Discovery rozhraní pro seznam internetových zdrojů

Bc. Dušan Gavenda

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Dušan Gavenda**
Osobní číslo:	**A19437**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačové a komunikační systémy**
Forma studia:	**Kombinovaná**
Téma práce:	**Discovery rozhraní pro seznam elektronických zdrojů**
Téma práce anglicky:	**Discovery Interface for a List of Electronic Resources**

Zásady pro vypracování

1. Proveďte literární rešerši na dané téma.
2. Shromážděte požadavky na rozhraní.
3. Navrhněte technické řešení ve formě webové aplikace.
4. Zdůvodněte výběr jednotlivých komponentů technického řešení.
5. Realizujte a otestujte výsledné technického řešení ve spolupráci s uživatelem.
6. Věnujte pozornost zabezpečení aplikace.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. AngularJS. AngularJS [online]. 2020 [cit. 2020-11-11]. Dostupné z: https://angularjs.org/
2. DAYLEY, Brad. Node.js, MongoDB and AngularJS web development. Upper Saddle River: Addison Wesley, 2014. ISBN 978-032-1995-780.
3. ULUCA, Doguhan. Angular 6 for Enterprise-Ready Web Applications: Deliver production-ready and cloud-scale. Birmingham: Mumbai Packt, 2018. ISBN 978-178-6462-909.
4. SESHADRI, Shyam. Angular: Up and Running: Learning Angular, Step by Step. Sebastopol, CA: O'Reilly Media, 2018. ISBN 978-149-1999-837.
5. KARPOV, Valeri a Diego NETTO. Professional AngularJS. Indianapolis, IN: John Wiley & Sons, 2015. ISBN 978-111-8832-073.

Vedoucí diplomové práce: **doc. Ing. Jiří Vojtěšek, Ph.D.**
Ústav řízení procesů

Konzultant diplomové práce: **Ing. Ivan Masár**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

Ing. Miroslav Matýsek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 22.5.2021 Dušan Gavenda, v.r.

ABSTRAKT

Diplomová práce se věnuje vývoji webové aplikace pro seznam elektronických informačních zdrojů.

Teoretická část je zpracována v podobě průzkumu možností vývoje požadované aplikace a technologií, kterými se dá vývoje dosáhnout.

V praktické části jsou definovány požadavky pro vývoj, a implementace funkcionality jednotlivých částí webové aplikace.

Výsledkem práce je webová aplikace, zpřístupněná pro širokou veřejnost pro přístup k elektronickým informačním zdrojům, která je provozována Knihovnou UTB.

Klíčová slova: Webová aplikace, Single-page aplikace, Angular, Elektronické informační zdroje

ABSTRACT

This thesis deals with the development of a web application for a list of electronic information resources.

Theoretical part is processed as a research of development options of the required application and technologies which can be used to develop it.

In the practical part, there are definitions of requirements for the development, and implementations of the application functionality.

The result of this thesis is a web application that is made available for the general public to access electronic information resources, operated by the UTB Library.

Keywords: Web application, Single-page application, Angular, Electronic information resources

Rád bych poděkoval doc. Ing. Jiřímu Vojtěškovi, Ph.D. za odborné vedení mé práce a za cenné rady při konzultacích. Dále bych chtěl poděkovat Ing. Ivanu Masárovi a Mgr. Světlaně Hrabinové, Ph.D. za komunikaci a rady při zpracování praktické části mé diplomové práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Cílem diplomové práce je vytvoření webové aplikace seznamu elektronických informačních zdrojů pro Knihovnu UTB. Tato aplikace nahradí aktuálně používané řešení, které je z technických důvodů rozděleno do dvou separátních aplikací. Výsledkem bude webová aplikace využívající návrhového modelu single-page.

Teoretická část práce je složena z popisu návrhových modelů single-page a multi-page, jejich charakteristik, vhodnost jejich použití, a výhody a nevýhody jednotlivých modelů. Východiskem této kapitoly je zhodnocení různých případů, ve kterých by bylo vhodné použít jeden, či druhý model.

Další část teoretické práce je věnována průzkumu vhodných technologií pro technické zpracování diplomové práce. Mezi zvážené technologie patří Angular, React a VueJS. Každé z těchto technologií je věnováno několik stran. V tomto prostoru jsou nastíněny jejich funkční vlastnosti, nejdůležitější vývojové prvky a rozdíly, jimiž se liší od svých konkurentů. Na konec je provedeno srovnání všech tří množností a nastínění možného využití technologií z různých hledisek.

V další hlavní kapitole teoretické části jsou definovány elektronické informační zdroje, jejich specifikace a rozdělení. V této kapitole jsou vysvětleny i nejčastější typy elektronických informačních zdrojů a technologie, které je možné používat pro přístup k těmto zdrojům. Poslední kapitola je věnována možným problémům s bezpečností webových aplikací a způsobům k jejich mitigaci.

V praktické části diplomové práce jsou definovány požadavky, které webová aplikace musí splňovat a jsou předvedeny i aktuálně používané aplikace, které budou nahrazeny právě aplikací vyvíjenou jako součást této diplomové práce. Jsou definovány i elektronické informační zdroje, ke kterým poskytuje přístup Knihovna UTB.

Obsahu praktické části je věnován primárně návrhu a implementaci technického řešení. Jsou zde popsány nejdůležitější funkce aplikace a způsob jakým bylo dosaženo jejich zprovoznění.

Na závěr praktické části je také prostřednictvím fotografií poskytnut pohled na finální verzi aplikace a posouzení její bezpečnosti vůči bezpečnostním hrozbám definovaným v teoretické části práce.

I. TEORETICKÁ ČÁST

1 SINGLE-PAGE A MULTI-PAGE APLIKACE

World Wide Web (www) byl vyvinut již v roce 1989 a první funkční webová stránka byla vytvořena o rok později. Webové aplikace za sebou tedy mají už přes 30 let historie. Mnoho věcí se změnilo, ale základní stavební kameny zůstaly stejné. Většina webových stránek, kterou v prohlížeči uvidíme je HTML kód, který prohlížeč dokáže transformovat do výstupu pro uživatele. [1]

Po většinu tohoto času webovému světu dominoval model multi-page aplikací (MPA). V posledních letech však tento model začal navrhovat modernější model single-page aplikací (SPA).

1.1 Single-page aplikace

SPA je webová aplikace, jejíž architektura umožňuje přechody mezi jednotlivými stránkami uvnitř stejné aplikace bez nutnosti opětovného stahování celého Document Object Modelu (DOM). [2]

Aplikace navržená pomocí modelu SPA vždy všechny potřebné zdroje stáhne již při prvním navštívení stránky. Při přechodu mezi jednotlivými stránkami se již načítají pouze dynamická data (např. zprávy) nebo multimediální obsah (např. obrázky).

1.1.1 Výhody a nevýhody SPA

Každý druh aplikace má své výhody a nevýhody. V této sekci se budeme věnovat výhodám a nevýhodám SPA.

VÝHODY

Rychlost při interakci

SPA si všechno potřebné ze serveru stáhne již při prvním načtení aplikace. Při interakci následně již stahuje pouze dynamická data, jejichž velikost se obvykle pohybuje v rámci bytů a kilobytů. Jakákoliv interakce s aplikací tedy vyvolá datově nenáročnou operaci, která se z pohledu uživatele může jevit téměř instantně. [2]

Opětovná použitelnost

Renderování HTML až v prohlížeči přináší velkou řadu výhod. Jednou z nich je nezávislost na tom, co se děje na serveru. SPA ke své funkci potřebuje pouze výstupní data, která ze serveru přichází, už však nezáleží, jaké věci se dějí v pozadí.

Server tedy slouží pouze jako zdroj dat, která může odebírat kdokoliv, kdo k nim má přístup. Na základě jedné API lze postavit neomezené množství nezávislých aplikací (např. webovou a mobilní). [2]

Debugování (ladění)

Debugování SPA je v dnešní době jednoduché a přívětivé. Byť se můžou detaily lišit v jednotlivých prohlížečích, každý prohlížeč nabízí nějakou formu pro inspekci zdrojového kódu (např. Chrome Dev Tools pro Google Chrome a Firefox Developer Tools pro Firefox). Pomocí těchto integrovaných nástrojů lze jednoduše prohlížet HTML, CSS i JavaScript. Navíc je každému frameworku, pomocí nějž lze vyvíjet SPA, k dispozici nejrůznější množství doplňků pro prohlížeče, které mohou sloužit od prohlížení struktury aplikace až po přepisování interních dat. [2]

Cachování

SPA jsou účinnější při práci s lokální pamětí. Jeden dotaz na serveru načte všechny potřebné zdroje pro fungování aplikace – aplikace následně do jisté míry může fungovat i bez připojení k internetu. [3]

Menší zátěž na internet

Zdroje SPA se načítají pouze jednou – pokud uživatel tedy provádí přechody mezi stránkami, tak ve výsledku spotřebuje mnohem méně internetových dat než by tomu bylo u tradičních webových stránek. [4]

NEVÝHODY

Pomalé prvotní načtení

Stahování všech potřebných zdrojů při prvním načtení má i své nevýhody – SPA jsou obvykle tvořeny velkým množstvím JavaScriptu, který je třeba stáhnout, a proto může prvotní načtení trvat déle než u klasické MPA. [3]

Velká zátěž prohlížeče

O generování HTML se stará JavaScript v prohlížeči. To může způsobit velké nároky na paměť a výkon zařízení (také záleží na velikosti aplikace). Uživatelé s málo výkonnými zařízeními mohou mít problémy s rychlostí dané aplikace, navíc se tyto problémy mohou promítnout do rychlosti celého zařízení. [2]

Monitorování paměti

Stahování celé stránky při každém přechodu má jednu velkou výhodu – všechny alokované zdroje se mohou jednoduše uvolnit. SPA však může běžet bez jediného znovu-načtení několik hodin, nebo dní. Je tedy na zodpovědnosti každého vývojáře, aby sám dával pozor, jaká data u klienta v prohlížeči udržuje a zda se mu tam nehromadí. Špatný design může vést k alokaci veškeré paměti pouze pro data jediné aplikace. [2]

Závislost na JavaScriptu

JavaScript je hlavním článkem každé SPA. Pokud si vývojář nedá pozor, a nepřipraví záložní variantu pro nedostupnost JavaScriptu, uživateli se při snaze pro načtení aplikace nemusí zobrazit vůbec nic.

Složitá implementace pro SEO

Search Engine Optimalization (SEO) je důležitým faktorem pro úspěch webové aplikace. Vyhledávací služby (např. od Google nebo Bing) k vyhledávání používají obsah, které jim webové stránky vrátí po zavolání dotazu na server. SPA však ze serveru vrací pouze prázdnou šablonu, jejíž obsah se následně vyplní v prohlížeči.

Naštěstí i při použití SPA je do jisté míry možné využít server-side rendering (vygenerování HTML již na serveru) a pomoci tak při optimalizaci pro vyhledávání. Firmy jako Google navíc stále přichází s novými způsoby ulehčení života pro majitele a vývojáře SPA. [3]

1.2 Multi-page aplikace

MPA fungují tradičním způsobem. Každá navigace mezi stránkami vyvolá dotaz na server, který požadavek zpracuje a vrátí do prohlížeče obvykle již hotové HTML. Tyto aplikace jsou obvykle mnohem větší a mají komplikovanější strukturu než SPA. Z těchto důvodů jsou obvykle náročnější na vývoj a často u nich nelze spolehlivě rozdělit část frontendu a backendu. [5]

1.2.1 Výhody a nevýhody MPA

Pro posouzení vhodnosti použití tohoto modelu je třeba definovat jeho výhody a nevýhody.

VÝHODY

Lepší SEO optimalizace

V MPA modelu se každá záložka v aplikaci chová jako samostatná webová stránka. Díky tomu je poměrně jednoduché každou stránku nastavit zvlášť a usnadnit tak vyhledávacím systémům jednodušší práci s vyhledáváním. [6]

Jednoduchá škálovatelnost

Jak už bylo řečeno výše – každá záložka v aplikaci se chová jako samostatná stránka. Přidání dalších záložek nijak neovlivní funkcionalitu zbytku aplikace a je tedy mnohem jednodušší aplikaci rozšiřovat. [6]

Schopnosti analýzy

Analytické webové nástroje jako Google Analytics se dají do MPA jednoduše implementovat. Díky těmto nástrojům může firma, která webovou aplikaci provozuje, jednoduše sledovat, jak se aplikaci daří a případně ji dále vylepšovat na základě daných analýz. [6]

NEVÝHODY

Možné problémy s výkonností

V případě velkého množství dotazů na server nevyhnutelně dojde k degradaci rychlosti a výkonu aplikace. S větším provozem u webové aplikace dochází ke generování mnohem většího náporu jak na internet, tak na výkon serveru. [6]

Propojení frontendu a backendu

Serverová a klientská část aplikace je v modelu MPA obvykle poměrně úzce propojena. Vývoj a testování jednotlivých komponentů aplikace se v tomto případě stává komplikovanějším kvůli nutnosti funkcionality obou částí aplikace. Zároveň to vede k nutnosti opětovného vývoje backendu pro ostatní druhy aplikací (např. desktopové aplikace). [6]

1.3 Vhodnost použití SPA a MPA

Jak vyplývá z předchozích kapitol, SPA i MPA mají různé specifikace a každý návrhový model se tak může hodit do jiných situací.

1.3.1 Rychlost

V tomto ohledu nemá SPA konkurenci. Rychlost načítání a interaktivita dávají těmto aplikacím výhodu, kterou tradiční webové stránky budou jen těžce dohánět.

1.3.2 Uživatelský zážitek

Uživatelský pohled je nejdůležitější částí vývoje. Pokud se uživatelům nebude aplikace dobře používat, je větší pravděpodobnost, že přejdou ke konkurenci.

SPA jsou obvykle přístupnější k mobilním zařízením, které pomalu ale jistě přebírají majoritní kontrolu nad používáním internetu. Frameworky, na kterých jsou tyto aplikace stavěné, navíc obvykle nabízejí příležitost vývoje jak nativních, tak i hybridních mobilních aplikací. Pokud se tedy očekává většinová uživatelská základna ze strany mobilních uživatelů, určitě se vyplatí jít cestou SPA. [4]

MPA však mají v tomto ohledu také své výhody. Obvykle se mohou pyšnit lepší navigační strukturou a jsou tedy vhodnější v případě, kdy se od uživatele očekává spíše pasivní přístup (např. web se zprávami ze světa – od uživatele se neočekává žádný vstup, pouze sledování již vytvořeného obsahu).

1.3.3 Vývojový proces

Vývoj SPA je ve spoustě směrů jednodušší než u MPA. Použitelnost backendového kódu pro více než jednu aplikaci (např. webovou a mobilní) dokáže uspořit velké množství vývojového času. [4]

Oddělení frontendu a backendu navíc pomáhá zrychlit rychlost vývoje díky možnosti, že se mohou vyvíjet obě části zvlášť.

Bohužel i toto přináší své problémy. Práce na SPA obvykle vyžaduje práci s větším množstvím technologií. Firma, která se rozhodne pro tento typ aplikace, musí přijmout vývojáře s větším rozměrem jejich schopností nebo přijmout více specializovaných vývojářů pro různé části aplikace – a to se může prodražit.

1.3.4 Škálovatelnost

V tomto ohledu vyhrávají MPA. U tradičních webů není problém přidávat stále více a více obsahu – každá záložka je webovou stránkou sama o sobě. U SPA je třeba myslet na integraci s dalšími částmi aplikace, což může někdy vést k náročnému přepisování. [7]

2 JAVASCRIPTOVÉ FRAMEWORKY A KNIHOVNY

Vývoj webových aplikací se v posledních letech značně posunul právě díky SPA modelu návrhu aplikací. Realizaci této architektury by však vývojář těžce dosahoval pouze za použití klasických nástrojů.

Pro vývoj SPA jsou uzpůsobené JavaScriptové frameworky a knihovny, které vývoj značně usnadňují. JavaScript je vysokoúrovňový programovací jazyk, který dokáží zpracovat webové prohlížeče. Jeho primární využití je právě ve webových aplikacích, kde slouží k přidání interaktivních elementů. Framework je abstrakční vrstvou programovacího jazyka, která poskytuje standardizovaný postup vývoje a funkcionalitu usnadňující práci vývojářům. Primárním cílem je vždy redukce času na programování generických věcí, které již byly mnohokrát vytvořeny a přesunout se k programování samotné funkcionality aplikace. [8]

Z dotazníku na webu StackOverflow pro rok 2020 vyplývá, že v současné době na pomyslných stupních vítězů stojí trojice frameworků/knihoven. Nazývají se Angular, React a VueJS. [9]

2.1 Angular

Angular je open-source framework pro webové aplikace založený na TypeScriptu. Jeho vývoj je vedený týmem z Googlu a komunitou jednotlivců a jiných korporací. [10]

2.1.1 Historie

Angular je výraz používaný pro tento framework od verze 2. Verze 1 nesla název AngularJS a byla napsána v JavaScriptu. Verze 2 byla kompletně přepsána do nového jazyka – TypeScriptu.

AngularJS

Začátky Angularu se datují do roku 2009. Miško Hevery a Adam Abrons pracovali na vedlejším projektu, který by nabídnul online úložnou službu pro JSON a také klientskou knihovnu pro tvorbu webových aplikací, která by na tom závisela. Tento projekt zveřejnili a jako doménu použili *GetAngular.com*. [11]

Ve stejné době již Hevery pracoval pro Google. Spolu se dvěma dalšími kolegy byl přiřazen na Google Feedback projekt. Společně napsali dohromady více než 17000 řádků kódu během 6 měsíců. Hevery byl frustrovaný z nepřehlednosti kódu. Nabídl svému vedoucímu, že sám dokáže přepsat celou aplikaci během dvou týdnů s pomocí jeho vlastního frameworku.

Vedoucí mu tuto akci povolil a Hevery zvládnul přepsat celou aplikaci během tří týdnů a za použití pouze 1500 řádků kódu. Tento výsledek si vysloužil velký zájem uvnitř firmy a frameworku bylo přiřazeno jméno AngularJS. [11]

První oficiální stabilní verze AngularJS (0.9.0) byla vydána na GitHubu v říjnu 2010 pod MIT licencí. [11]

Důvody pro velký úspěch frameworku se dají shrnout do několika bodů:

- **Dependency Injection (DI)** – AngularJS byl prvním klientským frameworkem, který implementoval tuto vývojářskou architekturu. Jednalo se o obrovskou výhodu oproti soupeřícím technologiím (např. jQuery), které byly založeny na manipulaci DOM. Díky DI mohli vývojáři psát volně vázané a jednoduše testovatelné komponenty. Práce s vytvářením, zajišťování závislostí a předávání je do ostatních komponentů již byla přenechána na samotném frameworku. [11]
- **Direktivy** – Jedná se o popisky na specifických elementech, atributech, stylech atd. Direktivy jsou mocným nástrojem pro vytvoření vlastních, znovupoužitelných prvků, podobných HTML nebo atributů, které specifikují datovou vazbu HTML prvků s JavaScriptem. [11]
- **Oboucestná datová vazba (Two-way data binding)** – Tato technika umožňuje automatickou synchonizaci dat mezi modelem (proměnné v JavaScriptu, které uchovávají stav aplikace) a zobrazením. Pokud se změní data v modelu (např. výpočtem), tak se automaticky změní ta část HTML, která zobrazovala stav tohoto modelu. Stejná funkcionalita funguje i naopak. Například, pokud uživatel změní hodnotu ve formuláři, v JavaScriptu, který je na tento formulář navázán, se změna hned promítne a umožní, tak dokonalou synchronizaci dat se zobrazením v reálném čase. [11]
- **Single-Page Approach –** AngularJS byl prvním frameworkem, který úplně odstanil potřebu pro znovunačítání stránky. Nastínil tak cestu pro další frameworky (např. React a VueJS), které začly s vývojem později a začaly také používat SPA systém. [11]

Angular 2-11

Nová verze AngularJS byla vydána v září 2016 pod jménem Angular 2. Z původního kódu AngularJS nezůstalo vůbec nic, jednalo se o kompletní přepsání celého frameworku. [11]

Změny na úrovni architektury, zacházení s HTTP, životního cyklu aplikace a správy stavu aplikace byly tak velké, že bylo prakticky nemožné migrovat svůj projekt ze starého frameworku do nového. Nejednalo se tedy pouze o novou verzi frameworku, ale o kompletně nový, který s tím předchozím pouze nese podobný název. [11]

Rozhodnutí neudělat Angular 2 kompatibilní s AngularJS ukázalo záměr vývojářského týmu – přejít na kompletně nový přístup (nejen k syntaxi kódu, ale i v návrhu celé aplikace). Nový Angular byl velmi modulární, založený na komponentové struktuře, s novým a vylepšeným DI a mnohými programovacími systémy, které v původním frameworku vůbec nebyly. [11]

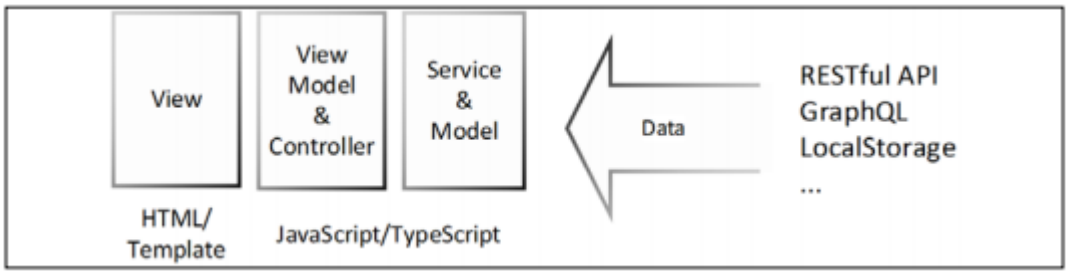
Novinky, které přišly s Angular 2:

- **Sémantické verzování** – Angular 2 byl prvním vydáním, který používal sémantické verzování – univerzální způsob verzování jednotlivých vydaní. Sémantické verzování je založeno na třech číslech – *X.Y.Z*. *X* je číslo znázorňující hlavní verzi, *Y* stojí za vedlejší verzi a *Z* znázorňuje verzi opravy. Změna hlavní verze probíhá pouze, pokud dojde ke změnám ve stabilních API. Vedlejší verze se inkrementuje, pokud dojde k přidání nových, zpětně kompatibilních systémů. Verze *Z* se změní v případě zpětně kompatibilní opravy chyb. [11]
- **TypeScript** – TypeScript je rozšíření JavaScriptu od Microsoftu. Přidává do JavaScriptu typy a objektově orientované vlastnosti (např. třídy, deklarace typů). TypeScript je následně převeden na JavaScriptový kód, se kterým dokáží pracovat prohlížeče. [11]
- **Server-Side Rendering –** Byl implementován Angular Universal – open-source technologie, která umožňuje serverům spustit angularové aplikace a klientovi předávat pouze vygenerované HTML soubory. [11]
- **Angular Mobile Toolkit –** Sada nástrojů pro vytváření mobilních aplikací. [11]
- **Command-Line Interface (CLI) –** Umožňuje vývojářům generovat komponenty, routy, servicy, a pipy pomocí terminálu. [11]
- **Komponenty –** Nahrazují *controller* a *scope* z AngularJS a pokrývají i většinu úkolů, o které se dříve staraly direktivy. [11]

V přibližně půlročních rozestupech jsou vydávány nové verze Angularu až do současnosti. Nejnovější verzí je Angular 11, který byl vydán v listopadu 2020.

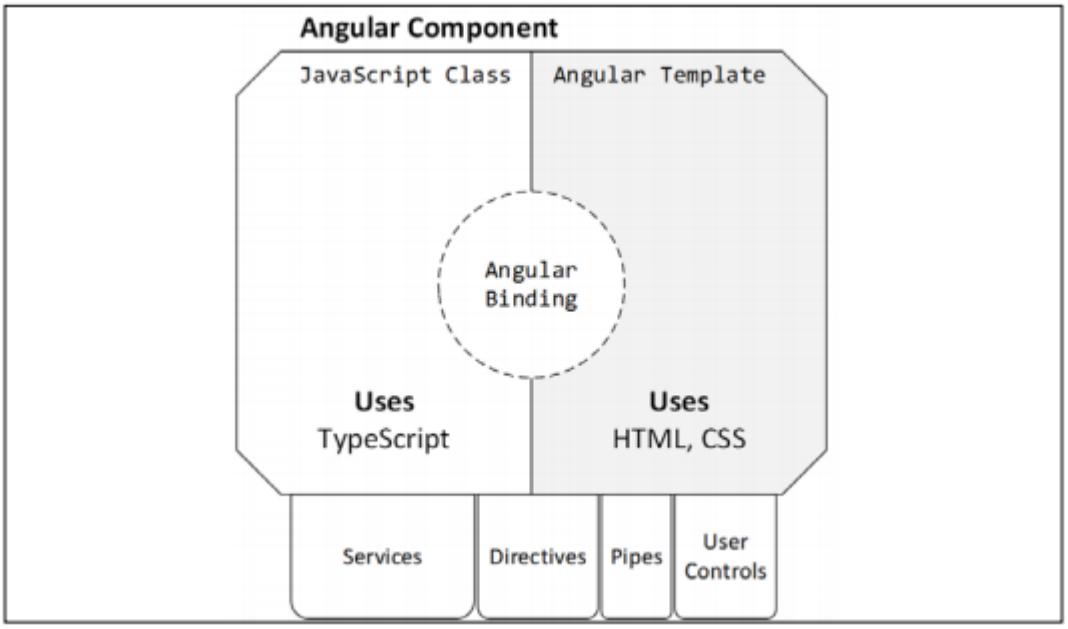
2.1.2 Základní architektura

Angular používá MV* model. MV* je hybridním modelem mezi Model-View-Controller (MVC) a Model-View-ViewModel (MVVM). Z vnějšího pohledu je architektura všech těchto modelů relativně podobná. [12]



Obr. 1 MV* architektura [12]

Novým konceptem je ViewModel, který reprezentuje spojení *view* (zobrazení) s modelem nebo servicem. V Angularu se toto spojení nazývá *binding*.



Obr. 2 Anatomie komponentu [12]

Třídy jsou konstrukce objektově orientovaného programování (OOP). Díky tomuto systému lze v Angularu používat *dependency injection* (DI). DI vývojáři poskytuje příležitost skládat komponenty dohromady z několika nezávislých částí, které jsou na základě aktuálních potřeb instancovány a vloženy do komponentu. Vývojář si tedy nemusí lámat hlavu s instancováním, inicializováním, a následným odstraňováním komponentů. [12]

Podobným způsobem lze používat stávající kód pomocí direktiv, *pipes* a dalších komponentů. [12]

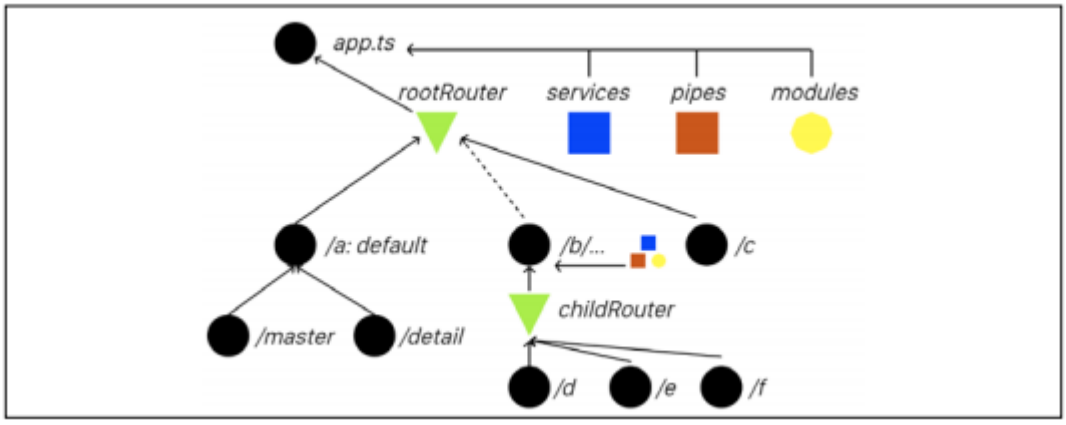
Všechny komponenty, servicy, direktivy a pipy jsou organizovány uvnitř modulů. Každá angularová aplikace je zabalena do kořenového modulu, který vykreslí první komponent, vloží do něj všechny servicy a připraví potřebné závislosti. [12]

2.1.3 Angular Router

Angular Router je balíček, který je kritickou součástí vytváření SPA s Angularem. [12]

Router přiřadí URL cesty k pohledům místo celým stránkám. Při změně URL pak provádí navigaci podle stanovených cest a zabraňuje klasické navigaci mezi stránkami, o kterou se stará prohlížeč. [14], [13]

Jednou z důležitých vlastností je podpora *lazy loadingu* (líné načítání). V diagramu na Obr. 3 je zobrazená aplikace, kde soubor *app.ts* obsahuje hlavní modul aplikace. Součástí tohoto modulu je router jménem *rootRouter* s deklarovanými cestami ke komponentům *a*, *master*, *detail* a *c*. Dále jsou součástí i *servicy*, pipy a moduly. Všechny tyto součásti budou při překladu zpracovány do jednoho souboru, který se stáhne při prvním načtení aplikace, pokud uživatel zvolí jednu z cest definovaných v *rootRouter*. [12]



Obr. 3 Diagram angularové aplikace [12]

Komponent *b* lze s pomocí *lazy loadingu* se svým vlastním routerem *childRouter* a komponenty *d*, *e*, *f* implementovat pomocí vytvoření zvláštního modulu, ve kterém budou deklarovány všechny tyto komponenty. Při překladu se vše obsažené v tomto modulu zpracuje do dalšího souboru, který se stáhne, až uživatel přejde na jednu z cest definovaných v *childRouter*. [12]

2.1.4 Funkcionalita

Aplikace v Angularu nabízí několik různých funkcionalit, bez jejichž využití se při vývoji neobejdeme.

Komponenty

Komponenty jsou základním stavebním kamenem Angularu. Každý komponent se skládá z:

- HTML šablona, která deklaruje to, co se bude vykreslovat [13]
- TypeScriptová třída, která definuje chování komponentu [13]
- CSS selektor definující používání komponentu v šabloně [13]
- CSS styly upravující zobrazení HTML [13]

Příklad jednoduchého komponentu:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-hello-world',
  template: '<h1>Hello World!</h1>',
  styles: ['h1 { font-weight: normal; }'],
})
export class HelloWorldComponent {}
```

Kód 1 Příklad nejjednoduššího komponentu

V příkladu v kódu 1 jsou uvedeny všechny prvky tvořící komponent. Obvykle se však šablona (HTML) a styly (CSS) vkládají do souboru zvlášť. Stejný komponent by tímto zápisem mohl vypadat takto:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css'],
})
export class HelloWorldComponent {}
```

Kód 2 Příklad komponentu pomocí standartního zápisu

V souborech specifikovaných pomocí *templateUrl* a *styleUrls* v kódu 2 by pak byly obsaženy prvky HTML a CSS z příkladu v kódu 1.

Dependency Injection

V doslovném překladu se jedná o „vložení závislostí“. DI je typ designu, kdy si třída vyžádá závislosti z externích zdrojů, než aby si je sama vytvořila. [13]

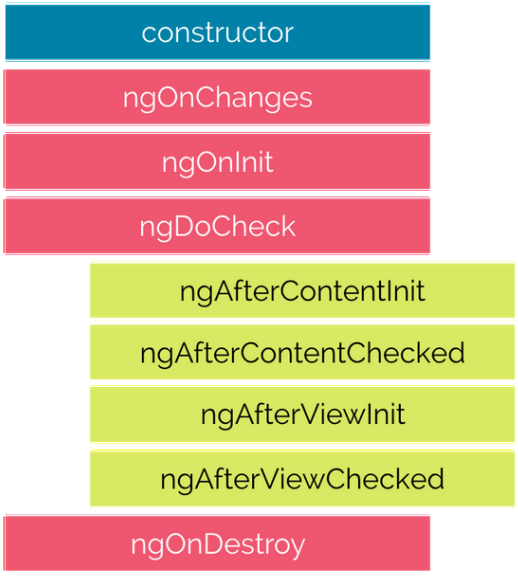
V Angularu se ke vkládání používají *servicy*. *Service* je klasickou JavaScriptovou třídou a jediný rozdíl oproti ostatním třídám (např. komponentům) je v dekorátoru, kterým se tato třída označí. Pro označení takové třídy se používá dekorátor *@Injectable()*. [13]

Component Lifecycle

Instance komponentu má životní cyklus začínající, když Angular vytvoří instanci třídy komponentu a vykreslí HTML komponentu společně s jeho *child* komponenty. Následuje detekování změn v poskytnutých datech a update třídy i pohledu (pokud jsou detekovány změny). Životní cyklus končí se zničením instance komponentu a odstranění jeho šablony z DOM. [13]

Lifecycle Hooks

Životní cyklus prochází několika různými fázemi. Vývojář má možnost exekuci každé této fáze odchytit a spustit vlastní kód vždy, když daná fáze proběhne. K tomu je třeba ve třídě komponentu specifikovat určité metody. [14]



Obr. 4 Lifecycle Hooks [14]

Tyto fáze (viz Obr. 4) jsou rozděleny na ty, které se týkají samotného komponentu (označeny růžově) a jeho *child* komponentů (označeny žlutě). [14]

Komponent má přístup k těmto *lifecycle hooks*:

- **constructor** – konstruktor je volán vždy, když Angular vytvoří novou instanci komponentu. [14]
- **ngOnChanges** – voláno při každé změně vstupních parametrů. [14]
- **ngOnInit** – jak už název napovídá – proběhne vždy po inicializaci komponentu (a po prvním *ngOnChanges*). [14]
- **ngDoCheck** – tento *hook* je volán po každém volání *change detectoru*. Jedná se o každou změnu hodnot v instanci komponentu. [14]
- **ngOnDestroy** – tato metoda bude volána těsně před zničením komponentu. [14]

Child komponenty mají přístup k následujícím *lifecycle hooks*:

- **ngAfterContentInit** – vyvoláno po projekci obsahu z *child* komponentu do *parent* komponentu. [14]
- **ngAfterContentChecked** – zavolá se po každé kontrole obsahu komponentu vyvolané *change detection* mechanismem. [14]
- **ngAfterViewInit** – voláno po úplné inicializaci komponentu. [14]
- **ngAfterViewChecked** – volá se po každé kontrole pohledu (*view*) vyvolané *change detection* mechanismem. [14]

Direktivy

Direktivy jsou třídy, které přidávají dodatečné chování elementům v aplikaci. [13]

Existuje několik druhů:

- **Komponenty** – komponent (viz. 2.3.1) je direktiva se šablonou. [13]
- **Attribute directives (atributové direktivy)** – jedná se o direktivy měnící chování nebo vzhled elementu, komponentu nebo jiné direktivy. [13]
- **Structural directives (strukturální direktivy)** – direktivy, které mění rozmístění prvků v DOM. [13]

Attribute directives

Tyto typy direktiv sledují změny ve stavu aplikace a mohou na základě těchto změn upravovat chování HTML elementů, atributů, vlastností a komponentů. [13]

Mnoho modulů si definuje svoje vlastní direktivy. Mezi nejpoužívanější patří:

- **NgClass** – přidává nebo maže CSS třídy na specifikovaném HTML elementu. Funguje na základě specifikování podmínky, jejíž splnění způsobí aplikování specifikovaných CSS tříd. [13]

Structural directives

Strukturální direktivy jsou zodpovědné za rozložení HTML obsahu. Upravují strukturu DOM dokumentu pomocí přidávání, mazání nebo manipulací elementů, ke kterým jsou připojeny. [13]

Mezi nejpoužívanější patří:

- **NgIf** – na základě splnění specifikované podmínky zobrazuje nebo schovává elementy, na které je napojen. Pokud je podmínka nepravdivá, tak *NgIf* smaže elementy z DOM a tako uvolní instance jejich komponentů a uvolní tak paměť. [13]
- **NgFor** – direktiva pro opakované zobrazení elementu, pro každý prvek v poli. [13]

2.2 React

React je open-source JavaScriptová knihovna pro vytváření uživatelských rozhraní. O jeho správu se stará tým Facebooku a komunita individuálních vývojářů a firem. [15]

React není na rozdíl od Angularu plným frameworkem. Jeho hlavním cílem je pouze správa stavu aplikace a vykreslování do DOM. Pro vytváření kompletních aplikací je třeba použít další knihovny pro navigaci a různé další potřebné funkcionality. [15], [16]

2.2.1 Historie

React začínal jako knihovna pro XHP – verze PHP od Facebooku. Cílem XHP projektu bylo zjednodušit vývoj frontendu a zabránit cross-site skriptovacím útokům (XSS). XSS je běžným skriptovacím útokem, který vloží útočný kód do webové aplikace. [17]

XHP však selhalo při vytváření dynamických webových aplikací. XHP stále používalo systém, kdy se musel překreslit celý DOM při změně stavu aplikace. Vývojáři v týmu Facebook Ads Org si uvědomili, že tento systém je nevhodný. [17]

V roce 2011 začal vývojář Jordan Walke pracovat na prototypu, který by tento proces zjednodušil a poskytnul by kvalitnější uživatelský zážitek. Tímto prototypem byla knihovna React. Během pár měsíců se knihovna začala používat na Facebooku pro funkcionalitu lajků a komentářů. [17]

V roce 2012 Facebook koupil Instagram a jedním z prvních úkolů bylo vytvoření jejich webové aplikace. První myšlenkou pro tuto práci bylo použití Reactu, ten byl však v tuto dobu pevně propojený s technologiemi Facebooku a nedal se používat externě. Vývojář Pete Hunt se pustil do práce a přetvořil React do nezávislé knihovny, která se dala používat nezávisle na ostatních technologiích. Instagram se tak stal prvním projektem, který externě používal knihovnu React. [17]

V roce 2013 React přešel na open-source a otevřel se tak vývojářům z celého světa. Od té doby je tato knihovna používaná v aplikacích od firem jako Trello, Slack, Docker nebo Airbnb. [17]

2.2.2 Vlastnosti

Javascript Syntax Extension (JSX)

JSX je syntaktickým rozšířením JavaScriptu, které umožňuje kombinovat syntaxi JavaScriptu a HTML v jednom kódu. Výsledkem této syntaxe jsou React elementy. [23]

React se pomocí JSX snaží podpořit fakt, že vykreslovací logika je přímo spjata s ostatní logikou uživatelského rozhraní, jako je správa stavu aplikace. Místo oddělení technologií do různých souborů (nebo částí souborů) jako tomu je u ostatních frameworků se v Reactu pomocí JSX dá psát vše na jednom místě. [18]

JSX není podmínkou pro používání Reactu, ale jedná se o vlastnost, které většina vývojářů využívá.

Virtual DOM (VDOM)

React si v paměti udržuje odlehčenou verzi „skutečného“ DOM – VDOM. Manipulace DOM je poměrně pomalou operací, naopak úprava VDOM je poměrně nenáročná, protože VDOM není vykreslován v prohlížeči. [19]

Když dojde ke změně stavu aplikace, tato změna se promítne nejdříve ve VDOM. Následně se provede porovnání původního DOM s aktuálním stavem VDOM a provede se překreslení pouze těch objektů, které jsou rozdílné. Tímto způsobem se šetří výpočetní výkon a nedochází ke zbytečnému překreslování neměnných objektů. [19]

One-way Data Binding

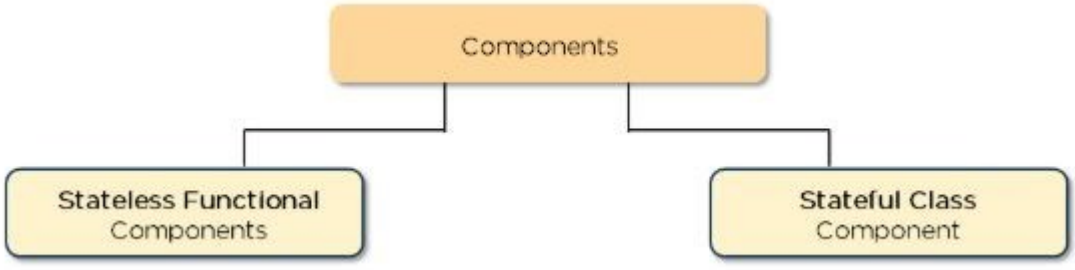
V aplikacích vybudovaných v Reactu data směřují pouze jedním směrem. Data jsou předávána z *parent* komponentů do *child* komponentů pomocí *read-only props*. Tato data nelze posílat zpět do *parent* komponentu. *Child* komponent však stále může informovat jeho nadřazený komponent pomocí *callback* funkcí. [19]

Komponenty

I React patří mezi technologie, které ke tvorbě uživatelského rozhraní používají komponenty. Každý komponent je částí uživatelského rozhraní, jejich cílem je opakovaná použitelnost a nezávislost na zbytku aplikace. [19]

React nabízí možnost použití dvou druhů komponentů:

- **Stateless (functional) komponenty** – jak už napoví název, tyto komponenty neobsahují žádný stav. Jejich obsahem je *render* funkce, ve které se specifikuje obsah, který se vykreslí v prohlížeči. Hodí se pro zobrazování statického obsahu jako je například patička webové stránky. [19]
- **Stateful (class) komponenty** – jedná se o opak *stateless* komponentů. Každý komponent spadající do této kategorie je rozšiřující instancí Reactem poskytnuté třídy *React.Component.* Tyto komponenty mohou udržovat a měnit svůj stav a používají se častěji. V praktickém příkladu by se změna stavu komponentu dala reprezentovat například změnou barvy pozadí komponentu po stisknutí tlačítka. [19]



Obr. 5 React komponenty [19]

State

State je vestavěným objektem Reactu, který se používá k uchovávání dat uvnitř *stateful* komponentů. Při používání aplikace se tento objekt může měnit a React na tyto změny reaguje překreslením částí komponentu, u kterých ke změnám došlo. [19]

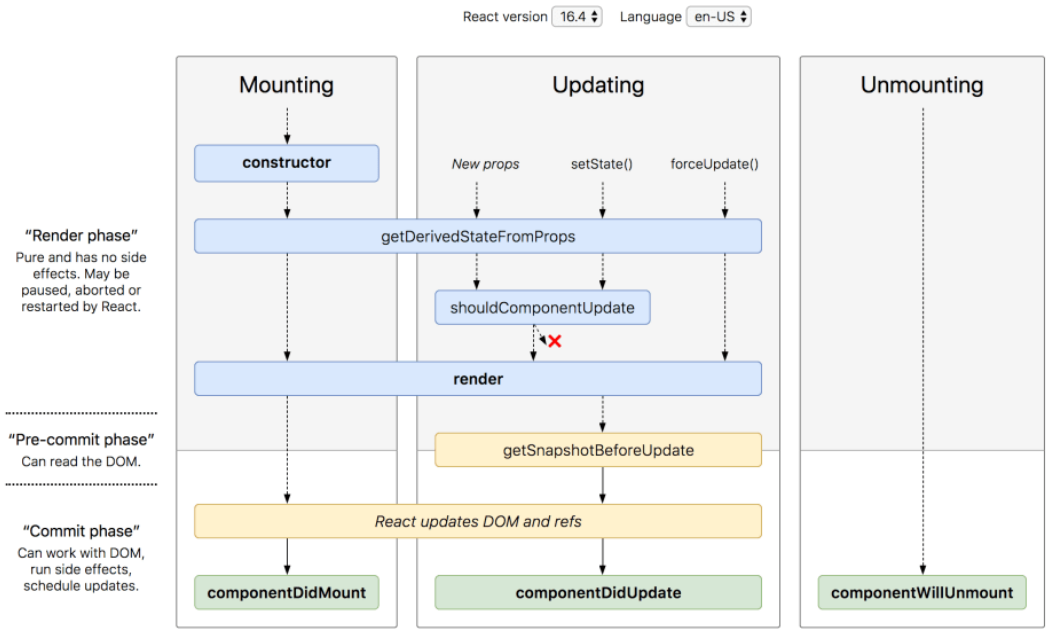
Změnu *state* v Reactu nelze provést pouhým přepsáním dané hodnoty, k tomuto účelu slouží metoda *setState*. Jedná se o asynchronní metodu, která provede změny stavu a zároveň oznámí Reactu, že došlo ke změnám a je potřeba provést překreslení komponentu a případně jeho *child* komponentů. [19]

Props

Props je zkratkou pro *properties* (vlastnosti), v Reactu se jedná o objekt, který ukládá hodnoty HTML tagů a umožňuje tak předávat parametry mezi komponenty. [19]

Životní cyklus komponentů

Během životního cyklu každého komponentu dochází k volání metod, které reagují na různé druhy událostí. Tyto metody je možné přepsat a přidat si do nich vlastní logiku, která se bude provádět při každé aktivaci této události. [20]



Obr. 6 React Component Lifecycle [20]

Tyto metody se dělí do tří fází – *Mounting*, *Updating* a *Unmounting*. Mezi nejpoužívanější metody patří:

- **render** – jedná se o nejpoužívanější metodu a zároveň jedinou metodu, která je povinná v každém *stateful* komponentu. Její volání probíhá v mounting a updating fázích cyklu a jejím cílem je definovat co se bude vykreslovat v prohlížeči. Je potřeba aby metoda neměla žádné vedlejší účinky (aby při vložení stejných parametrů vždy vrátila stejný výstup). Nedovoluje tak například volání funkce *setState* pro aktualizaci stavu. [20]
- **componentDidMount** – jak vyplývá z názvu – tato metoda se volá vždy, když je komponent zapojen a připraven. Jedná se o ideální metodu pro inicializaci dotazů na server. [20]
- **componentDidUpdate** – tato metoda je volána po každém updatu. [20]
- **componentWillUnmount** – název opět napoví. V této metodě lze vytvářet logiku, kterou chceme uskutečnit těsně před odpojením a zničením instance komponentu. [20]

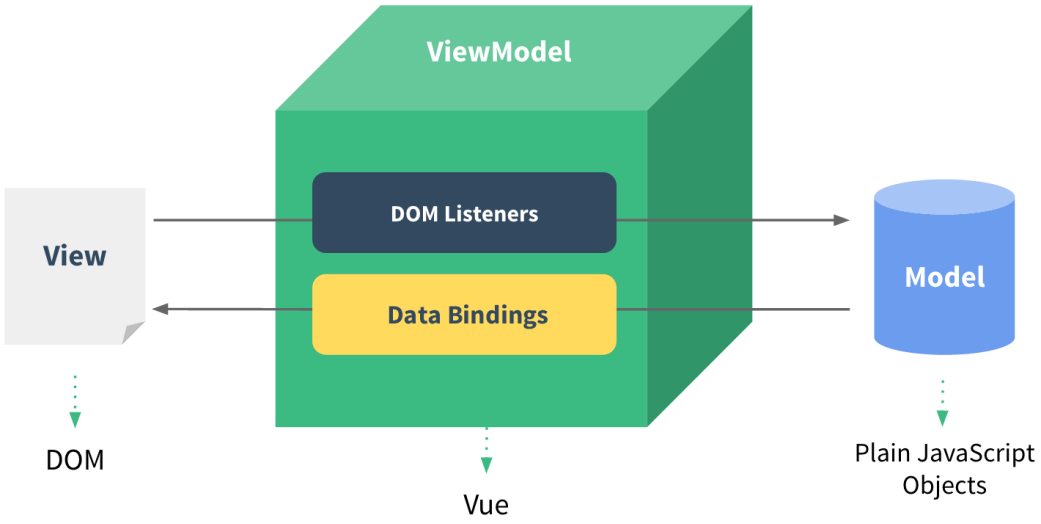
2.3 VueJS

Vue je progresivní JavaScriptový framework pro tvorbu uživatelských rozhraní. Na rozdíl od některých ostatních frameworků je Vue designováno pro inkrementální adopci. Základní framework se zaměřuje pouze na zobrazovací vrstvu a je jednoduché jej integrovat ve spolupráci s ostatními knihovnami nebo jej přidat do existujícího projektu. Stejně tak je Vue vhodné i pro tvorbu komplexních SPA, je-li použito v kombinaci s podpůrnými knihovnami a jinými moderními nástroji. [21]

2.3.1 Vlastnosti

Reactive Data Binding

Vue používá systém reaktivního propojení dat, který se stará o synchronizaci dat a DOM. V praxi se jedná o speciální syntaxi uvnitř HTML šablony, která „nabinduje“ výstup v DOM na data v pozadí. Když je toto propojení hotové, tak se Vue už postará o synchronizaci bez jakékoliv námahy od vývojáře. Výsledkem odpadá nutnost přímé manipulace objektů v DOM, přechází se na přímou manipulaci dat. Kód se tak lépe píše i čte a mnohem jednodušeji se pak udržuje. [21]



Obr. 7 Reactive Data Binding ve Vue [21]

Virtual DOM

Tak jako React, i Vue používá VDOM. V obou případech jde o stejný princip, jehož cílem je zlepšení výkonu při změně stavu aplikace. [21]

Šablony

Vue používá šablony založené na HTML. Tyto šablony umožňují deklarovat propojení elementů v DOM s daty uvnitř Vue instance. [21]

Na pozadí Vue zkompiluje šablony do VDOM a ve spolupráci s reaktivním systémem je schopno inteligentně rozpoznat minimální množství komponentů, které je potřeba překreslit a také minimální množství manipulací s DOM, které je k tomu potřeba udělat. [21]

V zájmu podpory nejrůznějších typů vývojářů Vue nabízí možnost využití JSX a manuálního psaní *render* funkcí místo použití šablon. Tuto funkcionalitu mohou ocenit právě vývojáři se zkušenostmi z Reactu. [21]

Data a metody

Instance Vue obsahuje několik objektů, které používá ke své funkcionalitě. Objekt *data* slouží k definování dat, se kterými se vývojář chystá pracovat. Po vytvoření instance se Vue podívá do tohoto objektu a všechny naleznuté vlastnosti přidá do svého reaktivního systému. Při změně některého z těchto dat Vue zareaguje překreslením potřebných změn. Je nutno podotknout, že data, která do tohoto objektu přidáme až po vytvoření Vue instance, již

nebudou reaktivní. Pro dosáhnutí reaktivity je potřeba takové vlastnosti definovat předem a přiřadit jim prázdnou hodnotu. [21]

Dalším objektem, který ve své instanci Vue využívá, je objekt *methods*. V tomto objektu lze definovat metody, které uvnitř komponentu budeme využívat. Metody lze volat navzájem mezi sebou uvnitř JavaScriptové části komponentu a také je lze volat ze šablony. [21]

Computed Properties

Výrazy uvnitř šablony jsou velmi užitečné a pohodlné na používání, měly by se však používat pouze pro jednoduché operace. Příliš mnoho logiky v šabloně může způsobit nepřehlednost kódu, se kterou se špatně pracuje. Všechny výrazy uvnitř šablony by měly být jednoduše čitelné již na první pohled. [21]

Pro potřeby složitějších výrazů slouží *computed property*. Ve své podstatě se jedná o *getter*, vracející výraz pro vykreslení v šabloně. Rozdíl oproti použití volání klasické metody je v cachování hodnot z *computed property*. V případě, že použijeme *computed property* v šabloně více než jednou, tak se výraz zavolá jenom jednou a vrátí stejný výsledek do všech pozic v šabloně. Ušetří tím tak výpočetní výkon. [21]

Watchers

Pro většinu případů je vhodnější používat *computed property*, existují však případy, kdy je třeba vykonávat změny v datech na základě změny jiných dat – pro tyto potřeby je tady objekt *watch*. V tomto objektu lze definovat metody, které se budou volat v případě změny některé vlastnosti v objektu *data*. [21]

Komponenty

Vue využívá systému komponentů podobně, jako to dělá Angular i React. I v tomto případě se jedná o izolované části UI, které lze opakovaně používat. [21]

Pro opakované používání komponentů je třeba jim předávat data. K tomu slouží objekt *props*. Tento objekt lze definovat v komponentu stejným způsobem, jako se definují objekty *data* nebo *methods*. V *parent* komponentu následně tento objekt naplníme při deklaraci komponentu uvnitř šablony. [21]

Mixins

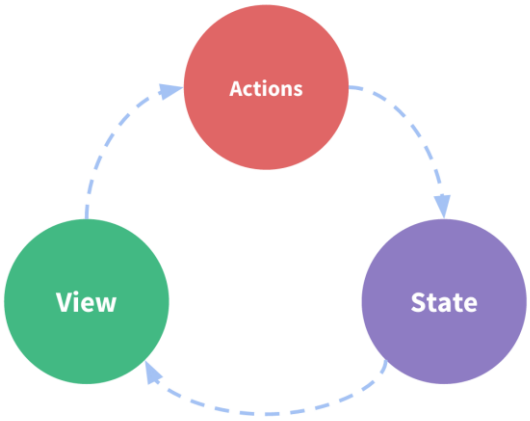
Mixin je flexibilním způsobem distribuce funkcionality komponentů. Mixin může obsahovat jakékoliv vlastnosti komponentů. Pokud komponent použije mixin, tak se všechna funkcionalita mixinu „zamixuje“ do funckionality komponentu. Lze tak například definovat metodu, kterou bude používat několik různých komponentů. [21]

Mixin lze definovat lokálně nebo globálně. Lokální mixin bude použit pouze v komponentech, které se ho vyžádají. Globální mixin naopak bude aplikován na všechny komponenty a jeho použití se příliš nedoporučuje. [21]

2.3.2 Vuex

Vuex je knihovna pro správu stavu aplikací ve Vue. Vuex není součástí oficiálního balíčku Vue, ale je oficiálním doplňkem pro Vue. Slouží jako centralizovaný sklad pro všechny komponenty v aplikaci a má pravidla, která povolují úpravu stavu aplikace pouze podle předpokládaných kroků. [22]

Správa stavu uvnitř jednoho komponentu je poměrně jednoduchá. Lze ji definovat velmi jednoduše, viz Obr. 8. [22]

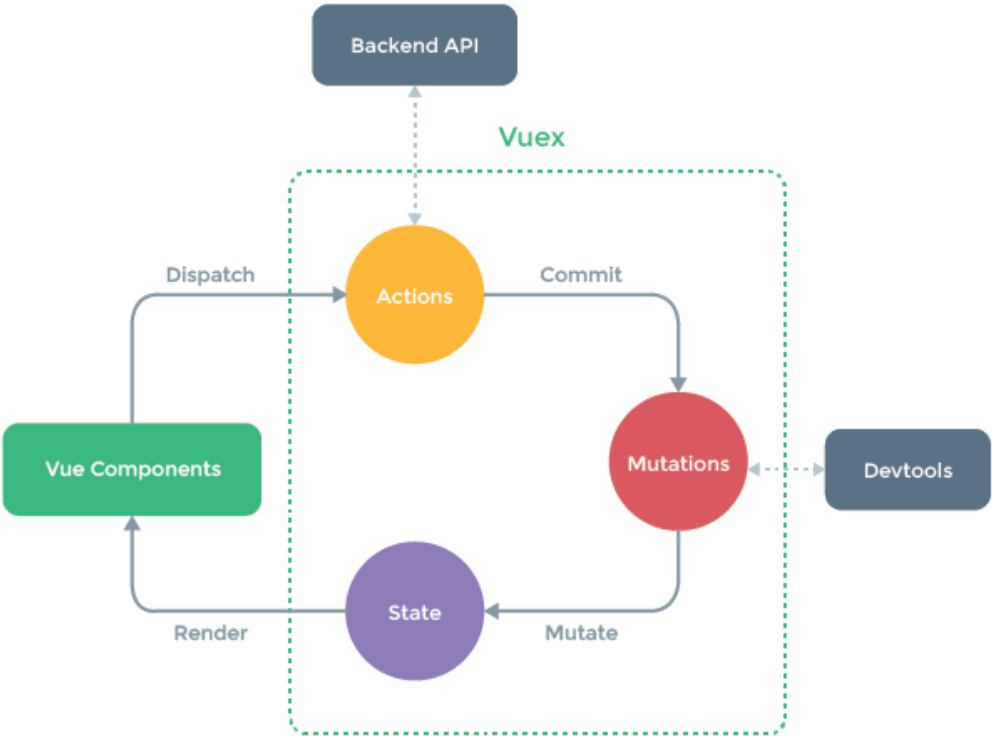


Obr. 8 Správa stavu ve Vue [22]

Jednoduchost však začne narážet na problémy, když máme několik komponentů, které sdílejí společný stav:

- Několik komponentů může záviset na stejné části stavu. [22]
- Akce z několika komponentů mohou upravovat stejnou část stavu. [22]

Právě tyto problémy se snaží řešit Vuex. Samotná knihovna je založená na stejných principech jako je Flux nebo Redux. Na rozdíl od těchto knihoven, které se často používají u jiných frameworků je však Vuex navrženo přímo pro Vue s ohledem na maximální využití jeho reaktivního systému. [22]



Obr. 9 Správa stavu s pomocí Vuex [22]

Obr. 9 znázorňuje správu stavu s využitím Vuex. Z obrázku vyplývá, že veškerá správa společného stavu se nevykonává uvnitř komponentů, nýbrž uvnitř Vuex. Komponenty pouze do Vuex předávají informaci o provedení uživatelské akce. Samotná knihovna následně provede úpravu stavu odpovídající dané akci, následně předá všem komponentům využívajícím daný stav informaci o jeho změně. [22]

2.4 Srovnání

Každá technologie má svoje výhody a nevýhody. Nedá se říct, že by některá z nich byla lepší než jiná, zkrátka se každá může hodit na jiný druh projektu. Nelze také opomenout subjektivní názor samotného vývojáře. Ve své podstatě lze v každé technologii vytvořit

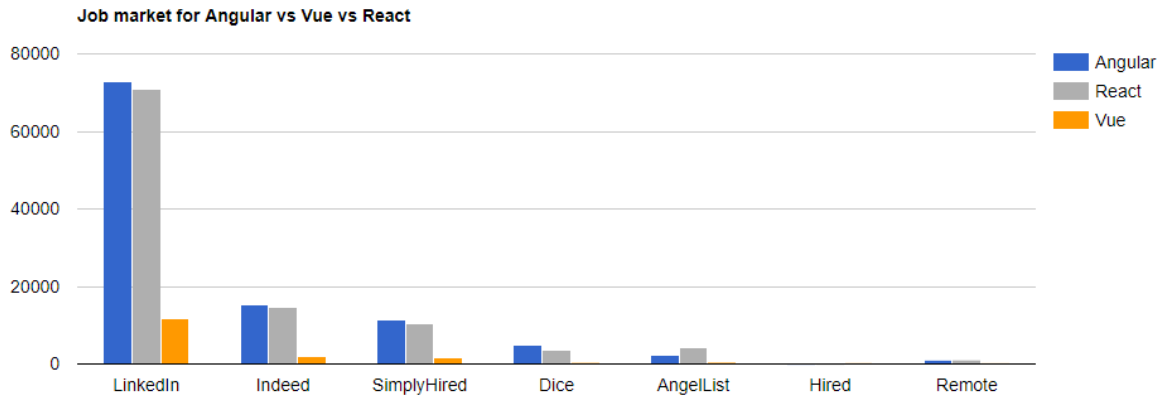
stejné věci jako v ostatních a to aniž by si toho samotný uživatel nějak všimnul. Rozhodujícím faktorem se tak stává názor vývojáře a jeho vlastní preference.

2.4.1 Popularita

Popularita by se dala měřit několika způsoby. Každý z těchto způsobů však bude mít svoje mouchy. Jedním z nich je počet hvězd, které jednotlivé technologie obdržely na GitHubu. Jelikož jsou všechny tři open-source, je jejich kód veřejně dostupný a každý do něj může přispět. Dalo by se tedy říct, že počet hvězd, které tyto projekty obdržely, svědčí o popularitě mezi vývojáři. [23]

Dle této metriky je aktuálně mezi vývojáři nejoblíbenější Vue se 184 tisíci hvězdami. V těsném závěsu je React se 169 tisíci hvězdami a se značným odstupem je Angular s pouhými 73 tisíci. Z této metriky by se tedy dalo odvodit, že Vue je nejpoužívanější technologií a Angular je výrazně pozadu. Je třeba se však podívat i na jiné srovnání. [23]

Asi nejlepší obrázek o popularitě si můžeme udělat při pohledu na nabídky prací pro jednotlivé technologie.



Obr. 10 Srovnání nabídky prací pro jednotlivé technologie [23]

Metrika použitá na Obr. 10 nám nabízí velmi rozdílný pohled na věc, než-li . Lze jasně vidět přibližně podobnou poptávku po vývojářích zabývajících se Reactem a Angularem zatímco Vue se v těchto srovnáních dostává pouze přibližně na 20% poptávky svých konkurentů. [23]

Pokud použijeme obě metriky najednou, lze jednoduše vyvodit, že React se jeví velké popularitě jak mezi vývojáři, tak i mezi firmami. Naopak Vue se zatím na trhu nedostává takové pozornosti, jakou by si asi zasloužilo dle popularity na GitHubu. Relativně malá popularita Angularu na GitHubu by se dala vysvětlit menší potřebou pro vývoj komunitních nástrojů. Angular je ve svém základu vydávaný se vším, co je třeba pro vývoj rozsáhlých

aplikací, zatímco například React se bez použití komunitních rozšíření prakticky nedá použít pro vývoj komplexních aplikací. [23]

2.4.2 Práce s Angular, React a Vue

Pro vývojáře je důležité se podívat na charakteristiky, které se týkají samotného vývoje aplikací. [23]

Křivka učení

React je pro začátečníky nejpřívětivějším způsobem, jak se dostat k vývoji SPA. Ze všech tří konkurentů se těší největší popularity a je pravděpodobné, že jakýkoliv problém, na který vývojář narazí, už dříve někdo řešil. React je však pouze knihovnou, která si sama o sobě nedokáže poradit s vývojem kompletních SPA a je tedy třeba použít další knihovny. V tomto směru záleží na samotném vývojáři, kudy se vydá. [23]

Vue je také poměrně přístupné pro začátečníky. Na rozdíl od Reactu poskytuje vývojářům více funkcionality a prakticky vše potřebné pro vývoj SPA je dostupné v oficiálních balíčcích. Vue tedy nabídne více funkcionality, což způsobí o něco větší náročnost pro začátečníky. [23]

Angular je rozhodně nejnáročnějším frameworkem pro začátečníky. Nutnost používání TypeScriptu je skvělým doplňkem, bohužel to však způsobuje nutnost studia další technologie. Angular je sám o sobě plně funkčním frameworkem, který v oficiálním vydání poskytne vše, co je potřeba. Svým způsobem tak vývojářům určuje jednu správnou cestu, kterou se ubírat. Na rozdíl od Reactu i Vue nejde Angular jednoduše vložit do stávajícího projektu a používat jen některé funkcionality. Ve zkratce Angular nabízí větší spektrum funkcionality výměnou za jednoduchost. [23]

TypeScript

Angular je jediným ze tří zmíněných technologií, která ve svém základu používá TypeScript. React i Vue nabízí možnost používání TypeScriptu, nejedná se však o integrovanou potřebu a podpora tak může být horší než u Angularu. [21]

HTML a CSS

React pro veškerou logiku používá JSX. To oproti šablonám ve Vue i Angularu nabízí několik výhod. Je třeba zdůraznit, že JSX lze používat i ve Vue, nejedná se však o obvyklý způsob a podpora tak nebude na úrovni Reactu.

- Lze použít veškerou funkcionalitu JavaScriptu uvnitř vykreslování. Vue i Angular pro propojení HTML šablon a modelu používají direktivy, které sice svůj účel splní, ale nemohou se srovnat se samotným využitím JavaScriptu uvnitř HTML. [21]
- Podpora editoru je v JSX mnohem lepší než cokoliv, co mohou nabídnout direktivy. Ať už se jedná o napovídání *autocompletem*, podtrhávání potenciálních chyb nebo ověřování typů, JSX bude mít vždy navrch. [21]

Vývoj rozsáhlých aplikací

Ve své podstatě se dá říct, že si zde nelze vybrat špatně. Všechny technologie nabízí robustní systémy pro vývoj rozsáhlých korporátních aplikací.

Angular je pro tyto typy aplikací stavěný. Framework nabízí pro tyto typy aplikací veškerou podporu a všechny potřebné moduly jsou udržované týmem, který se stará o vývoj samotného frameworku. Za všech okolností se tedy vývojáři mohou spolehnout na plnou synchronizaci verzí všech potřebných modulů. [21]

React naopak nechává vývoj knihoven, které slouží například pro správu stavu nebo pro navigaci, na své komunitě. Komunita Reactu je obrovská a pro jakýkoliv problém pravděpodobně nebude těžké naleznout hned několik různých řešení. Bohužel však mohou nastat problémy se synchronizací jednotlivých knihoven, komunitní vývojáři nemají povinnost vše udržovat a může tak docházet k problémům. [21]

Vue je v tomto ohledu někde na půl cesty. Oficiální knihovny udržované hlavním týmem poskytují vše, co je potřeba (Vuex pro správu stavu a Vue Router pro navigaci). Jedná se tedy o kompromis mezi striktnějším návrhem aplikací v Angularu a mnohem volnějším způsobem návrhu aplikací v Reactu. [21]

3 ELEKTRONICKÉ INFORMAČNÍ ZDROJE

Elektronický informační zdroj (EIZ) je informační zdroj uchovávaný v digitální podobě a je dostupný prostřednictvím internetu nebo jiných technologií přenosu dat (např. CD-ROM). [24]

3.1 Dělení EIZ

EIZ vznikají jak v elektronické tak i tištěné formě. Dají se dělit podle jejich tématiky, typu nebo rozhraní. Jejich cílem je umožnit jednoduché i pokročilé vyhledávání, a prohlížení rejstříku nebo indexů. [25]

EIZ se dělí podle několika základních atributů:

- Z hlediska typu
    - Online katalogy
    - Databáze
    - Digitální knihovny
    - Oborové brány
- Z hlediska tematického a oborového
    - Jednooborové
    - Multioborové
    - Univerzální
- Z hlediska technického zpřístupnění
    - Přístup online
    - Přístup offline
- Z hlediska dostupnosti
    - Volně dostupné
    - Licencované

- Z hlediska popisu primárního dokumentu
    - Plnotextové
    - Bibliografické
    - Faktografické
    - Obrazové
    - Multimediální
- Z hlediska původnosti obsahu
    - Primární
    - Sekundární
    - Terciální

3.2 Typologie podle typu zpřístupňované informace

Bibliografické databáze

V těchto databázích se nachází pouze informace o primárních dokumentech (dokumentech obsahujících plný text) ve formě bibliografických záznamů. Bibliografický záznam se značí specifikováním několika parametrů jako je např. jméno autora, název článku, rok vydání aj. Neobsahuje tedy plné texty dokumentů. V minulosti byl tento typ databází výrazně nejpoužívanější a i v současné době se nejprestižnější databáze na světě řadí mezi tento typ databází. Informace jsou těmto databázím poskytovány obvykle z odborných časopisů a jejich rozsah může zahrnovat až tisíce titulů. [26], [27]

Aktuálně se tento typ databází rozšiřuje o odkazy na plné texty vedoucí na stránky jiných, komerčních institucí. Z tohoto důvodu se k plnému textu povede přistoupit pouze zřídka. K propojování databází se používá systém Digital Object Identifier (DOI), kdy se ke každému materiálu přidělí jedinečný identifikátor, podle kterého se dá na daný materiál jednoduše propojit. Bohužel ani DOI plně neřeší problém se získáváním plných textů, a tak si jednotlivé databáze budují vlastní kolekce plných textů. [26]

Plnotextové databáze

Jak už napoví název – v těchto databázích se kromě bibliografických údajů objevují i plné texty dokumentů. Ne všechny databáze však poskytují plné texty, licenční politika některých vydavatelů nedovoluje poskytnout více než pouhý abstrakt dokumentu. V jiných případech může dojít ke zpřístupňování dokumentů s časovým odstupem – někdy i více než půl roku od vydání časopisu. [26], [27]

Pomalu ale jistě se tento druh databází dostává do popředí, bohužel však s nimi jsou spojena i různá negativa – např. vysoké nároky na provoz. [26]

Faktografické databáze

V těchto databázích se shromažďují fakta a data o nejrůznějších věcech. Faktografické databáze se používají v nejrůznějších odvětvích, mezi tyto patří například ekonomické vědy, statistiky, chemie nebo geografie. Lze sem zařadit i encyklopedie. [26]

Citační databáze

Sem se řadí databáze obsahující citace prací z oblasti výzkumu a vývoje.

3.3 Typologie vzdáleného přístupu

V případě licencovaných databází je nutností mít předplacenou licenci k těmto zdrojům. Obvykle tyto licence předplácí knihovny a přístup ke zdrojům je prováděn ze specifikované množiny IP adres, které odpovídají dané knihovně. Pro běžného uživatele tedy není problém se k licencovaným zdrojům dostat z počítačů umístěných přímo v knihovně. Fyzicky navštěvovat knihovnu vždy, když se chce uživatel dostat ke zdrojům je však značně nepohodlné. Jako řešení tohoto problému byly vyvinuty systémy tzv. vzdáleného přístupu. [26]

3.3.1 Shibboleth

Cílem systému Shibboleth je poskytnutí vzdáleného přístupu uživateli, který se ke zdroji snaží dostat mimo rozsah povolených IP adres. Shibboleth uživateli nabídne seznam institucí, které k danému zdroji mají přístup. Pokud uživatel má účet v některé z těchto institucí, nechá se přesměrovat na její přihlašovací stránky, kde provede klasické přihlášení do systému. Po úspěšném přihlášení je přesměrován zpět na požadovaný zdroj, kde je již označen jako člen příslušné instituce a má zprovozněny služby, které jsou této instituci poskytnuty. [26]

Projekt je kooperací dvou druhů subjektů. Prvním druhem jsou předplatitelé (univerzity, knihovny, vědecké instituce) – označováni jako poskytovatelé identity. Druhým druhem jsou samotní producenti informačních zdrojů, kteří se označují jako poskytovatelé služeb. Systém Shibboleth je rozšířený do takové míry, kdy je na jeho integraci připravena drtivá většina prestižních vydavatelství. [26]

3.3.2 Hidden Automatic Navigator (HAN)

Jedná se o softwarový produkt od společnosti H+Software GmbH. Uživatel software použije k ověření svojí identity. Pokud je ověření úspěšné, je uživateli přidělena IP adresa z rozsahu univerzitních IP adres a zároveň i přístupová práva ke zdrojům, ke kterým se snaží dostat. Podobnou funkcionalitu by zvládnul i klasický proxy server, ale HAN nabízí mnohem robustnější řešení, které se specializuje přímo na elektronické zdroje a nenabízí přístup pouze ke zdrojům omezených IP adresou, ale případně i heslem. Navíc je zde implementovaný systém, který uživateli pomůže s přístupem ke zdrojům omezeným množstvím současně připojených uživatelů. V případě vyčerpání všech volných míst uživatele zařadí do fronty a umožní mu přístup po uvolnění předchozím uživatelem. [26]

3.3.3 EZproxy

EZproxy je prostředníkem, který zvládá i spolupráci s ostatními typy vzdáleného přístupu. Jedním z rozdílů oproti ostatním systémům je možnost identifikace uživatelů podle skupin. V praxi tak lze například rozdělit zdroje univerzity na přístupy podle jednotlivých fakult a lépe rozdělit jednotlivé přístupy a poskytnout uživateli pouze zdroje, ke kterým má na základě svého zaměřením přístup. [26]

3.3.4 SeamlessAccess

SeamlessAccess je nejnovějším řešením, které buduje na základech ostatních. Zatím tuto možnost nabízí pouze několik zdrojů, ale dá se předpokládat, že se časem budou přidávat další.

Na rozdíl od Shibbolethu zde není třeba přecházet na stránky univerzity a provést autentizaci na univerzitních webových stránkách. SeamlessAccess nabízí možnost přihlášení do účtu poslední vybrané instituce přímo na stránkách zdroje. [28]

4 BEZPEČNOST WEBOVÝCH APLIKACÍ

Webové aplikace mohou být v případě nesprávného navržení náchylné k útokům zvenčí. Druhů těchto útoků existuje hned několik a jejich dopady mohou způsobovat krádež privátních dat, zničení dat, napadnutí počítačů jiných uživatelů nebo i vyřazení webové aplikace z provozu.

4.1 Cross-Site Scripting

Cross-Site Scripting (XSS) útoky jsou prováděny vkládáním útočných skriptů do jinak neškodných a důvěryhodných webových aplikací. K tomuto útoku dojde, když útočník využije webovou aplikaci pro poslání kódu – obvykle ve formě prohlížečového skriptu na adresu jiného koncového uživatele. Chyby v designu, které dovolují provádění XSS, jsou rozšířeny po celém webu a obvykle souvisí s přijímáním vstupu od jednoho uživatele a následné použití tohoto vstupu pro vygenerování výstupu pro jiného uživatele, a to bez validace daného vstupu. [29]

Uživatel, který se stane obětí XSS, nemá vůbec tušení, k čemu může dojít. Stejně tak nemá ani jeho prohlížeč šanci rozpoznat, že skript z důvěryhodného zdroje vlastně není důvěryhodný. Protože skript přichází z důvěryhodného zdroje, má přístup k nejrůznějším datům, která prohlížeč uchovává. Mezi tato data patří cookies a session tokeny. Navíc může tento skript změnit i samotné HTML stránky. [29]

4.2 SQL Injection

SQL Injection spočívá ve vložení SQL dotazu z klientského vstupu webové aplikace do databázové vrstvy. Úspěšný útok může vést ke krádeži citlivých dat, jejich upravení nebo zničení, provedení nepovolených akcí jako třeba nastavení administrátorského přístupu a v některých případech i provedení akcí v operačním systému. [29]

Nejčastěji se tento typ útoku objevuje v aplikacích založených na PHP a ASP díky využívání starých funkčních rozhraní. V modernějších technologiích je stále těžší tento útok úspěšně provést. [29]

4.3 Distributed Denial of Service (DDoS)

Cílem DDoS útoku je narušit běžný provoz cíleného serveru, služby, nebo sítě přehlcením cíle nebo jeho pomocné infrastruktury velkým množstvím internetového provozu. [30]

DDoS útoky ke své efektivitě často používají ukradené nebo nabourané zařízení, které jsou využity k vytvoření internetového provozu. Přehlcení napadené služby může způsobit výpadky provozu, zpomalení dotazů nebo i kompletní vyřazení aplikace z provozu. [30]

Tomuto druhu útoku se dá bránit pomocí různých řešení:

- **Rate limiting** – omezení počtu povolených dotazů na server je relativně efektivním způsobem, jak aplikaci zabezpečit proti velkému množství nevyžádaných dotazů. Proti komplexnímu DDoS útoku však není dostatečnou ochranou. [30]
- **Web Application Firewall (WAF)** – WAF je implementován jako ochrana mezi internetem a serverem a může sloužit jako ochrana serveru proti některým druhům nevyžádaného provozu. [30]
- **Anycast network diffusion** – tento ochranný systém použije Anycast network na rozprostření dotazů do sítě distribuovaných serverů až do bodu, kdy je provoz pohlcen sítí. Tento způsob ochrany ve své podstatě rozprostře útok do větší šířky a může pomocí proti s ustáním velkého množství provozu. Vždy však bude záležet na velikost DDoS útoku a velikosti, efektivitě distribuované sítě. [30]

II. PRAKTICKÁ ČÁST

5 EIZ POUŽÍVANÉ V KNIHOVNĚ UTB

Knihovna UTB v době psaní této práce zpřístupňuje seznam 107 EIZ. Zdroje jsou děleny podle několika kategorií. Některé z těchto kategorií se týkají obecných parametrů EIZ, jiné se zaměřují přímo na UTB a dělí obsah dle potřeb univerzity. Uvnitř aplikace jsou hodnoty jednotlivých kategorií reprezentovány pomocí tagů a umožňují filtraci EIZ.

Kategorie tagů jsou:

- **Přístup** – v této kategorii je ke každému EIZ přiřazena pouze jedna hodnota. Zdroje se dělí na licencované, volně dostupné a zdroje se zkušebním přístupem.
- **Typ zdroje** – většina zdrojů spadá pouze pod jeden typ, existují však i výjimky. Nejobvyklejší jsou plnotextové databáze, bibliografické databáze a faktografické databáze. V menšině se objevují např. citační a strukturní databáze.
- **Typ obsahu** – obsah EIZ pochází nejčastěji z e-časopisů a e-knih. Dále se zde objevují konferenční materiály, patenty, statistiky a také akademické práce.
- **Fakulta/součást** – tato kategorie zdroje dělí dle vhodnosti pro jednotlivé fakulty UTB. Většina zdrojů nespadá pod žádnou konkrétní fakultu a je označena hodnotou „multioborové“. Z konkrétních fakult má dle dat z EIZ největší zastoupení fakulta FT. Ostatní fakulty jsou zastoupeny přibližně stejně.
- **Obory** – zde probíhá dělení dle jednotlivých studijních zaměření. I zde je většina EIZ označena jako „multioborová“. Ostatní zdroje jsou nejčastější vhodné pro biologii, chemii, přírodní vědy, společenské vědy a sociologii. Najdou se zde však zdroje prakticky téměř pro všechny obory. Jediné obory, které zde nemají konkrétní zastoupení, jsou móda a ošetřovatelství.
- **Jazyk** – z hodnot v této kategorii lze vidět, že 65 ze 107 EIZ je zpřístupněno v anglickém jazyce, 24 zdrojů je vícejazyčných a pouze 14 zdrojů je v českém jazyce. Jediné další zastoupení zde má jazyk německý, ve kterém jsou publikovány 2 EIZ.

Na základě sítě se v aplikaci dělí přístup k jednotlivým zdrojům. Volně přístupné zdroje jsou vždy dostupné přes přímý odkaz. Licencované a zkušební zdroje jsou zpřístupněny pomocí přímého odkazu uvnitř interní sítě UTB. Pokud uživatel přistupuje z externí sítě, tak jsou EIZ dostupné přes Shibboleth. Jestliže není odkaz na Shibboleth dostupný, je přístup zprostředkován s pomocí EZproxy.

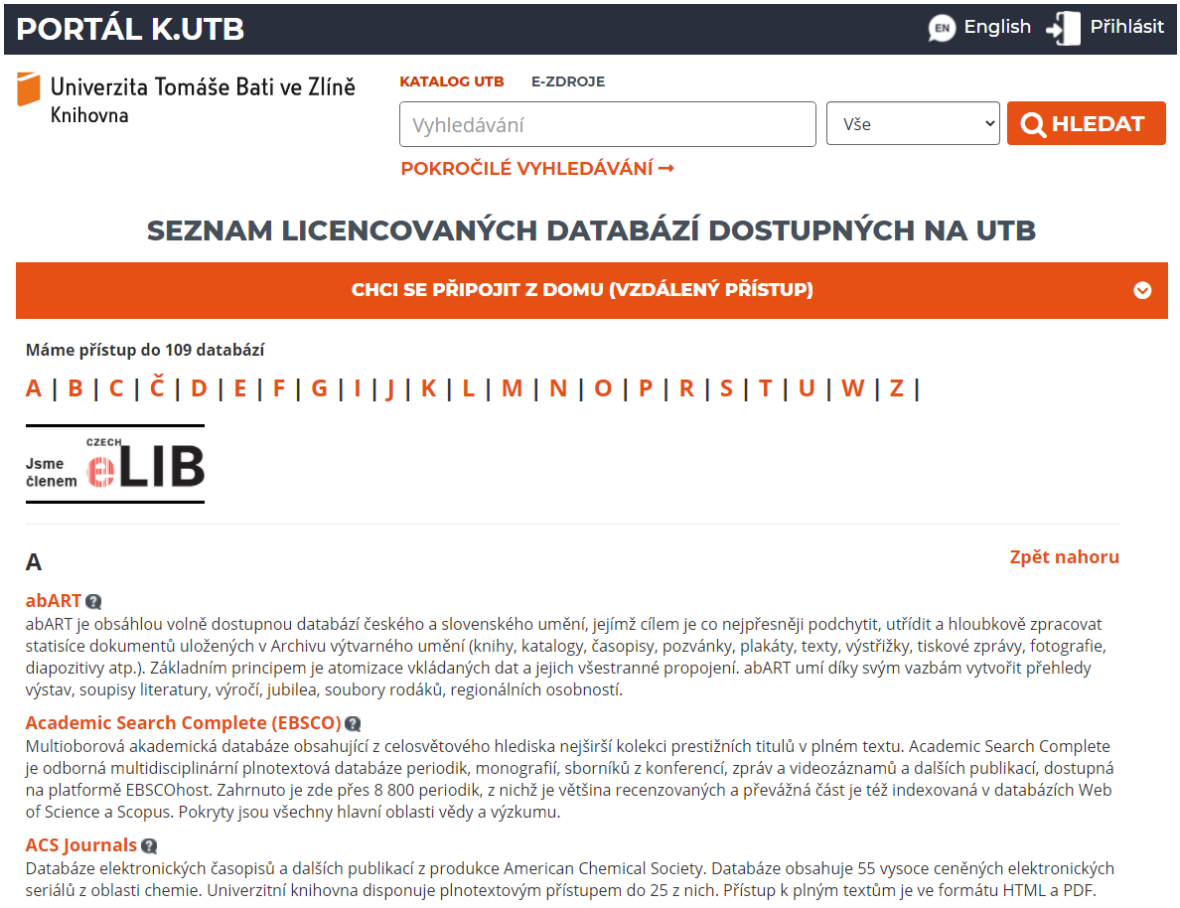
6 POŽADAVKY NA ROZHRANÍ

Diplomová práce byla zadána Knihovnou UTB. Práce se zabývá vytvořením univerzálního rozhraní pro zobrazování EIZ, které nahradí dosavadně používané řešení.

6.1 Stávající řešení

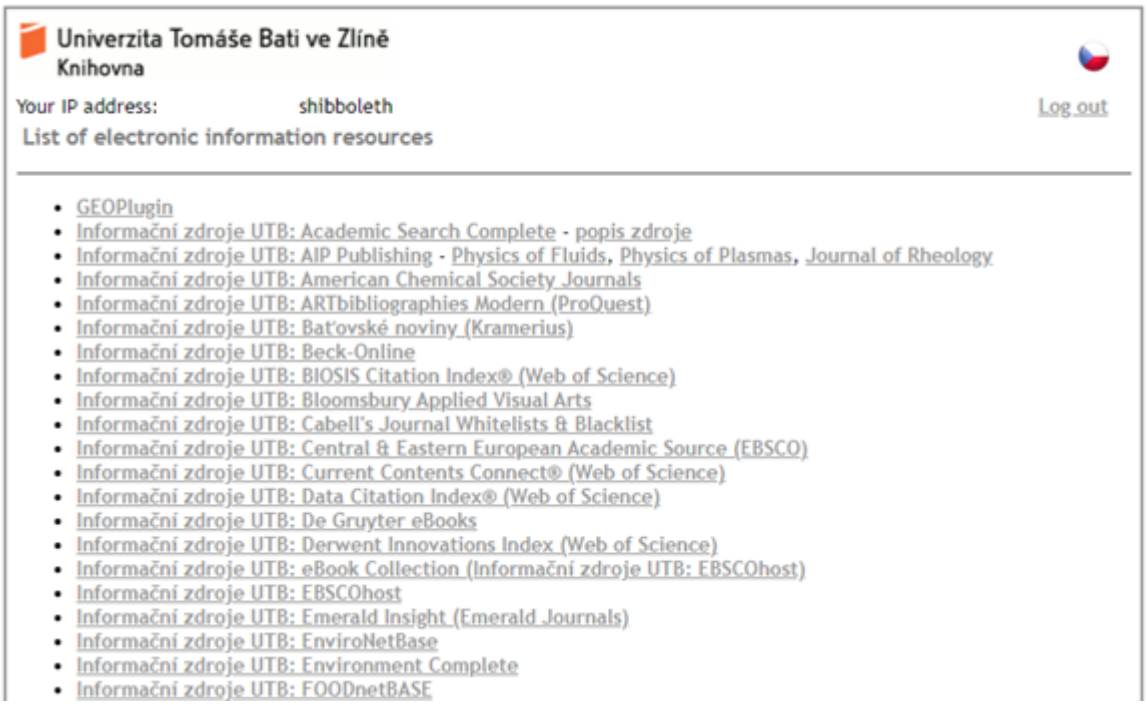
Knihovna UTB pro zobrazování EIZ aktuálně používá dvě nezávislé webové aplikace. Dvě aplikace jsou používány kvůli rozdělení podle přístupu z interní nebo externí sítě.

První z aplikací funguje pro přístup z interní sítě UTB, kde jsou i licencované informační zdroje zpřístupněny pod přímým odkazem. Náhled aplikace je dostupný na Obr. 11. Aplikace je zprovozněna použitím technologie PHP a funguje v rámci knihovního katalogu VuFind.



Obr. 11 Katalog K.UTB

Druhá aplikace slouží pro přístup z externí sítě a nabízí uživatelům přihlášení pomocí Shibbolethu. Informační zdroje jsou následně zpřístupněny díky tomuto propojení. Vzhled dosavadní aplikace, která je zpřístupněna po přihlášení přes Shibboleth, lze pozorovat na Obr. 12.



Obr. 12 Proxy K.UTB

Uživatelské rozhraní pro zobrazení zdrojů z externí sítě je zprovozněno v podobě několika téměř statických HTML stránek v rámci komerční technologie EZproxy. Kromě seznamu EIZ (viz Obr. 12) je zde přítomno ještě několik chybových stránek, přihlašovací stránka a administrativní rozhraní.

6.2 Požadavky na nové řešení

Nové řešení musí splňovat všechny parametry, kterými se aktuálně prezentuje stávající řešení. Řešení musí být zpracováno ve formě webové aplikace. Tato aplikace musí nahradit obě aktuálně používané webové aplikace ve formě jednoho řešení. Je tedy nutné rozlišovat síť, ze které se uživatel připojuje, dle tohoto údaje je nutné měnit odkazy na jednotlivé zdroje.

O grafický návrh aplikace se postará zaměstnanec UTB a z pohledu této práce je tedy úkolem grafický návrh dostat do funkčního stavu.

Hlavní část webové aplikace je nutno implementovat ve formě seznamu dostupných EIZ. Každý zdroj v seznamu bude zobrazen ve formě kartičky, která bude obsahovat název zdroje, odkaz na zdroj, proklik na detail zdroje a několik doplňujících údajů, např. krátký popis zdroje nebo typ přístupu.

Aplikace se bude zabývat pouze klientskou částí – tzv. frontend. Databázová a serverová logika je již implementována a pouze se převezme z předchozích řešení. Seznam EIZ bude zpřístupněn pomocí jednoho API endpointu, který vrátí celý seznam ve formátu JSON. Je nutno si tento seznam uložit na straně klienta a veškeré operace již provádět bez asistence serveru. Celé zobrazení musí být plně responzivní a přístupné na všech zařízeních.

V seznamu je třeba implementovat možnost pohybu v seznamu dle abecedy. Pro tyto účely je potřeba zdroje seřadit podle abecedy a vytvořit seznam jednotlivých písmen abecedy. Následně je nutno vypočítat pozice prvních výskytů jednotlivých počátečních písmen u zdrojů a přiřadit je k již vytvořenému seznamu písmen v abecedě.

Další funkcionalitou, kterou je třeba implementovat, je filtrování zdrojů. Každý zdroj ve své JSON dokumentaci bude mít definované tzv. tagy, dle kterých bude možno vytvořit systém pro filtrování.

S filtry musí být kompatibilní i vyhledávání pomocí textového inputu.

Druhým zobrazením v aplikaci je detail samotného EIZ. Na tomto zobrazení budou zobrazeny všechny informace, které přijdou ze serveru. Téměř všechny položky u zdroje jsou volitelné, může se tedy stát, že o zdroji nebudou vyplněny žádné informace a v tom případě je třeba skrývat všechny nevyplněné pole.

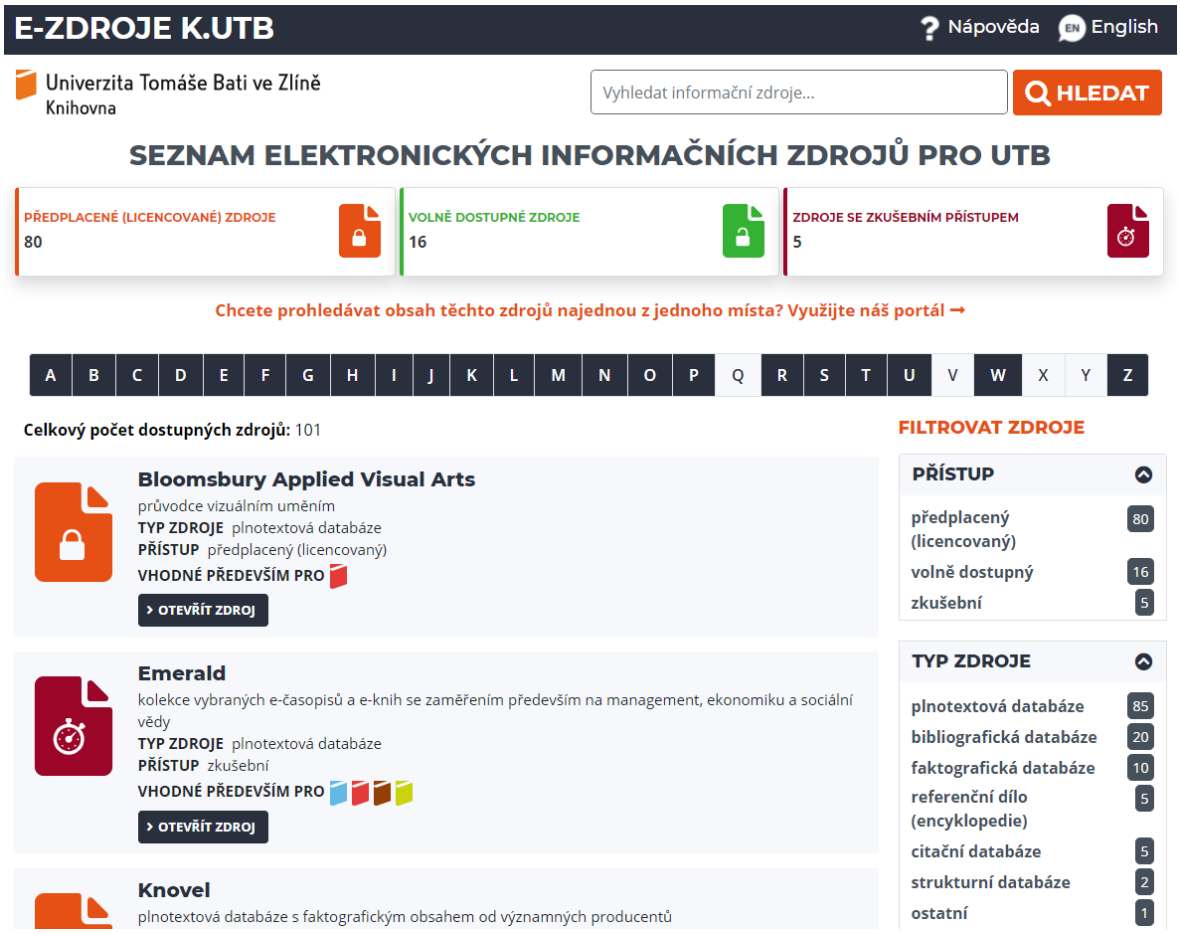
Posledním požadavkem je implementace ve dvou jazycích – českém a anglickém. Mezi těmito jazyky bude možno přepínat pomocí jednoho tlačítka.

7 NÁVRH ŘEŠENÍ

Jak už bylo zmíněno v rámci požadavků – je nutno vytvořit frontend webové aplikace. Hlavní komunikace se serverem bude probíhat pouze v rámci jednoho endpointu, který bude vracet všechna data najednou. Pro tyto potřeby se hodí využít modelu SPA.

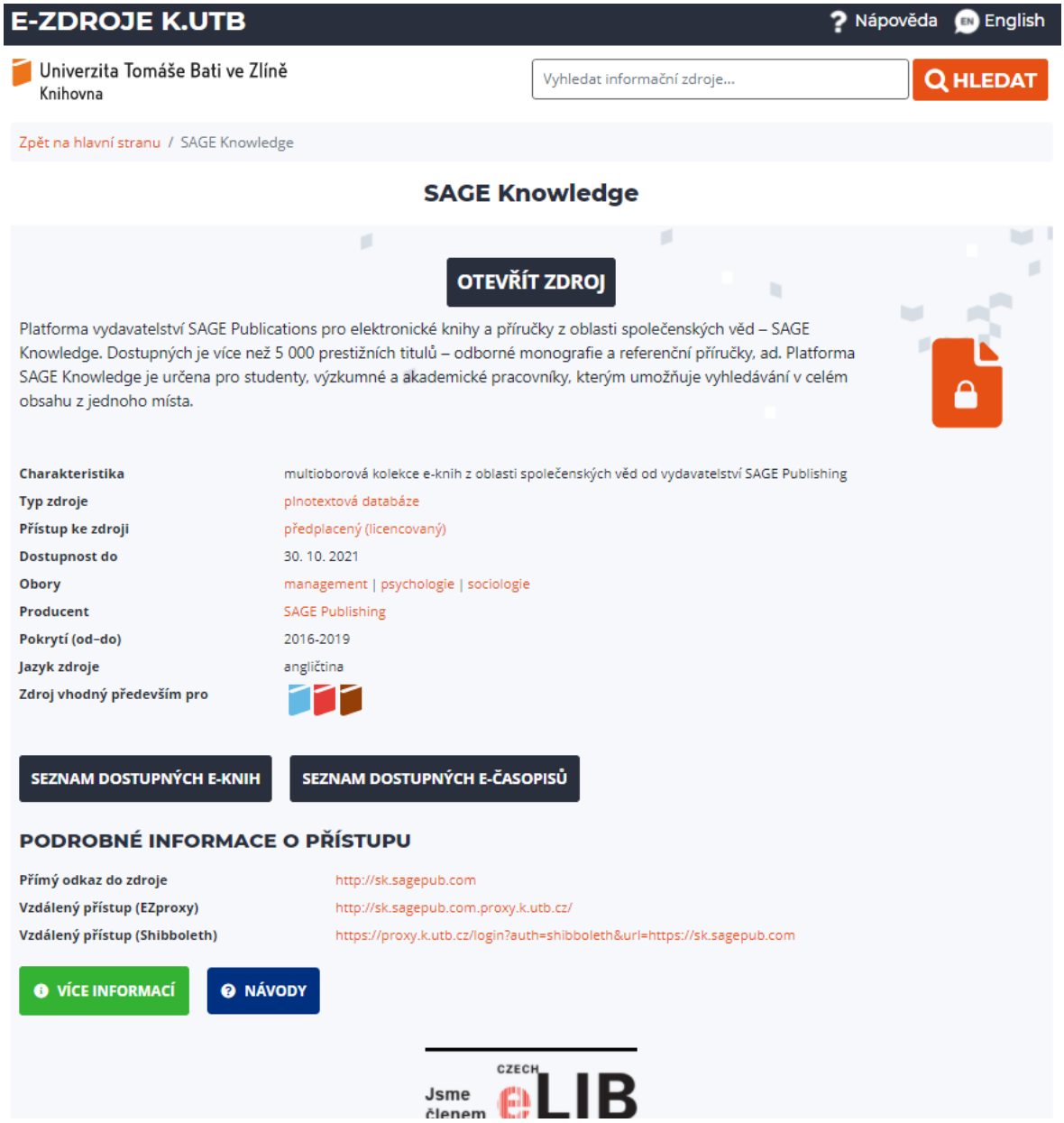
7.1 Grafický návrh

Aplikace je tvořena ve spolupráci s knihovnou UTB a po jejím dokončení bude vystavena pro veřejnost. Grafické zobrazení tedy musí ladit s ostatními weby UTB. O grafický návrh webové aplikace v podobě statického HTML a CSS se postarala Mgr. Světlana Hrabinová Ph.D.



Obr. 13 Grafický návrh aplikace – hlavní strana

Obr. 13 zobrazuje návrh hlavní strany aplikace. Na návrhu lze vidět prakticky všechny požadavky z předchozí kapitoly. Odshora lze postupně vidět tlačítko pro přepínání jazyků, vyhledávací pole, abecední navigaci, kartičky se zdroji a filtry podle jednotlivých tagů.



Obr. 14 Grafický návrh aplikace – detail EIZ

Obr. 14 znázorňuje návrh detailu EIZ. Hlavička stránky se nemění a funkce jako vyhledávání a přepínání jazyka tedy musí fungovat i u této strany aplikace. V detailu jsou vidět všechna pole, která budou přístupná z databáze.

7.2 Návrh technologií

Pro zpracování tohoto řešení jsem v teoretické části provedl průzkum možných návrhových modelů a technologií.

Pro zpracování tohoto zadání je určitě vhodnější použít model SPA. Aplikace bude obsahovat velmi malé množství unikátních stran a nebude tedy nutno manuálně řešit různé požadavky pro každou stránku v aplikaci zvlášť. Hlavní funkcionalitou aplikace bude možnost filtrování a prohledávání zdrojů – k tomuto se SPA přímo nabízí, protože je v tomto modelu velmi jednoduché aplikovat jednotlivé změny v UI bez neustálého refreshování.

V teoretické části jsem uvedl tři aktuálně nejpoužívanější možnosti vývoje SPA. Pro tento projekt jsem se rozhodl použít framework Angular z následujících důvodů:

- Z prozkoumaných možností se jedná o jedinou technologii, se kterou lze vytvořit kompletní aplikaci i bez použití jakýchkoliv komunitních nástrojů.
- Vývoj v Angularu se drží poměrně striktních pravidel. Mně osobně tento způsob vývoje vyhovuje, protože do jisté míry eliminuje možné problémy způsobené špatným návrhem od vývojáře.
- Celý framework stojí na TypeScriptu. Osobně preferuji TypeScript před JavaScriptem, protože jsem zvyklý na práci s jazyky používající statické typování. Ostatní možnosti sice také nabízí využití TypeScriptu, ale Angular je jediný framework, který TypeScript vyžaduje a je k němu kompletně přizpůsoben.

8 IMPLEMENTACE

V této kapitole se budu věnovat implementaci řešení pomocí Angularu dle požadavků specifikovaných v kapitole 6.2 - Požadavky na nové řešení.

8.1 Zpracování dat z API

Seznam EIZ je dostupný na URL *https://www-new.k.utb.cz/eresources-list.php*. Každý zdroj je reprezentován 26 poli a uvnitř aplikace se mapuje na třídu *Resource*.

```
export class Resource {
  public source_id: string;
  public active: boolean;
  public active_in_proxy: boolean;
  public type: string;
  public title_display: string;
  public publisher: string;
  public link_publisher: string;
  public description_cs: string;
  public description_en: string;
  public description_short_cs: string;
  public description_short_en: string;
  public link_native_home: string;
  public link_via_proxy: string;
  public link_guide: string;
  public link_more_info: string;
  public link_ebooks: string;
  public link_journals: string;
  public via_czechelib: boolean;
  public ezproxy_end_of_trial: string;
  public wayfless: string;
  public facet_access: string;
  public facet_access_map: Map<string, object>;
  public facet_resource_type: string;
  public facet_resource_type_tags_map: Map<string, object>;
  public facet_content_type: string;
  public facet_content_type_tags_map: Map<string, object>;
  public facet_faculty: string;
  public facet_faculty_tags_map: Map<string, object>;
  public facet_discipline: string;
  public facet_discipline_tags_map: Map<string, object>;
  public facet_language: string;
  public facet_language_map: Map<string, object>;
}
```

Kód 3 Třída *Resource*

Kód 3 reprezentuje třídu *Resource*. Ne všechny položky jsou však pouhou reprezentací dat přicházejících z API. Pole *link_via_proxy* na řádku 14 slouží pro vytvoření odkazu pro přístup z externí sítě. Pole na řádcích 23, 25, 27, 29, 31 a 33 slouží pro uchování jednotlivých tagů z kapitoly 5 - EIZ používané v Knihovně UTB. Plnění těchto polí je popsáno níže.

Seznam zdrojů si aplikace vyžádá vždy při jejím prvotním načtení. O veškerou správu zdrojů se stará *ResourceService*, který je pomocí DI vkládán do mnoha komponentů napříč aplikací. Hlavním komponentem, který zabaluje celou aplikaci je *AppComponent*, který si pomocí DI vyžádá *ResourceService* a nastartuje celý proces.

Samotný *ResourceService* ve svém konstruktoru volá metodu *getResources*, která zajišťuje prvotní správu zdrojů. Všechna data přicházející z API se ukládají do proměnné *_resources*. K zobrazení aktuálně vyfiltrovaných dat slouží *_filteredResources*.

```
109   private getResources(): void {
110     if (!this._resources) {
111       this._resources = [];
112       this._filteredResources = [];
113       this.http
114         .get<Resource[]>('https://www-new.k.utb.cz/eresources-list.php')
115         .subscribe((res) => {
116           for (const r of res) {
117             if (!r.active) {
118               continue;
119             }
120             r.link_via_proxy = `https://proxy.k.utb.cz/login?url=${r.link_native_home}`;
121             r.facet_content_type_tags_map = this.createFacetTagsMap(···
124             );
125             r.facet_discipline_tags_map = this.createFacetTagsMap(···
128             );
129             r.facet_resource_type_tags_map = this.createFacetTagsMap(···
132             );
133             r.facet_access_map = this.createFacetTagsMap(···
136             );
137             r.facet_language_map = this.createFacetTagsMap(···
140             );
141             r.facet_faculty_tags_map = this.createFacetTagsMap(···
144             );
145
146             this._resources.push(r);
147           }
148           if (this._activeFilters.length === 0) {
149             this._filteredResources.push(...this._resources);
150           } else {
151             this.filterResources();
152           }
153           this.setupLetters();
154           this.subject.next('changed');
155         });
156     }
157   }
```

Kód 4 Metoda *getResources*

Metodu *getResources* ukazuje kód 4. K uchování všech zdrojů slouží pole *_resources*. Metoda *getResources* se volá v konstruktoru, který svou logiku provede pokaždé, když je service vyžádána nějakým komponentem. Aby se zabránilo neustálému volání API a vykonávání logiky pro zpracování zdrojů, nejdříve se provede podmínka na řádku 110. Pole *_resources* je *undefined* pouze při prvotním načtení aplikace a podmínka tak bude pravdivá pouze jedenkrát.

S využitím integrovaného HTTP klienta se vytvoří dotaz na API s očekáváním příchozích dat. Tato data se zpracují uvnitř metody *subscribe* na řádku 115. Samotný HTTP klient již vytvoří instanci třídy *Resource* pro každou položku. Již dříve bylo zmíněno, že ve třídě *Resource* jsou některá pole, která je třeba naplnit uvnitř aplikace. Právě k tomu slouží cyklus *for* začínající na řádku 116. Nejdříve se vynechají všechny zdroje, které nejsou aktivní, a následně je třeba zpracovat pole s jednotlivými tagy.

Kategorie jednotlivých tagů byly rozebrány v kapitole 6.2 - Požadavky na nové řešení. V reálném provedení jsou tagy v jednotlivých kategoriích rozděleny pomocí středníku a je třeba je rozdělit na jednotlivé hodnoty.

```
"facet_content_type": "e-knihy;e-časopisy;konferenční materiály"
```

Obr. 15 Příklad tagů

```
159   private createFacetTagsMap(
160     facet: string,
161     type: FilterType
162   ): Map<string, object> {
163     if (!facet) {
164       return new Map<string, object>();
165     }
166     const facets = facet.split(';');
167     const map = new Map<string, object>();
168     for (let f of facets) {
169       const en = this.getEnglishFacetName(f, type);
170       map.set(f, { cs: f, en: en });
171     }
172     return map;
173   }
```

Kód 5 Metoda *createFacetTagsMap*

V kódu 5 lze vidět metodu, která slouží pro zpracování jednotlivých tagů. V metodě *getResources* (viz kód 4) se tato metoda volá pro každý druh tagu – řádky 121, 125, 129, 133, 137 a 141.

Na řádku 166 lze vidět použití funkce *split*, která v tomto případě rozdělí jednu hodnotu typu *string* (viz Obr. 15) na pole hodnot typu *string*. Každý tag je reprezentován objektem *Map<string, object>*. Tento objekt umožňuje uložit všechny tagy ve formátu klíče a hodnoty. Hodnota je v tomto případě ve formátu objektu a umožní tak uložení hned několika hodnot najednou – českého a anglického názvu tagu.

8.2 Reaktivita

Důležitým konceptem, kterého jsem v této aplikaci využil, je reaktivita. Data jsou uchovávána mimo komponenty a tak by bez použití reaktivního programování nebylo možné promítat změny, ke kterým došlo např. uvnitř *ResourceService* přímo do zobrazovací vrstvy aplikace. Právě tento problém reaktivita řeší.

K dosažení reaktivity uvnitř aplikace byla použita knihovna RxJS. Konkrétně objekt typu *Subject*.

Příkladem reaktivity je zobrazování pole EIZ na hlavní straně. Nejdříve byla definována proměnná tohoto typu uvnitř *ResourceService* – viz kód 6.

```
22    subject = new Subject<any>();
```

Kód 6 Definice objektu typu *Subject*

Použití tohoto objektu lze pozorovat například v kódu 4 na řádku 154. Tímto řádkem jsou informovány všechny části aplikace, ve kterých byla zavolána metoda *subscribe* vůči tomuto objektu, že došlo ke změnám.

```
48        this.resourceService.subject.subscribe(() => {
49          this.updateResources();
50        })
```

Kód 7 Příklad využití reaktivního programování

V kódu 7 lze sledovat, jakým způsobem lze použít metodu *subscribe.* Vše co je definováno uvnitř *callbacku* této metody (řádek 49) se provede vždy, když bude předána informace o provedených změnách. V tomto případě se bude vždy volat metoda *updateResources,* ve které dojde k aktualizaci pole vyfiltrovaných EIZ.

8.3 Filtrování EIZ

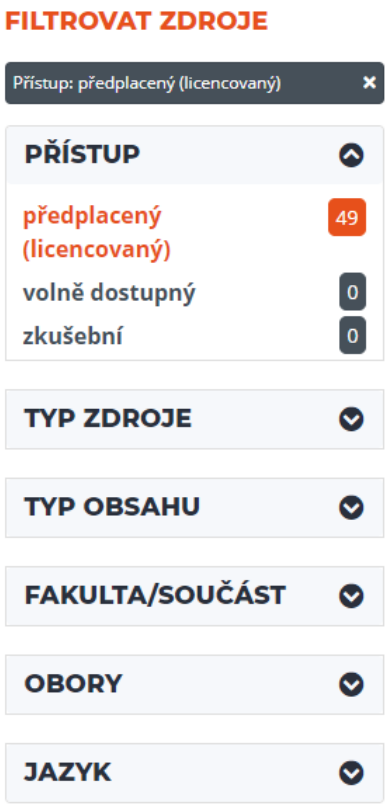
Hlavní funkcionalitou uvnitř webové aplikace je možnost vyfiltrovat si EIZ dle jednotlivých tagů.

8.3.1 Filtrovací komponenty

Filtrování je z uživatelského pohledu implementováno ve formě zvolitelných možností v seznamu tagů. Každý typ tagu je zpracován do komponentu zvlášť. Komponenty slouží pouze jako uživatelská vrstva, která předává informace o aktivaci jednotlivých filtrů do *ResourceService*.

FacetsFilterComponent

Jedná se o agregační komponent, který shlukuje všechny komponenty, které slouží pro filtrování dle tagů. Zároveň slouží k zobrazování aktuálně aktivních filtrů.

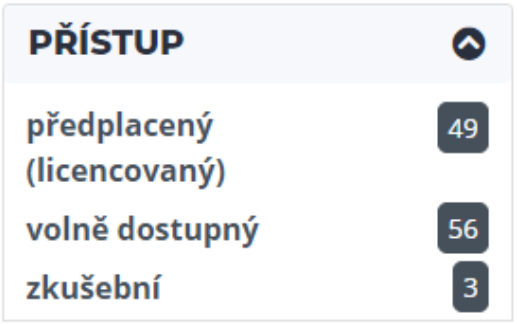


Obr. 16 *FacetsFilterComponent* s jedním aktivním filtrem

Obr. 16 zobrazuje všechny kategorie filtrů v aplikaci. Zároveň lze vidět příklad jednoho aktivního filtru.

Komponenty pro jednotlivé tagy vypadají velmi podobně a všechny plní stejnou funkcionalitu s velmi malými rozdíly. Pro příklad je uvedena implementace pouze jednoho z nich.

AccessFilterComponent



Obr. 17 *AccessFilterComponent*

Každý z těchto komponentů se skládá ze tří viditelných prvků.

- **Nadpis** – staticky vepsaný do HTML.
- **Název tagu** – každý typ tagu je definován jako pole uvnitř JavaScriptové konstanty. Toto pole je následně pomocí direktivy *NgFor* vykresleno a je k němu implementována další logika.
- **Počet EIZ, které danému tagu odpovídají** – toto číslo je vypočítáno pomocí metody *getAccessCount* znázorněné v Kód 8. Metoda provede iteraci přes všechny EIZ a vrátí počet nalezených shod s daným tagem. Tato metoda se volá pro každý tag v seznamu.

```
26    getAccessCount(access: string): number {
27      let count = 0;
28      for (let r of this.resources) {
29        if (r.facet_access === access) {
30          count++;
31        }
32      }
33      return count;
34    }
```

Kód 8 Metoda *getAccessCount*

Funkcionalita komponentů spočívá ve schopnosti aktivovat a deaktivovat jednotlivé filtry. Uživatel tuto funkcionalitu aktivuje kliknutím na název tagu. Po aktivaci filtru proběhne změna barvy tagu na oranžovou. Pokud již je filtr aktivní, tak dojde k deaktivaci filtru. Na každém tagu je napojeno volání metody uvnitř komponentu. Tato metoda uvnitř samotného komponentu neprovede žádné změny, ale pouze předá informaci o kliknutí na daný filtr do *ResourceService*, kde dojde k aktivaci nebo deaktivaci filtru dle aktuálního stavu. Konkrétně se zde volá metoda *setActiveFilter*, jejíž funkce bude popsána v dalších kapitolách.

8.3.2 Implementace datové vrstvy filtrů

Komponenty slouží v tomto případě pouze jako vizuální a interakční vrstva, samotná filtrovací logika je prováděna v *ResourceService* a jednotlivé komponenty jsou o změnách informovány reaktivním způsobem popsaným v kapitole 8.2 - Reaktivita. Zde je ve všech komponentech, jejichž vizuální stránka je závislá na stavu filtrů, nutné provést metodu *subscribe* objektu *filtersSubject*.

Pro reprezentaci aktivního filtru byla vytvořena třída *ActiveFilter* se dvěma vlastnostmi. Tato třída je znázorněna v kódu 9. První vlastnost je typu *FilterType* což je typ *enum* reprezentující jednotlivé druhy tagů. Tento typ je důležitý pro rozdělení dat, ve kterých je tagy nutno při filtrování vyhledávat. Vlastnost *value* je samotný název tagu, který se následně porovnává s tagy na EIZ.

```
3  export class ActiveFilter {
4    public filterType: FilterType;
5    public value: string;
6
7    constructor(filterType: FilterType, value: string) {
8      this.filterType = filterType;
9      this.value = value;
10   }
11 }
```

Kód 9 Třída *ActiveFilter*

Uvnitř *ResourceService* je s filtry spojeno velké množství kódu. Všechny aktivní filtry jsou ukládány v proměnné *_activeFilters* typu *ActiveFilter[]*.

```
67    setActiveFilter(filter: ActiveFilter): void {
68      const foundFilterIndex = this._activeFilters.findIndex(
69        (x) => x.filterType === filter.filterType && x.value === filter.value
70      );
71
72      if (foundFilterIndex !== -1) {
73        this._activeFilters.splice(foundFilterIndex, 1);
74      } else {
75        this._activeFilters.push(filter);
76      }
77      this.filtersSubject.next('changed');
78    }
```

Kód 10 Metoda *setActiveFilter*

Kliknutí na jednotlivé tagy uvnitř komponentů s filtry je napojeno na metodu *setActiveFilter* (viz kód 10). Tato metoda nejdříve zkontroluje, zda již daný filtr není aktivován. V případě, že filtr je již aktivní, tak dojde k jeho odstranění z pole aktivních filtrů, jak je vidět na řádku 73. Naopak, pokud filtr aktivní není, tak dojde k jeho aktivaci přidáním do pole aktivních filtrů. Až je proces hotový, tak dojde k oznámením o změnách pomocí *filtersSubject* (řádek 77).

Na změnu v aktivovaných filtrech je napojena metoda, která se stará o samotné filtrování EIZ. Při každé změně filtru se provede cyklus pro *for* pro všechny EIZ. Pro každý EIZ se následně provede iterace přes všechny aktivní filtry. Filtrování je založeno na principu průniku. Aby byl EIZ vyfiltrován je nutno aby splňoval podmínky všech filtrů – tedy aby u něj byly přítomné všechny tagy, podle kterých se aktuálně filtruje.

8.3.3 Implementace filtrů do URL

Aplikace nabízí i možnost přenosu aktivních filtrů pomocí URL. Pokud by chtěl uživatel například vyhledat EIZ vhodné pro fakultu FAI a obsahující obsah typu e-knihy, mohl by aktivovat odpovídající filtry a aplikace tyto parametry automaticky přidá do URL:

http://ezdroje.k.utb.cz/?contentType=e-knihy&faculty=FAI

Pro tyto účely byl vytvořen *URLService*, který slouží k interakci aplikace s URL. Konkrétně se pro zápis jednotlivých filtrů do URL používá metoda *appendFilterParamsToUrl*.

```
19  appendFilterParamsToUrl(): void {
20    let queryParams = {};
21    for (const filter of this.resourceService.activeFilters) {
22      const type = this.getFilterStringByType(filter.filterType);
23      queryParams[type] = filter.value;
24    }
25    const url = this.router
26      .createUrlTree([], {
27        relativeTo: this.activatedRoute,
28        queryParams: queryParams,
29      })
30      .toString();
31    this.location.go(url);
32  }
```

Kód 11 Metoda *appendFilterParamsToUrl*

Pro interakci s URL aplikace je nutno použít dva objekty, které poskytuje samotný framework. Jedná se o *Router* a *Location*. Kód 11 znázorňuje použití těchto objektů. Metoda *createUrlTree* umožňuje vytvořit URL s použitím query parametrů, které jsou vytvořeny z aktivních filtrů. Následně už se tato URL jenom použije. Objekt typu *Location* slouží k manipulaci URL, aniž by došlo k samotné navigaci. Tímto je možno přepsat URL při každé změně filtru, aniž by byla aktivována jakákoliv navigační logika (řádek 31).

```
76    this.activatedRoute.queryParams.subscribe((params) => {
77      for (const p in params) {
78        const type = this.getFilterTypeString(p);
79        const filter = new ActiveFilter(type, params[p]);
80        this.resourceService.setActiveFilter(filter);
81      }
82    });
```

Kód 12 Aktivace filtrů z URL

Kód 12 ukazuje způsob aktivace filtrů, které byly předány jako parametry v URL. Parametry jsou uloženy ve vlastnosti *queryParams* v objektu typu *ActivatedRoute*, který je s pomocí DI vyžádán. Pro každý parametr je poté získán jeho typ, viz řádek 78, a vytvořen nový objekt typu *ActiveFilter*. Metodou *setActiveFilter*, která je popsána v kapitole 8.3.2 - Implementace datové vrstvy filtrů, následně je filtr předán k aktivaci.

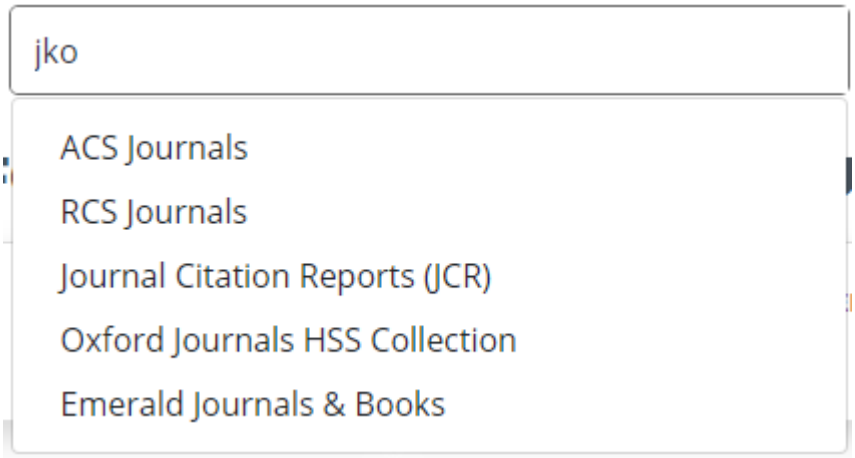
8.4 Vyhledávání

Vyhledávání EIZ uvnitř aplikace bylo implementováno pomocí textového inputu s vlastním automatickým doplňováním textu s tolerancí překlepu. Komponent v neaktivním stavu lze vidět na Obr. 18.

Obr. 18 *SearchInputComponent* - komponent pro hledání EIZ

8.4.1 Automatické doplňování

Samotný input je implementován jako klasický HTML input. K inputu navíc byla přidána rozbalovací nabídka, ve které se zobrazuje 5 položek s nejlepší shodou vůči vloženému textu.



Obr. 19 Automatické doplňování s tolerancí překlepů

Na Obr. 19 lze vidět, jak funguje rozbalovací nabídka s tolerancí překlepu. I přesto, že byl zadán text „jko“, kterému žádný EIZ neodpovídá, tak je zobrazena nabídka 5 EIZ s nejlepší shodou. K dosažení této funkcionality byla použita knihovna Fuse.js.

Knihovna Fuse.js poskytuje metodu *search*, která prohledá pole poskytnutých hodnot – v tomto případě seznam EIZ a ke každé hodnotě vypočítá úroveň shody. Tyto hodnoty následně seřadí a vrátí je zpátky. Tato metoda byla použita uvnitř metody *onSearchInput*. *OnSearchInput* se volá při každé změně hodnoty v inputu. Na řádku 63 v kódu 13 lze vidět, že z výsledku metody *search*, kterou poskytuje Fuse.js, pomocí metody *slice* je použito prvních 5 hodnot a ty jsou uloženy do proměnné *resourcesAutocomplete*. Tato proměnná reprezentuje pole hodnot, které se následně zobrazují v rozbalovací nabídce v HTML.

```
60    onSearchInput(value: string): void {
61      if (value.length > 0) {
62        const fuse = new Fuse(this.resources, this.searchOptions);
63        this.resourcesAutocomplete = fuse.search(value).slice(0, 5);
64      }
65      this.searchText = value;
66    }
```

Kód 13 Metoda *onSearchInput*

Na položky v rozbalovací nabídce lze klikat. Kliknutí na danou položku aktivuje navigaci na detail daného EIZ:

8.4.2 Aplikace vyhledávacího textu

Druhá část komponentu souvisí s tlačítkem „HLEDAT“. Po kliknutí na toto tlačítko nebo po zmáčknutí tlačítka enter na klávesnici, se vyhledávaný text aplikuje ve formě filtru. Implementace filtru je popsána v kapitole 8.3.2 - Implementace datové vrstvy filtrů. Po aktivaci se tedy, stejně jako filtrovací tagy, vyhledávaný text přidá do pole *_activeFilters* a váže se na něj stejná funkcionalita jako na ostatní filtry. Vizuálně lze poznat, že je vyhledávání aktivní, podle přidaného pole nad seznamem typů filtrů, viz Obr. 20.

Obr. 20 Vizuální zobrazení aktivního filtru

8.5 Navigace v seznamu

Uživatel si může zredukovat počet položek v seznamu EIZ s pomocí filtrů a vyhledávání. Další možností jak se v seznamu pohybovat je využití abecední navigace. Tato navigace je implementována v podobě písmen zobrazených na hlavní straně aplikace nad seznamem EIZ a slouží k navigaci na první výskyt daného písmena na začátku názvu daného EIZ.

Obr. 21 Abecední navigace

Písmena s černým pozadím označují aktivní položku – existuje EIZ začínající na toto písmeno.

Implementace se skládá z několika částí. Zobrazování aktivních a neaktivních písmen je navázáno na *ResourceService*, ve kterém se při každé změně filtrů provede aktualizace aktivních písmen. Datová implementace těchto písmen se skládá z informace o aktivitě/neaktivitě a také indexu prvního výskytu písmena v seznamu EIZ.

Další část se týká elementů kartiček, ze kterých je sestaven seznam EIZ. Každá tato kartička má přiřazenu vlastnost *id* s hodnotou indexu, na kterém se aktuálně nachází v seznamu zdrojů. Tímto způsobem je ke každému písmenu přiřazen index výskytu a každá kartička má svůj index uložen ve svém HTML. Kliknutím na písmeno se už jenom spustí mechanismus, který v DOM najde element s žádaným indexem, a přejde na něj.

8.6 Rozpoznání sítě

Aplikace rozeznává IP adresu sítě, ze které se uživatel připojuje. To je umožněno díky externí službě, která dokáže IP adresu klienta rozpoznat.

Z praktického hlediska se jedná pouze o jednouchý HTTP dotaz, který se provede při prvním načtení aplikace (podobně jako získání pole EIZ). Tento dotaz vrací IP adresu klienta. Adresa se následně porovná s rozsahem povolených IP adres dodaných Knihovnou UTB:

- 195.178.88.0 až 195.178.95.255
- 195.113.96.0 až 195.113.99.255
- 10.0.0.1 až 10.255.255.254

Veškerá logika se děje uvnitř *IPAdressService*, která je pomocí DI dostupná uvnitř celé aplikace. Pokud IP adresa zapadá do některého z výše stanovených rozsahů, tak je identifikována jako přístup z interní sítě UTB a všechny EIZ jsou tak zpřístupněny pomocí přímých odkazů. Avšak opačný scénář způsobí nutnost zpřístupnění pomocí Shibbolethu nebo EZproxy. Prioritu má v tomto případu Shibboleth, odkaz do přes EZproxy je nabídnut pouze v případě nepřítomnosti odkazu přes Shibboleth.

8.7 Překlad

Aplikace je dostupná ve dvou jazycích – českém a anglickém. Jazyky lze přepínat pomocí tlačítka v hlavičce aplikace.

Obr. 22 Tlačítko přepnutí jazyka aplikace do angličtiny

Tlačítko na Obr. 22 značí, že aktuální jazyk aplikace je český a tlačítkem se přepne do angličtiny. Pokud bude aplikace v anglickém jazyce, tak se změní i tlačítko.

Změna jazyka napříč celou aplikací opět využívá reaktivity, popsané v kapitole 8.2 - Reaktivita. Změna jednotlivých textů byla vyřešena pomocí direktivy *ngIf*. Každá textová položka v aplikaci je v HTML zapsána dvakrát – jednou pro každý jazyk.

8.7.1 Překlad statických hodnot

```
35        <li class="breadcrumb-item active">
36          <span *ngIf="currentLanguage === 'cs'">Filtr pro e-zdroje</span>
37          <span *ngIf="currentLanguage === 'en'">Filter for e-resources</span>
38        </li>
```

Kód 14 Ukázka implementace překladu v HTML

V kódu 14 na řádcích 36 a 37 byly implementovány dva překlady pro jednu statickou položku v HTML. Díky podmínkám v *ngIf* je zajištěno, že vždy bude zobrazena pouze jedna verze. Aktuální jazykové nastavení je uloženo v proměnné *currentLanguage* a může být nastaveno na hodnotu *cs* pro český překlad a hodnotu *en* pro anglický překlad. Tato proměnná je přepisována reaktivním způsobem a bude tedy vždy nastavena na aktuální hodnotu.

8.7.2 Překlad dynamických hodnot

Dynamické položky závislé na datech z API jsou řešeny dvěma způsoby. U některých jsou přímo z API dostupné obě verze překladu a není problém je jednoduše zobrazit, viz kód 15.

```
15        <p class="result-item">
16          <span *ngIf="currentLanguage === 'cs'">{{ resource.description_short_cs }}</span>
17          <span *ngIf="currentLanguage === 'en'">{{ resource.description_short_en }}</span>
18        </p>
```

Kód 15 Překlad dynamických dat

Problém nastává u tagů. Tagy chodí ve formátu textového řetězce v českém jazyce. Při zpracování tagů je potřeba napárovat jednotlivé hodnoty na jejich anglický překlad. Pro tyto potřeby bylo potřeba uvnitř aplikace manuálně vytvořit pole s jednotlivými překlady pro všechny tagy, viz kód 16.

```
1   export const accesses = [
2     { cs: 'předplacený (licencovaný)', en: 'licensed' },
3     { cs: 'volně dostupný', en: 'free' },
4     { cs: 'zkušební', en: 'trial' },
5   ];
```

Kód 16 Definice překladu tagů

Již v kódu 5 lze vidět vytváření objektu *Map* pro reprezentaci tagů. Využití objektu tohoto typu bylo učiněno, protože umožňuje reprezentovat objekt ve formátu klíče a hodnoty. Do klíče je uložena česká varianta překladu. Hodnota je v tomto případě opět ve formě objektu a ten je tvořen právě vlastnostmi *cs* a *en* pro reprezentaci českého a anglického překladu tagu.

Pole, zobrazené v kódu 16, se navíc používá i pro další účely, například zobrazení jednotlivých tagů ve filtrovacích komponentech.

8.7.3 Cachování

V základním stavu je aplikace spuštěna v českém jazyce. Při změně jazyka bylo implementováno ukládání aktuální hodnoty do *localStorage*. *LocalStorage* je paměť uvnitř prohlížeče, která může sloužit u ukládání menších objemů dat.

```
28    changeLanguage(language: Language): void {
29      this.languageService.language = language;
30      localStorage.setItem('language', language);
31    }
```

Kód 17 Metoda *changeLanguage*

K uložení hodnoty je použita metoda *setItem*, která přijímá jako první argument název, pod kterým chci hodnotu uložit a jako druhý argument přijímá samotná data. Metoda *changeLanguage* (viz kód 17) se volá po zmáčknutí tlačítka pro změnu jazyka aplikace.

Uvnitř *AppComponent* se při startu aplikace zase provádí opačný proces. Aplikace se z *localStorage* pokusí získat hodnotu *language*.

```
69      const language = localStorage.getItem('language');
70      if (language) {
71        this.languageService.language = language as Language;
72      }
```

Kód 18 Získání hodnoty překladu z *localStorage*

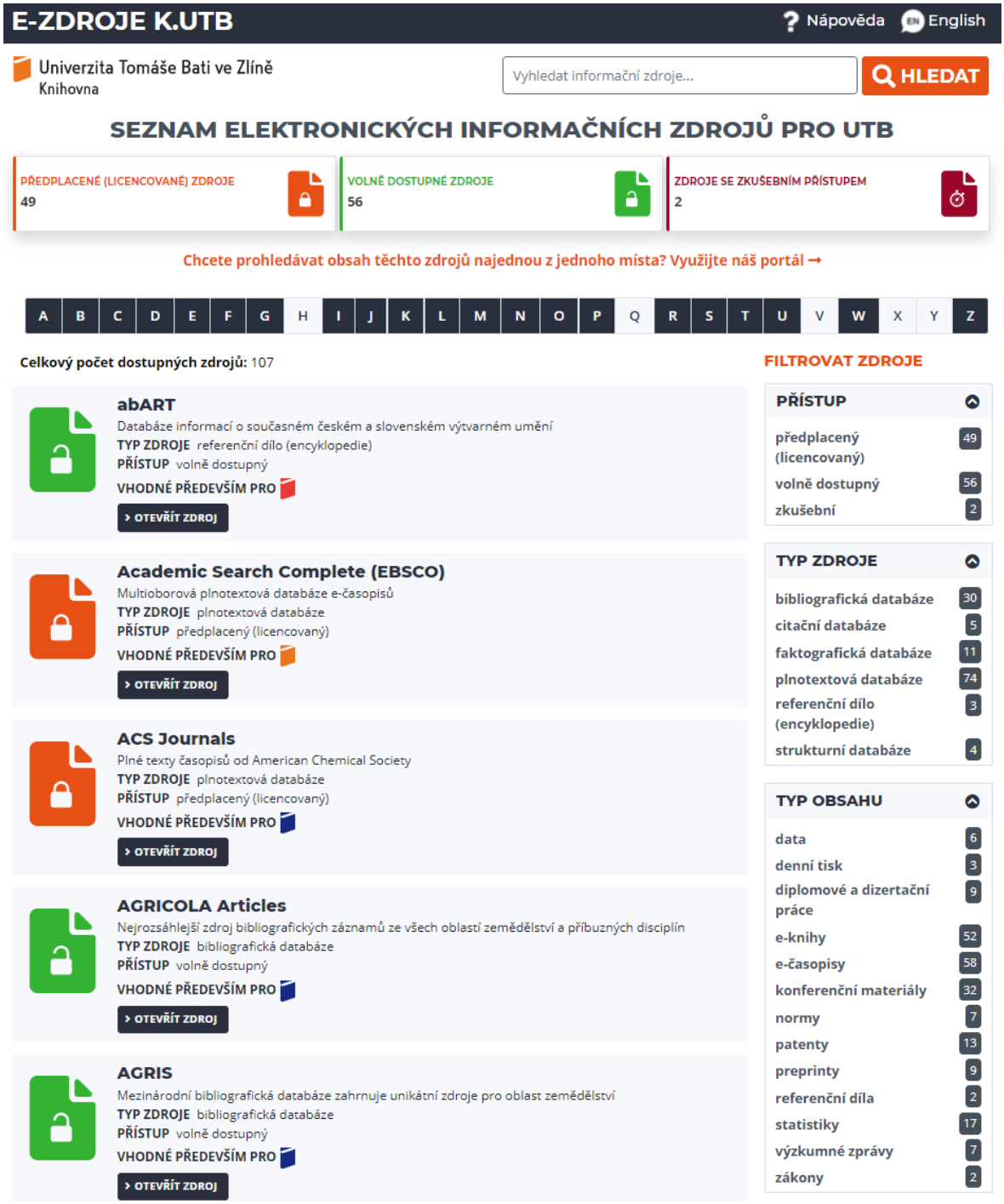
Hodnota překladu se v *localStorage* objeví až po změně jazyka, kterou musí vyvolat sám uživatel. *LocalStorage* je tedy nejdříve prázdný. V kódu 18 je vidět, že k přepsání hodnoty *language*, která se nachází uvnitř *LanguageService,* dojde pouze, pokud v *localStorage* již nějaká hodnota existuje. Pokud k žádnému přepsání nedojde, tak zůstane hodnota na původní - české.

9 VÝSLEDNÁ APLIKACE

Výsledná aplikace se skládá ze dvou hlavních částí – hlavní strany a detailu EIZ a z jedné doplňkové části v podobě strany s nápovědou.

9.1 Hlavní strana

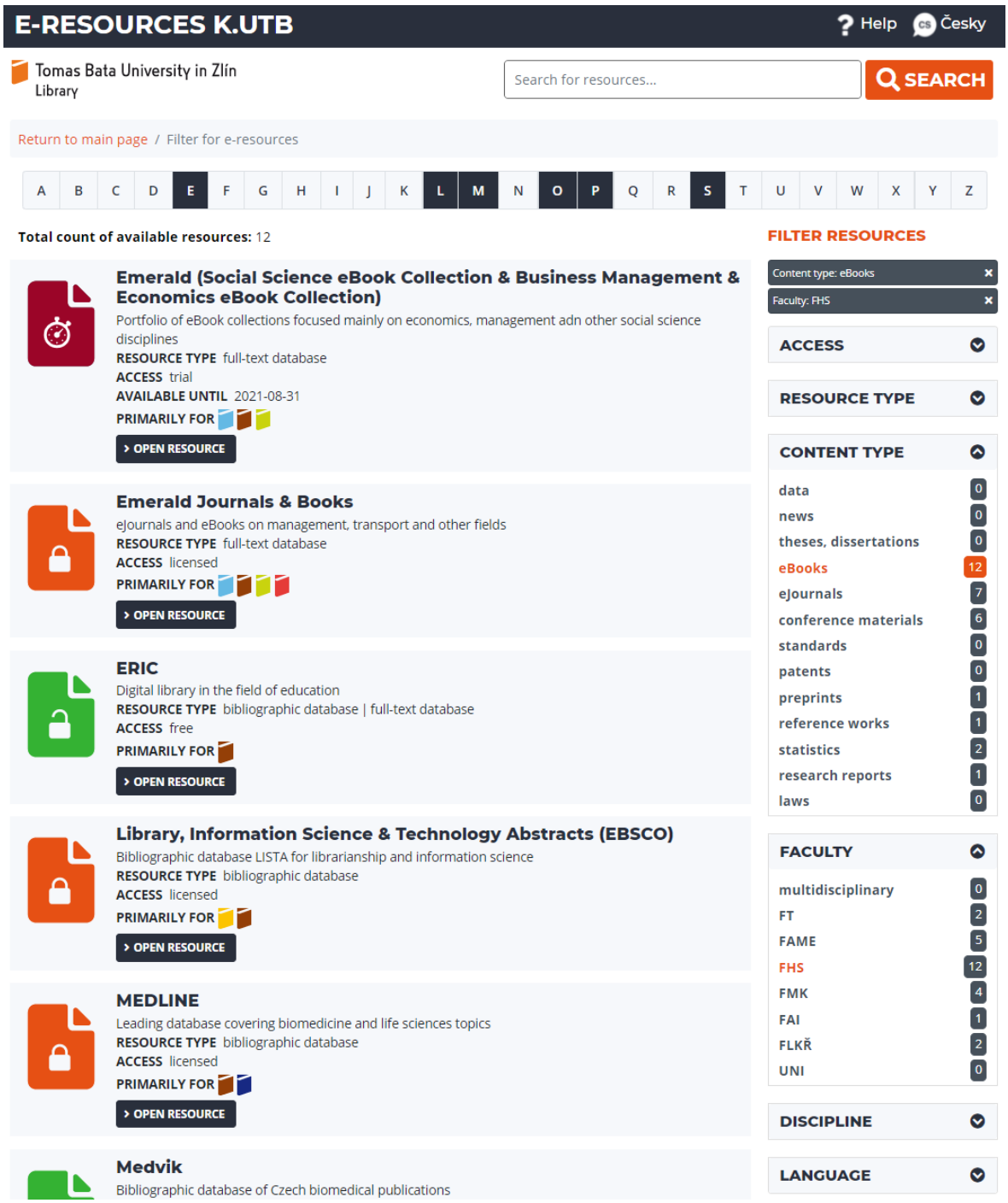
Hlavní strana aplikace má dvě lehce rozdílné varianty.



Obr. 23 Hlavní strana aplikace

Na Obr. 23 vidíme hlavní stranu ve stavu, ve kterém ji uživatel uvidí při spuštění aplikace.

Druhá varianta se aktivuje při aktivaci některého z filtrů. Horní část aplikace s nadpisem a filtry podle přístupu je nahrazena tlačítkem pro zrušení filtrů a navrácení se k původnímu stavu. Obr. 24 ukazuje tuto variantu zároveň i s anglickou verzí aplikace.

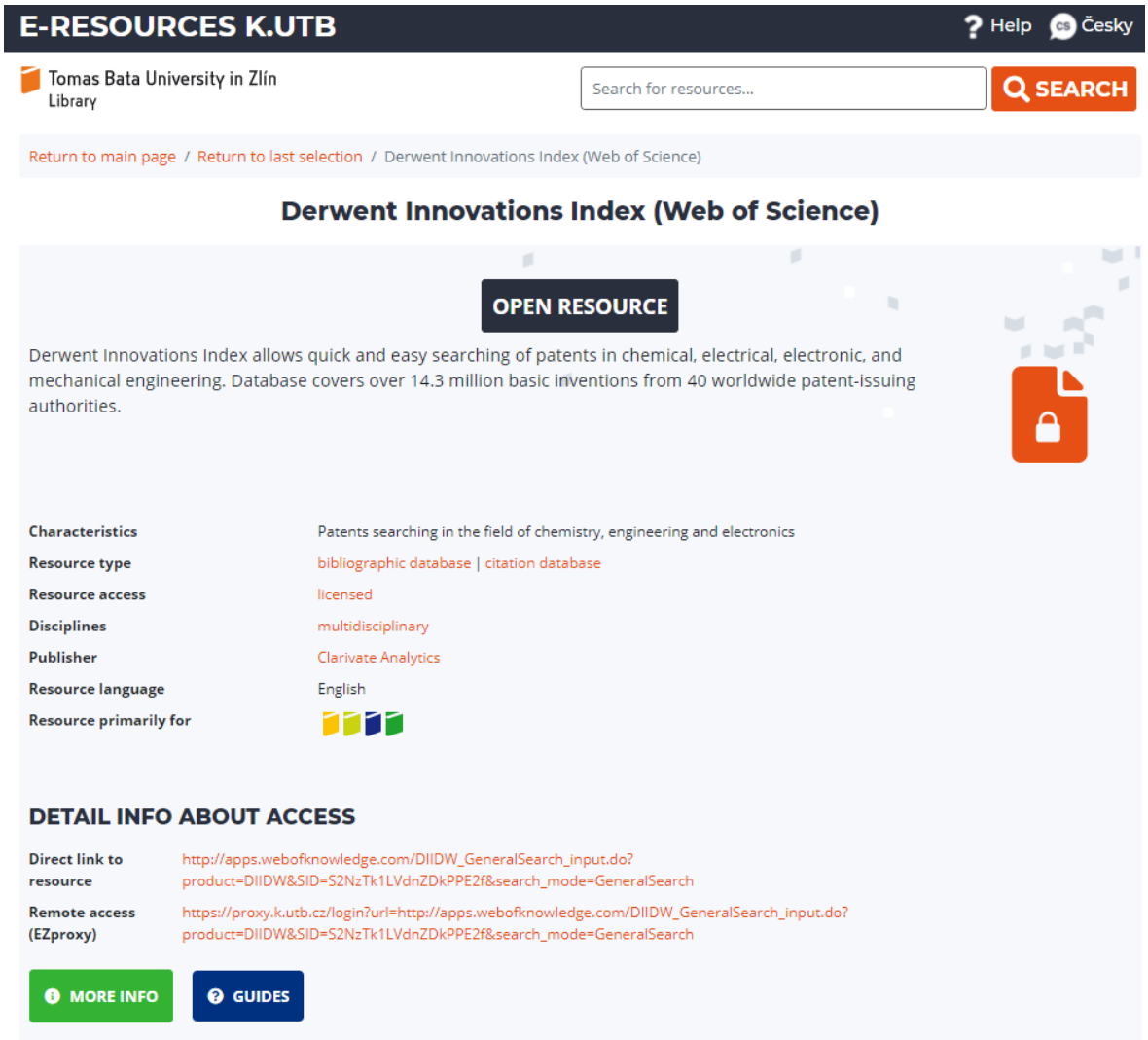


Obr. 24 Hlavní strana s aplikovanými filtry a anglickým překladem

9.2 Detail EIZ

Na straně s detailem EIZ jsou zobrazeny všechny položky, jejichž data jsou přítomna. Všechny oranžově zvýrazněné položky značí možnost interakce. V horní části je zobrazena navigace podobně jako na hlavní straně s aplikací filtrů. Jsou zde však přítomny dva odkazy. První z nich (na Obr. 25 je to text „Return to main page“) uživatele vrátí na hlavní stranu aplikace a smaže všechny dříve aktivované filtry. Druhý odkaz „Return to last selection“ provede stejnou navigaci, ale filtry vrátí do předchozího stavu – tzn. budou zobrazena stejná data jako před přechodem na detail EIZ.

Oranžově zvýrazněný text v samotném detailu EIZ značí dvě možnosti. V případě, že se jedná o tagy, tak se provede navigace na hlavní stranu s aplikováním filtru daného tagu. U ostatních položek – na Obr. 25 se jedná např. o položku „Publisher“ se jedná o odkazy na externí webové stránky.



Obr. 25 Detail EIZ s anglickým překladem

9.3 Nápověda

Nápověda (viz Obr. 26) neobsahuje žádná dynamická data. Jedná se pouze o statické HTML s využitím stávajících komponentů aplikace. I v nápovědě tak funguje přepínání jazyka nebo třeba vyhledávání, které uživatele opět přesměruje zpět na hlavní stranu aplikace.

Obr. 26 Nápověda

10 ZABEZPEČENÍ APLIKACE

Možné typy útoků na webové aplikace byly zmíněny již v kapitole 4 - Bezpečnost webových aplikací. V podkapitolách níže bude shrnuto zabezpečení aplikace vůči těmto typům útoků.

10.1 XSS a SQL Injection

XSS i SQL Injection mají společnou jednu věc – přijímání vstupu od uživatele. Ani jeden z útoků není pro tuto webovou aplikaci hrozbou.

Uživatel sice má možnost interakce s uživatelským rozhraním, to však na server již neposílá žádné dodatečné dotazy. Všechna data si aplikace vyžádá již při načtení a interakce již probíhají pouze v rámci klientské části aplikace. Aplikace filtrů a vyhledávání pomocí textu slouží pouze filtraci dat, která jsou uložena u klienta v prohlížeči.

10.2 DDoS

Ochrana proti DDoS v tomto případě není nezbytně nutná, protože je velmi nepravděpodobné, že by se o to někdo pokusil. Útok, který by mohl mít větší následky, by vyžadoval velké množství úsilí a peněz. Aplikace nebude sloužit ke komerčním účelům a zisk by pro útočníka tedy nebyl příliš lákavý.

V reálném provozu jsou data přicházející z API zkomprimována ve formátu gzip. Tato komprimace udržuje data, která jsou navrácena z databáze, po dobu 5 minut.

Systematická ochrana vůči DDoS útoku není zavedena. V případě DDoS se zátěž převede na server, který bude vracet data z výše zmíněného komprimovaného souboru. Nedojde tedy k přenesení zátěže na databázový server. Bohužel to však nezabrání přetížení serveru, na kterém je spuštěný backend aplikace a po přetížení nebude webová aplikace dostupná.

Pro zabezpečení aplikace proti tomuto typu útoku by teoreticky šlo využít externích služeb, které jsou k ochraně proti DDoS stavěny. Jednou z těchto služeb je Cloudflare nabízející ochranu proti útokům na aplikační vrstvě. Od verze „Pro“ je možno zabezpečit aplikaci s pomocí WAF a rozprostřením zátěže na síť serverů v cloudu. Aplikace by však musela být spuštěna přímo u Cloudflare a to pro potřeby této aplikace není žádoucí.

ZÁVĚR

V rámci této diplomové práce byla vytvořena webová single-page aplikace pro seznam elektronických informačních zdrojů. Před započetím vývoje aplikace byla věnována pozornost průzkumu možných způsobů vývoje požadované aplikace, porovnání výhod a nevýhod jednotlivých možností. Následně bylo zaměření přesunuto na možné technologie, které by splňovaly dané požadavky a byly by vhodné pro vývoj aplikace jak z vývojářského, tak i uživatelského hlediska.

V teoretické i praktické části práce byla věnována pozornost elektronickým informačním zdrojům, které jsou samotným obsahem aplikace. Jejich rozmanitost a definice různých pojmů byly definovány v rámci teoretické části, v praktické části byly vyzdvihnuty specifikace, které se týkají přímo dat o elektronických informačních zdrojích poskytnutých Knihovnou UTB.

Na základě poznatků získaných průzkumem v rámci teoretické části práce byl zvolen návrhový model single-page aplikace a JavaScriptový framework Angular, který je pro tento typ aplikace velmi dobře vybaven. Hlavním důvodem pro použití Angularu byla uniformnost ve vývoji frameworku a velkého množství dostupné funkcionality bez použití knihoven třetích stran. Díky této uniformnosti bylo poměrně jednoduché vyřešit všechny problémy, které se v průběhu vývoje objevily.

Finální aplikace splňuje všechny požadavky definované Knihovnou UTB. Všechny požadavky byly definovány v praktické části práce. Splnění těchto požadavků bylo zdokumentováno prostřednictvím náhledu na nejdůležitější části aplikace, včetně jejich implementace. Na závěr byla zhodnocena bezpečnost aplikace vůči hrozbám definovaným v teoretické části práce.

Výstupem práce je webová aplikace pro Knihovnu UTB, která bude sloužit veřejnosti pro přístup k elektronickým informačním zdrojům.

SEZNAM POUŽITÉ LITERATURY

[1] *A Little History of the World Wide Web. World Wide Web Consortium (W3C)* [online]. [cit. 2021-05-09]. Dostupné z: http://www.w3.org/History.html

[2] What is a Single Page Application?. *Software Development and IT Consulting | HUSPI* [online]. [cit. 2021-05-09]. Dostupné z: https://huspi.com/blog-open/definitive-guide-to-spa-why-do-we-need-single-page-applications

[3] Single-page applications vs. multiple-page applications: pros, cons, pitfalls. *OZiTAG* [online]. [cit. 2021-05-10]. Dostupné z: https://ozitag.com/blog/spa-advantages

[4] KAUR, Arshpreet. Single Page Application: Benefits, Pitfalls, and its Impact on Businesses. *Net solutions* [online]. [cit. 2021-05-10]. Dostupné z: https://www.netsolutions.com/insights/single-page-application/

[5] SKÓLSKI, Paweł. Single-page application vs. multiple-page application. *Medium* [online]. [cit. 2021-05-10]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

[6] Single-page App vs. Multi-page App: Pros, Cons, and Which is Better?. *Lvivity* [online]. [cit. 2021-05-10]. Dostupné z: https://lvivity.com/single-page-app-vs-multi-page-app

[7] VARAKSINA, Svitlana. Single-Page Applications vs Multi-Page Applications: The Battle of the Web Apps. *MindStudios* [online]. [cit. 2021-05-10]. Dostupné z: https://themindstudios.com/blog/spa-vs-mpa/

[8] SESHADRI, Shyam. *Angular Up & Running Learning Angular, Step by Step*. CA: O'Reilly Media, 2018. ISBN 978-149-1999-837.

[9] 2020 Developer Survey. *StackOverflow* [online]. [cit. 2021-05-19]. Dostupné z: https://insights.stackoverflow.com/survey/2020

[10] Angular (web framework). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-05-10]. Dostupné z: https://en.wikipedia.org/wiki/Angular_(web_framework)

[11] What about Angular? Angular JS history through the years (2009-2019): An extensive recap of the latest four years of Angular JS development: relevant milestones, new features, architecture improvements, optimizations, and so on. *Ryadel: Web Development, Networking, Security, SEO* [online]. [cit. 2021-05-10]. Dostupné z: https://www.ryadel.com/en/angular-angularjs-history-through-years-2009-2019/

[12] ULUCA, Doguhan. *Angular for Enterprise-Ready Web Applications*. 2nd edition. Washington DC: Packt, 2020. ISBN 1838648801.

[13] *Angular: The modern web developer's platform* [online]. [cit. 2021-05-10]. Dostupné z: https://angular.io/

[14] HUSSAIN, Asim. Lifecycle Hooks. *CodeCraft* [online]. [cit. 2021-05-10]. Dostupné z: https://codecraft.tv/courses/angular/components/lifecycle-hooks/

[15] React (JavaScript library). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-05-10]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[16] KRILL, Paul. React: Making faster, smoother UIs for data-driven Web apps. *InfoWorld* [online]. [cit. 2021-05-10]. Dostupné z: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html

[17] AFOLAYAN, Fisayo. The evolution of React. *Pusher* [online]. [cit. 2021-05-10]. Dostupné z: https://blog.pusher.com/the-evolution-of-react/

[18] *React: A JavaScript library for building user interfaces* [online]. [cit. 2021-05-10]. Dostupné z: https://reactjs.org/

[19] SUFIYAN, Taha. What is ReactJS: Introduction To React and Its Features. *SimpliLearn* [online]. [cit. 2021-05-10]. Dostupné z: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

[20] HAMEDANI, Mosh. React Lifecycle Methods – A Deep Dive. *Programming With Mosh* [online]. [cit. 2021-05-10]. Dostupné z: https://programmingwithmosh.com/javascript/react-lifecycle-methods/

[21] *Vue.js: The Progressive JavaScript Framework* [online]. [cit. 2021-05-10]. Dostupné z: https://vuejs.org/

[22] What is Vuex?. *Vuex* [online]. [cit. 2021-05-10]. Dostupné z: https://vuex.vuejs.org/

[23] DAITYARI, Shaumik. Angular vs React vs Vue: Which Framework to Choose in 2021. *CodeinWP* [online]. [cit. 2021-05-10]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

[24] CELBOVÁ, Ludmila. Elektronický zdroj. *Databáze národní knihovny ČR* [online]. [cit. 2021-05-10]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000872&local_base=KTD

[25] BENEŠOVÁ, Ludmila. *Elektronické informační zdroje* [online]. Jihočeská vědecká knihovna v Českých Budějovicích, 2018 [cit. 2021-05-10].

[26] FABIÁN, Ondřej. *Elektronické informační zdroje* [online]. Brno: Centrun NAKLIV, 2012 [cit. 2021-05-10]. Dostupné z: https://web2.mlp.cz/koweb/00/04/23/37/07/elektronicke-informacni-zdroje.pdf

[27] Elektronické databáze. *Iva: Informační výchova na UTB ve Zlíně* [online]. [cit. 2021-05-10]. Dostupné z: http://iva.k.utb.cz/lekce/elektronicke-databaze/

[28] Remote access to Taylor & Francis Online made easier with SeamlessAccess. *Taylor & Francis: An Informa Business* [online]. [cit. 2021-05-10]. Dostupné z: https://newsroom.taylorandfrancisgroup.com/remote-access-to-taylor-francis-online-made-easier-with-seamlessaccess/

[29] *OWASP* [online]. [cit. 2021-05-10]. Dostupné z: https://owasp.org/

[30] What is a DDoS Attack?: DDoS attacks are a primary concern in Internet security today. Explore details about how DDoS attacks function, and how they can be stopped. CloudFlare [online]. [cit. 2021-05-10]. Dostupné z: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SPA	Single-page aplikace
MPA	Multi-page aplikace
PWA	Progresivní webová aplikace
HTML	Hyper Text Markup Language
DOM	Document Object Model
API	Application Programming Interface
SEO	Search Engine Optimalization
MVVM	Model-View-ViewModel
MVC	Model-View-Controller
DI	Dependency Injection
CLI	Command Line Interface
EIZ	Elektronický informační zdroj
HAN	Hidden Automatic Navigator
DOI	Digital Object Finder
DDoS	Distributed Denial of Service
XSS	Cross-Site Scripting
WAF	Web Application Firewall
JSX	JavaScript Syntax Extension

SEZNAM OBRÁZKŮ

SEZNAM KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Přiložené CD obsahuje:

- Text
    - dusan_gavenda_dp.pdf – diplomová práce ve formátu PDF/A
- Zdrojové kódy aplikace
    - dusan_gavenda_dp_prakticka_cast.zip – všechny zdrojové kódy aplikace