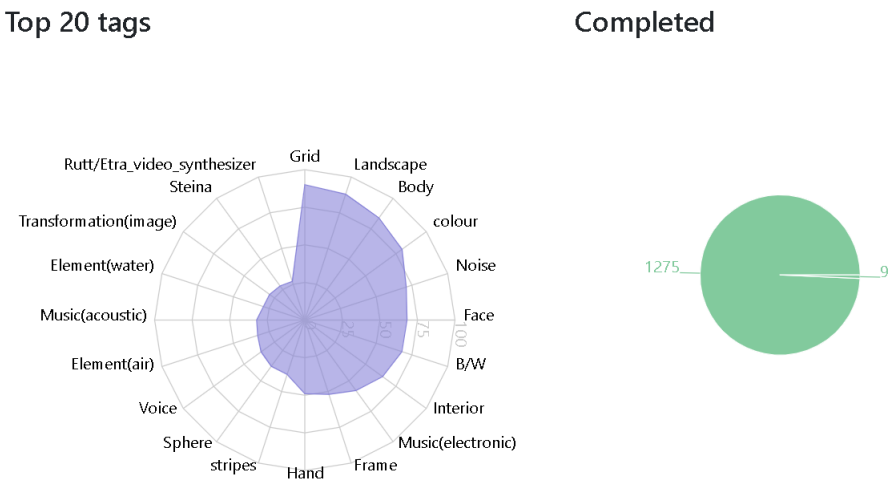


Obr. 2.8: Statistiky databáze

FilteredStats - Statistiky vyledávání

Vzhledem k tomu, že předchozí sekce vyobrazuje statistiky celé databáze bez ohledu na filtrované výsledky, bylo vzhledem k zadání potřeba ještě graficky reprezentovat i samotné vyhledávání. Obecně řečeno, funguje tato komponenta stejně jako *TagsStats*, změna je kromě pár stylovacích detailů v práci s polem videí *filteredVideos* místo všech videí v poli videí videos. Z kontextu je tedy potřeba zís-

kat i *filteredVideos*. Čtyři výsledné grafy zobrazují počet vyfiltrovaných tagů a videí v čase, počet videí od jednotlivých autorů, 20 nejčastěji se objevujících tagů a poměr videí, které byly již uzavřeny - jejich popisy byly už doplněny - ku nedokončeným.



Obr. 2.9: Statistiky vyhledávání

VideoPreview - Zobrazení videa a související komponenty

V této části je popsána celá první sekce aplikace. Rozdělená je na tři karty:

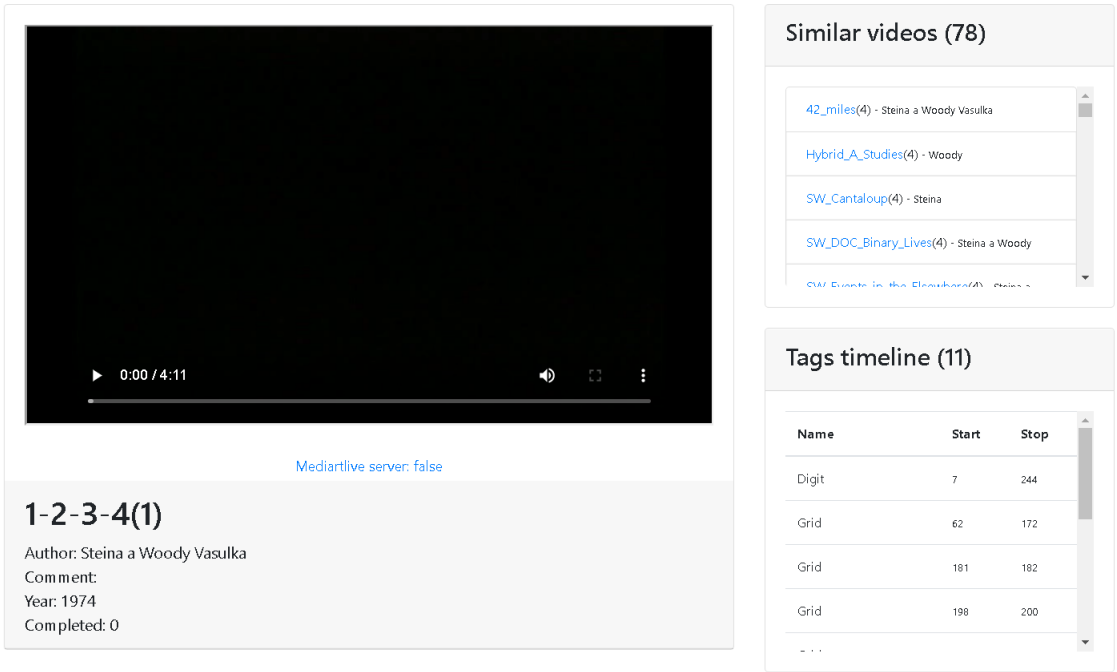
- Karta zobrazovaného videa a jeho popis
- Karta podobných videí
- Karta s tagy zobrazeného videa

Při použití standardního tagu <video> není možné videa v aplikaci spusit, neboť původní domény nemají povolenou tuto aplikaci v CORS. Toto lze obejít pomocí tagu <iframe>, který spouští stránku uvnitř stránky. Velikost videa je dána nastavením styled-components právě pro *iframe*. Videa se nacházejí na dvou serverech, kdy jeden je omezen připojením na VUT VPN a druhý přihlašujícími údaji. Webové aplikaci lze určit, z kterého serveru má data získávat pomocí tlačítka nacházejícího se pod videem. Kliknutí na tlačítko mění funkcí *changeServer* hodnotu booleanu server. Zdrojem pro iframe je návratová hodnota funkce *getUrl*, která vybírá URL adresu podle hodnoty booleanu server z pole dvou řetězců uložených do konstanty *url*. Odkaz na konkrétní zdroj se mění podle zrovna vybraného videa - jedná se o část odkazu:

```
videoPath=${video.PathId}
```

Pod tělem karty jsou zobrazeny informace o videu - jméno videa, autor, komentář, rok vytvoření a jestli jsou práce na videu hotové. Na druhé kartě je zobrazen seznam videí, které mají alespoň jeden společný tag a jsou seřazeny podle počtu těchto společných tagů. Seznam těchto videí byl vytvořen v komponentě VideoLogic.tsx a je získán společně s *onVideoClick, similarVideosByTags* a *selectedVideo* z kontextu. U nadpisu „Similar videos“ se napravo v závorkách zobrazuje počet videí s alespoň jedním společným tagem pomocí *similarVideosByTags.length* - délky pole.

Seznam podobných videí je vytvořen pomocí elementu <ListGroup> a je v něm nastavena výška a možnost scrollování nahoru a dolů (*overflowY: 'scroll'*). Názvy těchto videí jsou zároveň odkazy, které mění vybrané video pomocí *onVideoClick*. V závorce je za každým videem této karty počet společných tagů a také autor. Pokud autor neexistuje, vypíše se 'anonymní'. Karta zobrazující seznam tagů je zobrazuje jako tabulku pomocí značky <Table>, kde sloupce jsou název tagu, začátek jeho výskytu a konec (obojí uvedeno v sekundách), řádky jsou potom názvy tagů samotných.



Obr. 2.10: Sekce VideoPreview

2.2.8 Realizace ovládání aplikace pomocí Gamepadu

Implementace herního ovladače pro ovládání webové aplikace s knihovnou React neurčenou pro hry je poměrně problematická. Je to dáno samotným návrhem Gamepad API prohlížečů. Pro získávání dat o připojeném gamepadu je nutné využít metodu

sloužící k vytvoření nekonečné smyčky - *window.requestAnimationFrame*. Tato metoda slouží jako požadavek browseru pro provedení snímku animace, ve výchozím stavu v 60 snímcích za sekundu (přizpůsobuje se však rychlosti načítání stránky pro jednotlivé prohlížeče). Pro účely testování lze tento časový interval omezit metodou *setTimeout*.

Navázání této metody (a díky ní gamepadu samotného) na komponenty knihovny React pomocí stavu/kontextu vyvolává problém konstantního přerenderování stránky, což vede k výrazně nižší rychlosti aplikace a zpracování tohoto jevu je složité optimalizovat v tomto typu aplikace. Pro účely aplikace byla tedy funkcionalita gamepadu implementována mimo komponenty a nepracuje se stavem, což vede k nižší interaktivitě a omezení možností ovládání gamepadem.

Ovládání posunu pomocí joysticku

Jedna z možností omezené práce s webovou aplikací je pomocí scrollu - posouvání stránky. Metoda *tick* znázorňuje jednu periodu nekonečné smyčky, kdy veškeré proměnné, které se v této metodě nachází, jsou znovu naplněny, ve výchozím stavu šedesátkrát za sekundu. To umožňuje neustálé zjišťování informací ohledně gamepadu. Stav gamepadu tedy není ukládán, ale nepřetržitým záznamem změn je stav simulován.

Metodou *Navigator.getGamepads* je získáno pole objektů *Gamepad* nebo hodnot null. K získání aktivních gamepadů je tedy potřeba pole vyfiltrovat od položek s hodnotou null, což je realizováno při naplnění konstanty *activeGamepads*. Jednou z vlastností objektu *Gamepad* je *axes*, která konkrétně u joysticku zaznamenává hodnoty os X (horizontální) a Y(vertikální). Hodnoty těchto os se pohybují mezi -1 a 1. Pro jejich získání byla vytvořena funkce *gamepadAxis*. Vzhledem k tomu, že posouvání stránky je intuitivní ovládat pouze jedním joystickem a pouze na ose Y, byla vytvořena funkce *leftVerticalAxis* získávající data pouze o této hodnotě.

Pomocí metody *window.scroll* lze posouvat aktuální pozici na stránce o určitý počet pixelů. Pro osu X není pohyb žádoucí, ba v rámci této aplikace ani možný. *Metoda window.scrollY* umožňuje získávat data o aktuální pozici na ose Y. K této hodnotě je přičítána hodnota získaná z joysticku, vynásobená číslem 100, neboť při výchozích hodnotách je pohyb extrémně pomalý. Pokud gamepad není připojen, je výsledek metody *parseFloat*, sloužící k získání hodnoty osy Y levého joysticku při naplnění konstanty *leftVerticalAxis*, hodnota NaN indikující, že se nejedná o číslo. Z tohoto důvodu je metoda *window.scroll* volána pouze pokud *leftVerticalAxis* není NaN a zároveň je její hodnota buď větší než 0,1 nebo menší než - 0,1. Tato podmínka

ošetřuje nastavením minimálních hranic situaci, kdy joystickem není pohybováno, ale Gamepad API zaznamenává nepatrný pohyb (hodnota není přesně 0), tudíž by scrollování probíhalo samovolně.

Ovládání aplikace pomocí akcelerometru

Vzhledem k zjištění o nekompatibilitě gamepadu s návrhem aplikace, bylo po prostudování chování rozhraní Accelerometer Sensor API, založeného na podobném principu jako Gamepad API, přidávající aplikaci v rámci interaktivního ovládání minimální hodnotu z důvodů značných omezení, nebyla možnost ovládání aplikace akcelerometrem vytvořena.

Závěr

V rámci bakalářské práce byl pomocí knihovny React, systému Node.js a databáze MariaDB vypracován program sloužící k reprezentaci zadané databáze. Tento program disponuje funkčním uživatelským rozhraním s grafickou reprezentací výsledků a samostatným nastavením pro mobilní zobrazení. Front-end programu komunikuje s back-endem přes požadavky a odpovědi, které jsou zprostředkovány knihovnami Axios a Express. Back-end komunikuje s databázovým serverem pomocí objektově relačního mapování zprostředkovaného knihovnou Sequelize. Výsledky vyhledávání jsou zobrazeny formou seznamu, ve kterém je možné filtrovat výběrem z množiny, z rozsahu a podle binárního kritéria. Informace o vyfiltrovaných videích jsou reprezentovány pomocí grafů. Do přehrávače lze video umístit kliknutím na jeho název v senamu výsledků nebo v seznamu podobných videí. Pod videem v přehrávači jsou v popisku vypsány základní informace. Možnost ovládání interaktivními ovladači byla prozkoumána a ozkoušena, jejich funkce je však v rámci aplikace velmi omezená. Kromě úkolů stanovených zadáním byl v aplikaci vytvořen seznam podobných videí, grafy reprezentující celou databázi, tlačítko pro přepínání serverů a seznam klíčových slov objevujících se ve videu v daných časech. Pro příjemnější vzhled byla aplikace pomocí grafických knihoven nastylována.

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. 22.11.2019] 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] SKOUPIL, David. *Úvod do paradigmat programování*. [online] [cit. 06.06.2020] 2007. Dostupné z URL:
<https://phoenix.inf.upol.cz/esf/ucebni/uvod_para.pdf>.

[6] FOROUZAN, Behrouz. GILBERG, Richard. *Computer Science: An Object Oriented Approach Using C++ (1st Edition)* McGraw-Hill Higher Education, 2019. ISBN 978-0073523385

[7] KMETIUK, Anatolii. *Mastering Functional Programming: Functional Techniques for Sequential and Parallel Programming with Scala* Birmingham, UK: Packt Publishing, 2018. ISBN 9781788620796

[8] Oracle *The Java Tutorials* [online] [cit. 05.06.2020]. 2019, Dostupné z URL: <https://docs.oracle.com/javase/tutorial/java/concepts/object.html>.

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

[10] DataFlair *Features of JavaScript - 13 Vital JavaScript Features You Must Learn!* [online]. [cit. 05.06.2020]. 2019, Dostupné z URL: <https://data-flair.training/blogs/features-of-javascript/>.

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

[12] W3Schools Online Web Tutorials *JavaScript Versions* [online] [cit. 05.06.2020] Dostupné z URL: <https://www.w3schools.com/js/js_versions.asp>.

[13] W3Schools Online Web Tutorials *What is JSON* [online] [cit. 05.06.2020] Dostupné z URL: <https://www.w3schools.com/whatis/whatis_json.asp>.

[14] MIKŠŮ, Vojtěch. *JavaScript? Babel.*. [online]. [cit. 22.11.2019] 2016. Dostupné z URL:
<https://www.dzejes.cz/babel.html>.

[15] DAJBYCH, Václav. *K čemu je dobrý TypeScript*. [online]. [cit. 22.11.2019] 2013. Dostupné z URL:
<https://www.zdrojak.cz/clanky/k-cemu-je-dobry-typescript/>.

[16] MACHAČ, Marek. *Automatická detekce potíží v JavaScriptu s ESLint*. [online]. [cit. 22.11.2019] 2015. Dostupné z URL:
<shorturl.at/fiqv8>.

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

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

[19] AVIANI, Goran. *HTTP and everything you need to know about it*. [online] [cit. 22.11.2019] 2018. Dostupné z URL:
<shorturl.at/ozSUV>.

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

[21] MALÝ, Martin. *REST: architektura pro webové API*. [online]. [cit. 22.11.2019] 2009. Dostupné z URL:
<https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>.

[22] 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.

[23] SALVET, Pavel. *Seriál o komunikaci mezi stránkou a serverem: Díl 1. AJAX v kostce*. [online] [cit. 22.11.2019] 2015. Dostupné z URL:
<https://www.interval.cz/clanky/ajax-v-kostce/>.

[24] SALVET, Pavel *Seriál o komunikaci mezi stránkou a serverem: 2.AJAX: CORS,polling*. [online] [cit. 22.11.2019] 2015. Dostupné z URL:
<https://www.interval.cz/clanky/ajax-cors-polling/>.

[25] DEPOLD Sascha. *Sequelize* [online] [cit. 05.06.2020]. 2014, Dostupné z URL: <https://sequelize.org/v5/index.html>.

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

[27] MIKŠŮ, Vojtěch. *React - JSX*. [online]. [cit. 22.11.2019] 2016. Dostupné z URL: <https://www.dzejes.cz/react-jsx.html>.

[28] Facebook Inc. *React – A JavaScript library for building user interfaces*[online]. [cit. 05.06.2020]. 2020, Dostupné z URL: <https://reactjs.org/docs/react-component.html>.

[29] HAMEDANI, Mosh. *JavaScript for React Developers | Mosh*. In: Youtube [online] [cit. 22.11.2019] 2018, Dostupné z URL: <https://www.youtube.com/watch?v=NCwa_xi0Uuc>.

[30] WIERUCH, Robin. *How to pass props to components in React* [online] [cit. 05.06.2020] 2018, Dostupné z URL: <https://www.robinwieruch.de/react-pass-props-to-component>.

[31] HEIS, Kevin. *Let's Build a Virtual DOM from Scratch*. In: Youtube [online] [cit. 22.11.2019] 2017, Dostupné z URL: <https://www.youtube.com/watch?v=l2Tu0NqH0qU>.

[32] BERNARDES, Marlon. *How to Use Axios as Your HTTP Client*. [online] [cit. 22.11.2019] 2015. Dostupné z URL: <http://codeheaven.io/how-to-use-axios-as-your-http-client/>.

[33] ČÁPKA, David. *Lekce 1 - Úvod do CSS frameworku Bootstrap*. [online] [cit. 18.12.2019] 2018. Dostupné z URL: <https://www.itnetwork.cz/html-css/bootstrap/kurz/uvod-do-css-frameworku-bootstrap>.

[34] BRUMM, Ben. *How to Handle a Many-to-Many Relationship in Database Design*. [online] [cit. 22.11.2019] 2017. Dostupné z URL: <shorturl.at/ftHMX>.

[35] IBM Corporation *Relational Databases Explained* [online]. [cit. 05.06.2020]. 2020, Dostupné z URL: <https://www.ibm.com/cloud/learn/relational-databases>.

[36] CONNOLY, Thomas M.; BEGG, Carolyn E. *Database Systems – A Practical Approach to Design Implementation and Management* Pearson (6th ed.), 2014. ISBN 978-1292061184.

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

[38] CAMDEN, Raymond. *Using the HTML5 Gamepad API to Add Controller Support to Browser Games*. [online] [cit. 07.12.2019] 2019. Dostupné z URL: <shorturl.at/cejl6>.

[39] Mozilla Developer Network. *Using the Gamepad API - Web APIs* [online]. [cit. 05.06.2020]. 2005, Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API>.

[40] BAR, Adam. *Device Motion* [online] [cit. 07.12.2019] Dostupné z URL: <https://whatwebcando.today/device-motion.html>.

[41] Mozilla Developer Network. *Sensor APIs - Web APIs* [online]. [cit. 05.06.2020]. 2005, Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs>.

Seznam symbolů, veličin a zkratek

AJAX	Asynchronní JavaScript a XML
API	Application Programming Interface - rozhraní pro programování aplikací
App	Aplikace
BE	Back-end
CORS	Cross-origin resource sharing - mehcanismus umoňující omezení přístupu z jedné domény na druho
CRUD	Create, Read, Update, Delete - základní datové operace
CSS	Cascading Style Sheets - jazyk pro popis stylizace webové stránky
Col	Column - sloupec
Const	Konstanta
Ctx	Kontext
DB	Databáze
dev	Developer - vývojář, může mít i význam „vývojářské“, „pro vývojáře“ atd.
DOM	Document Object Model - viz str. 14
DevOps	Development & Operations - spolupráce vývoje a provozu
div	Division - sekce
e	Event - událost
enum	Enumerace - výčet
ES	ECMAScript
FE	Front-end
Float	Číslo s plovoucí řádovou čárkou
h(číslo)	Header - nadpis
HTML	Hypertext Markup Language - jazyk pro tvorbu webových stránek
HTTP	Hypertext Transfer Protocol - protokol pro komunikaci mezi servery
ID	Identifikace
IDE	Integrated Development Environment - vývojové prostředí
Int	Integer - celé číslo (omezení rozsahu čísel se liší podle programovacího jazyka)
JS	JavaScript
JSON	JavaScript Object Notation - formát určený zejména pro přenos dat
JSX	syntaktické rozšíření JavaScriptu kombinující HTML a JavaScript
MVC	Model-View-Controller - softwarová architektura rozdělující aplikaci na tři bloky
NPM	Node Package Manager - správce javascriptových balíčků
NaN	Not a Number - hodnota reprezentující „nečíslo“

OOP	Objektově orientované programování - programovací paradigma založené na objektech
ORM	Objektově relační mapování - technika pro konverzi dat mezi OOP jazykem a relační databází
PHP	Hypertext Preprocessor - skriptovací programovací jazyk
Props	Properties - volně přeloženo jako „vlastnosti“
RDBMS	Relational Database Management System - viz str. 31
req	Request - požadavek
res	Response - odpověď
REST	Representational State Transfer - softwarový architektonický styl pro webové aplikace
src	Source code file - zdrojový kód
SQL	Structured Query Language - strukturovaný dotazovací jazyk
SVG	Scalable Vector Graphics - formát souboru pro škálovatelnou vektorovou grafiku
tag	Klíčové slovo
TCP/IP	Transmission Control Protocol/Internet Protocol - rodina protokolů pro komunikaci v počítačové síti
TS	TypeScript - rozšíření jazyka JavaScript, zejména o statické datové typy
TSX	JSX pro TypeScript
UI	User Interface - Uživatelské rozhraní
URL	Uniform Resource Locator - jednoznačné určení zdroje
XHTML	Extensible Hypertext Markup Language - značkovací jazyk vycházející z HTML
XML	Extensible Markup Language - značkovací jazyk

Seznam příloh

A Uživatelská příručka

Přiložené CD neobsahuje testovací databázi, neboť není veřejně přístupná, dostupná je pouze na požádání u vedoucího práce doc. Ing. Jiřího Schimmela, Ph.D.

Postup spuštění aplikace (pro OS Windows):

1) Stáhněte programy Node.js a MariaDB ze stránek výrobců, tyto programy nainstalujte

2) V editoru kódu otevřete složky Back-end a Front-end a stáhněte v každé z nich příkazem „npm i“ potřebné knihovny

3) V aplikaci MySQL Client (nainstalovaném zároveň s MariaDB) se přihlašte a zadejte příkaz: „CREATE DATABASE (jméno databáze)“

4) Pomocí příkazu mysql -u root -p (jméno databáze) < (dumpTestovacíDatabáze).sql naplňte databázi daty (je možné, že bude nejdříve potřeba přidat program „mysql.exe“ do proměnné prostředí „path“)

5) Ve složce Back-end doplňte přihlašovací údaje do souboru env-config.ts

6) V obou složkách aplikaci spusťte příkazem „npm run start“

B Obsah přiloženého CD

Médium obsahuje text bakalářské práce ve formátu pdf, složku zdrojový kód (bez testovací databáze) a kopii uživatelské příručky. Téměř všechny soubory nacházející se uvnitř složky „zdrojový kód“ jsou do podrobna popsány v rámci textu BP.

```
/ ..... kořenový adresář přiloženého CD
├── BP-Dominik-Kure.pdf
├── Uzivatelska-prirucka.txt
└── Back-end
    ├── .vscode
    │   └── settings.json
    ├── fixtures
    ├── src
    │   ├── database
    │   │   ├── models
    │   │   │   ├── GroupsModel.ts
    │   │   │   ├── TagGroupsModel.ts
    │   │   │   ├── TagsModel.ts
    │   │   │   ├── UsersModel.ts
    │   │   │   ├── VideosModel.ts
    │   │   │   └── VideosTagsModel.ts
    │   │   └── core.ts
    │   └── main.ts
    ├── .eslintrc.js
    ├── .gitignore
    ├── .prettierrc.js
    ├── env-config.ts
    ├── package.json
    ├── package-lock.json
    ├── readme.md
    └── tsconfig.json
```

/ ..... kořenový adresář přiloženého CD

```
└── Front-end
    ├── .vscode
    │   └── settings.json
    ├── public
    │   ├── favicon.ico
    │   ├── index.html
    │   ├── logo192.png
    │   ├── logo512.png
    │   ├── manifest.json
    │   └── robots.txt
    ├── src
    │   ├── Components.tsx
    │   │   ├── FilteredStats.tsx
    │   │   ├── PropExistCheckboxes.tsx
    │   │   ├── SelectTags.tsx
    │   │   ├── TagsStats.tsx
    │   │   ├── VideoList.tsx
    │   │   ├── VideoPreview.tsx
    │   │   └── YearRange.tsx
    │   ├── index.tsx
    │   ├── react-app-env.d.ts
    │   ├── Root.tsx
    │   ├── serviceWorker.ts
    │   ├── utils.ts
    │   ├── VideoLogic.tsx
    │   ├── videosTagsService.ts
    │   └── View.tsx
    ├── .eslintrc.js
    ├── .gitignore
    ├── .prettierrc.js
    ├── package..json
    ├── package-lock.json
    ├── README.md
    └── tsconfig.json
```