FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace WhereIS
Student:	Tomáš Heger
Vedoucí:	Ing. David Bernhauer
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Nejen pro nové studenty může být na fakultě obtížné najít správnou místnost. Řešením může být aplikace, která po zadání hledaného objektu (místnost, předmět, vyučující) uživatele naviguje správným směrem (do místnosti, na webovou stránku).

Analyzujte výhody a nevýhody existujících nebo alespoň podobných aplikací a sestavte požadavky pro vyhledávací systém WhereIS.
Proveďte rešerši existujících systémů v souvislosti s aplikací.
Proveďte analýzu potřebných služeb a jejich API.
Navrhněte vhodné propojení systémů v rámci budované aplikace.
Vytvořte a otestujte prototyp webové aplikace na základě provedené analýzy a návrhu.
Diskutujte případné problémy se službami (zdroji dat), ke kterým nemáte přístup.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 21. listopadu 2021 v Praze.

Bakalářská práce

APLIKACE WHEREIS

Tomáš Heger

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. David Bernhauer
10. května 2022

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu práce Ing. Davidu Bernhauerovi, který mi pomáhal vést práci správným směrem. Dále bych chtěl poděkovat svým přátelům Illiovi Brylovovi, Jakubu Jabůrkovi, Veronice Vlkové, Gabrielu Hévrovi a Janu Kaprálovi, kteří se zúčastnili uživatelského testování aplikace, poskytli mi podrobnou zpětnou vazbu k aplikaci a podporovali mne během psaní práce. Na závěr bych rád poděkoval svým rodičům, prarodičům a rodinným příslušníkům, kteří mne podporovali během celého studia a psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 10. května 2022 .....

Abstrakt

Tato práce se zabývá analýzou, návrhem a tvorbou webového vyhledávacího a informačního systému s názvem WhereIS, který má za cíl zjednodušit studentům FIT ČVUT vyhledávání informací o předmětech, učitelích a místnostech a umožnit studentům vyhledávat volné místnosti k učení. Návrh a implementace samotného prototypu jsou cíleny na tři základní části, a to získání dat ze zdrojů, vyhledávání nad daty (agregaci výsledků) a jednoduchou rozšiřitelnost aplikace. V rámci praktické implementační části je vytvořen a otestován prototyp takového vyhledávacího a informačního systému, který splňuje požadavky na agregaci výsledků a snadné rozšiřitelnosti.

Klíčová slova WhereIS, webová aplikace, vyhledávací systém, metavyhledávací systém, informační systém, agregátor

Abstract

This thesis focuses on the analysis, design and development of a web-based search and information system called WhereIS, which aims to simplify the search for information about courses, teachers and rooms for students of FIT CTU and to enable students to search for available rooms for studying. The design and implementation of the prototype itself are aimed at three main parts, namely obtaining data from sources, searching over the data (aggregation of results) and easy extensibility of the application. As part of the practical implementation part, a prototype of such a search and information system that meets the requirements for aggregation of results and simple extensibility is developed and tested.

Keywords WhereIS, web application, search system, metasearch engine, informational system, aggregator

Seznam zkratek

HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
URN	Uniform Resource Name
API	Application Programming Interface
JSON	JavaScript Object Notation
RSS	Really Simple Syndication
XML	Extensible Markup Language
Atom	Atom Syndication Format
CSRF	Cross Site Request Forgery
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
PHP	Hypertext Preprocesor (původně Personal Home Page)
DOM	Document Object Model
JVM	Java Virtual Machine
MVC	Model-View-Controller
CLI	Command Line Interface
ORM	Object Relational Mapping
SQL	Structured Query Language
NoSQL	Not only SQL
XSS	Cross Site Scripting
GUI	Graphical User Interface

Kapitola 1
Úvod

Představte si, že jste začali svůj první den na Fakultě informačních technologií ČVUT v Praze a máte jít na své první cvičení. Podíváte se do rozvrhu a tam vidíte pouze kódové značení místnosti T9:349, což Vám, jako čerstvému prvákovi, nic moc neřekne. A co teď? Nezbývá Vám nic jiného, než se zeptat někoho znalejšího nebo zběsile vyhledávat mezi velkým množstvím informačních systémů a webových aplikací, které škola nabízí, abyste zjistili, kde se daná místnost nachází.

Hlavním problémem je tedy velké množství informačních systémů a chybějící centrální bod zastřešující všechny informace dohromady. Další častou nemilou situací, ve které se za svou dobu studia na této fakultě ocitne nemalé množství studentů, je nalezení volné počítačové nebo seminární místnosti, ve které by se studenti chtěli se svými kamarády připravovat na zápočtové testy nebo zkoušky. Jenže to znamená postupné procházení tištěných rozvrhů všech místností, aby po chvíli zjistili, že žádná místnost aktuálně volná není. Přesně kvůli těmto dvěma častým situacím jsem si vybral toto téma bakalářské práce, aby výsledná aplikace pomáhala studentům rychle a jednoduše nalézt místnosti ve fakultních budovách, aby usnadnila vyhledávání volných místností k učení, ale také i dalších informací, jako informace o předmětech či učitelích. Název aplikace WhereIS byl zvolen proto, že v anglickém jazyce tato slova znamenají „Kde je“ a „IS“ je zkratka pro informační systém.

1.1 Cíle práce

Hlavní cíl této bakalářské práce je tedy navrhnout a vytvořit funkční prototyp vyhledávacího systému WhereIS, který budou moci využívat všichni studenti i učitelé k snadnější orientaci na fakultě a hledání potřebných informací.

Teoretická část se zaměřuje na analýzu již existujících řešení a přístupů, které lze zvolit při vytváření takové webové aplikace jako je WhereIS, a jaké jsou jejich výhody a nevýhody. Dále se práce zaměřuje na analýzu potřebných služeb a jejich API, co je potřeba udělat k získání přístupů k datům těchto služeb, a na analýzu procesů a požadavků pro vyhledávací systém WhereIS. Dalším cílem je takový návrh aplikace, který má umožnit jednoduchou integraci dalších entit do vyhledávání bez nutnosti velkého zásahu do aplikace. Posledním cílem teoretické části je diskuse nad volbou programovacích jazyků a frameworků, ve kterých je možné aplikaci vytvořit, a potřeby použití databázového systému.

Cílem praktické části je samotná implementace prototypu webové aplikace, který bude splňovat všechny řečené požadavky na základě zvoleného návrhu, programovacího jazyka a frameworku, o kterých se diskutovalo v závěru teoretické části. Dalším cílem je nalezení vhodného způsobu ukládání dat a práce s daty získaných z již existujících systémů FIT ČVUT. Posledním

cílem praktické části je otestování vytvořeného prototypu aplikace WhereIS studenty FIT ČVUT a zpracování jejich zpětné vazby o případných možnostech budoucího rozšíření aplikace.

1.2 Struktura práce

Kapitola 1 uvádí čtenáře do prostředí řešené problematiky, definuje cíle práce a seznamuje čtenáře se strukturou práce.

Kapitola 2 se zabývá analýzou. V první části této kapitoly jsou analyzována podobná existující řešení. V rámci rešerše je popsáno, co je to webový vyhledávač a jak takový vyhledávač funguje. Je zmíněn jeden z největších problémů webových vyhledávačů a je představen jejich nástupce, což jsou vyhledávací agregátory (metasearch engines). Na konci rešerše jsou shrnuty výhody a nevýhody obou vyhledávacích systémů. Po rešerši následuje analýza zdrojů dat, ze kterých bude aplikace data získávat a zpracovávat, a jsou popsány jejich výhody a nevýhody. Následuje analýza funkčních a nefunkčních požadavků aplikace, kde je podrobněji popsáno, co je od aplikace vyžadováno. Na konci kapitoly jsou analyzovány procesy v aplikaci.

Kapitola 3 se zabývá návrhem a implementací. Na začátku kapitoly jsou zmíněny a popsány použité technologie potřebné pro správné fungování aplikace. Poté následuje diskuse nad dostupnými a vhodnými programovacími jazyky, frameworky a databázovými systémy, které lze pro vytvoření aplikace využít. Následně je popsána architektura aplikace z pohledu dvouvrstvé a třívrstvé architektury. Poté následuje popis modelu MVP a jeho využití v aplikaci WhereIS. Ke konci kapitoly je řešena bezpečnost aplikace, jaké bezpečností prostředky aplikace využívá a jaké prostředky jsou implementovány pro ochranu dat, nad kterými se vyhledává. Na konci kapitoly je popsáno, jakým způsobem lze v aplikaci jednoduše integrovat nové entity a zdroje dat.

Kapitola 4 se zabývá testováním vzniklého prototypu, jaké možnosti testování, případně heuristiky, byly použity a jaké jsou výsledky. Kromě výsledků se kapitola zabývá i zpětnou vazbou studentů, kteří prototyp testovali, a jsou zpracovány jejich návrhy na možné budoucí rozšíření aplikace.

Poslední kapitola 5 obsahuje samotný závěr práce, ve kterém jsou shrnuty dosažené a případně nedosažené cíle této práce a ve kterém se zamýšlím nad možným budoucím vývojem vyhledávacího systému WhereIS.

Kapitola 2
Analýza

2.1 Rešerše existujících řešení

Vyhledávání informací (nejen na internetu) je běžnou činností, která pro typického uživatele není žádnou záhadou. Vyhledávání usnadňuje uživatelům často složitou orientaci v různých hierarchických strukturách. V rámci operačního systému je dnes samozřejmostí vyhledávání souborů, na jakém disku a v jaké složce se nachází, nebo vyhledávání určitých nastavení, kde uživatel může nastavit výstup zvuku, kde změní pozadí plochy apod.

Na internetových stránkách se situace nijak neliší. Téměř každá webová stránka má implementovaný nějaký vyhledávač, ať už je to na vyhledávání článků, zaregistrovaných uživatelů, zpráv, fotek, videí, e-mailů, aktualizací apod. Co to tedy ten (webový) vyhledávač je?

2.1.1 Webový vyhledávač

Jak je uvedeno v [1], webový vyhledávač je softwarový systém, jehož cílem je prohledávání internetu systematickou cestou za účelem nalezení hledaných stránek či informací. V dnešní době navíc většina takových vyhledávačů prohledává kromě internetu také databáze, do kterých má přístup. Výsledky těchto systémů jsou často zobrazovány jako řádky na specifických webových stránkách, které se nazývají *search engine results pages*. Vrácené výsledky mohou být také různých typů. Může se jednat o směs odkazů na webové stránky, videa, obrázky, soubory, články atd.

V [1] také vymezují 3 primární funkce, na základě kterých dnešní webové vyhledávače fungují:

1. crawling — využívá tzv. *crawler*, což je program (bot), který prohledává obsah každé stránky, kterou najde nebo ke které se dostane pomocí URL,
2. indexing — uchovává a organizuje obsah nalezený během crawling procesu, jakmile je stránka indexovaná, je připravena k zobrazení jako výsledek relevantních dotazů,
3. ranking — řazení nalezených výsledků od nejvíce relevantních po nejméně relevantní na základě daného dotazu.

Web crawling Jak je naznačeno výše a je podobně řečeno v [2], (web) crawling je proces prohledávání webových stránek, který využívá tzv. crawler (též robot, spider). Ten si stáhne danou stránku pomocí její URL adresy a HTTP metody `GET` a v staženém textovém obsahu hledá žádané informace.

Jak říkají Olston a Najork v úvodu své knihy [2], crawleři se používají i pro jiné účely než jen ve webových vyhledávačích pro prohledávání internetu. Druhým účelem je jejich použití v archivaci webu, kdy crawleři sbírají různé webové stránky a archivují je do těchto archivů. Třetím účelem je vytěžování dat, kde jsou stránky analyzovány pro statistické účely nebo je na stránkách prováděna datová analýza. Poslední účel, který Olston a Najork zmiňují, je využití služeb starající se o monitorování webu, které na klientův dotaz neustále prohledávají web a informují klienta o výsledcích na jeho dotaz.

V současné době se crawleři potýkají s dvěma největšími problémy. Prvním z nich je fakt, že čím dál tím větší počet webových stránek se chrání před těmito (a i dalšími) roboty pomocí JavaScriptu či jiných nástrojů jako je například CAPTCHA. CAPTCHA (akronym pro Completely Automated Public Turing test to tell Computers and Humans Apart) je krátký jednoduchý test sloužící k rozlišení lidí od robotů. Crawler tedy nemusí být schopen získat data z těchto stránek. Druhým problémem je velikost samotného webu. Crawler v rozumném čase nebude schopný stáhnout a prohledat každou dostupnou stránku internetu. Tento proces by zabral nejen obrovské množství paměti, ale také by trval velmi dlouhou dobu.

Indexing Indexování je v [1] popsáno jako proces, který ukládá výsledky nalezené crawlerem do databáze a nad těmito daty vytváří index. Díky indexaci je pak vyhledávací systém schopný velmi rychle a efektivně v této databázi vyhledávat data a vracet uživatelům nejrelevantnější data mezi prvními výsledky. Databáze těchto dat zabírá obrovské množství paměti a bez indexace by vyhledávání v této databázi mohlo zabrat desítky minut až hodiny.

Ranking Ranking je proces, během kterého se výsledky na uživatelův dotaz seřadí podle určitých pravidel tak, aby se mezi prvními výsledky zobrazily ty, které jsou pro uživatele nejvíce relevantní a chtěné. Každý vyhledávací systém může mít toto řazení naprogramované jiným způsobem. Jeden z nejznámějších a neúspěšnějších ranking systémů je *PageRank* od společnosti Google, který je používán ve webovém vyhledávači *Google Search*.

2.1.2 Agregátory (metasearch engines)

Webové vyhledávače jsou velmi rozšířené i přes fakt, že mají jednu zásadní nevýhodu — malé pokrytí způsobené velkým množstvím dat na internetu. Pro jeden webový vyhledávač je nemožné pokrýt a vyhledávat ve všech dostupných stránkách. I dnes nejznámější a nejpoužívanější webový vyhledávač *Google Search* od společnosti Google, který má podle [3] podíl na trhu 91,42 %, nemá pokryté všechny webové stránky. V článku [4] Indig zmiňuje, že v roce 2016 objevil Google za onen rok přes 130 bilionů stránek, nicméně je neprozkoumal a neindexoval všechny, protože kolem 30 % stránek je považováno za spam, podobné jiným, nebo nesmyslné. Z toho důvodu není hlavním cílem Google v současné době mít co nejvíce indexovaných stránek, ale mít kvalitní data, ve kterých se vyhledává. Toto chování je vidět na obrázcích 2.1, kde na obrázku 2.1a je vidět počet nalezených webových stránek v bilionech za rok (tento počet roste téměř exponenciálně) a na obrázku 2.1b celkový počet indexovaných stránek.

Za účelem zvětšení celkového pokrytí webových vyhledávačů vznikly *vyhledávací agregátory* (*metavyhledávací systémy*, v anglickém jazyce *metasearch engines*). Meng ve svém článku [5] definuje vyhledávací agregátor jako komponentu vyhledávacích systémů. Uživatel zadá svůj dotaz do metavyhledávacího systému a ten předá stejný dotaz do několika vyhledávacích systémů, na které je napojený. Vyhledávání tedy provede množina nezávislých vyhledávacích systémů, které své výsledky vrátí volajícímu vyhledávacímu agregátoru. Tyto výsledky jsou pak spojeny do jednoho výsledkového listu, dojde k rankingu výsledků a k následnému zobrazení uživateli. Díky tomu, že vyhledávací agregátory používají množinu nezávislých vyhledávacích systému k vyhledávání, budou mít tato vyhledávání mnohem větší pokrytí, než jeden samotný vyhledávací systém.

2.1.2.1 Heureka a Zboží.cz

Mezi nejznámější vyhledávací agregátory v České republice patří aplikace *Heureka*[1] a *Zboží.cz*[2]. Jedná se o vyhledávací agregátory zboží, které mají za cíl uživateli najít nejlevnější či nejoblíbenější produkty ve zvolené kategorii. Tyto portály fungují velmi podobně. Uživatel zvolí typ zboží, který chce nakoupit, například „webkamera“, a portály mu vypíší nejoblíbenější webkamery. Navíc po kliknutí na určitou webkameru mu portály naleznou všechny obchody, ve kterých se webkamera prodává, seřazené podle ceny vzestupně. Z toho lze usoudit, že oba portály agregují podle oblíbenosti produktů (počtu nakoupených kusů a hodnocení) a ceny produktu.

Jak je z obrázků 2.2 vidět, oba portály vypadají podobně. Výsledky na dotaz „webkamera“ se samozřejmě liší, protože portály pravděpodobně agregují výsledky z jiné množiny obchodů a mají jiné informace o tom, co jací uživatelé kupují a co je jak oblíbené. To je dáno tím, že většina uživatelů preferuje používat pouze jeden z těchto agregátorů a velmi výjimečně používají oba dva tyto agregátory stejně často.

Heureka ani Zboží.cz nemají zveřejněné své zdrojové kódy a ani dokumentaci popisující proces vyhledávání a ranking produktů. Nicméně dle mého názoru fungují přesně na principu vyhledávacího agregátoru. Obě aplikace budou mít databázi obchodů, ve kterých budou zboží vyhledávat. Po zadání uživatelova dotazu na produkt dojde k prohledání všech obchodů pomocí jejich vyhledávacích systémů a získají se výsledky. Potom už samotné aplikace podle určitých pravidel uživateli výsledky seskupí a seřadí. Samozřejmě věcí na pozadí se u obou aplikací bude dít mnohem více (např. cachování, indexování…), ale to je v rámci pochopení základního fungování implementační detail.

2.1.2.2 Google Search jako agregátor

Google Search[3], jak už bylo řečeno, je v současné době nejpoužívanějším webovým vyhledávačem. Nicméně u většiny uživatelských dotazů Google Search také funguje jako vyhledávací agregátor. Mějme například dotaz na herce „Johnny Depp“. První výsledky dotazu, které jsou vidět na obrázku 2.3, jsou různé články a zprávy o herci z různých novinářských a televizních webových stránek. Jedná se o ukázkový příklad výsledků vyhledávacího agregátoru.

Další příklad agregace je vidět ihned napravo od zmíněných novinových článků (obrázek 2.4). Jedná se o box nejzákladnějších informací o herci. Nalezneme zde fotky, část informací o herci a jeho životě (z Wikipedie), filmy a seriály, ve kterých herec hrál (pravděpodobně z IMDb), odkazy na sociální sítě a další osobnosti, které uživatelé ve spojení s hercem také vyhledávali.

Posledním příkladem je agregace videí z různých webových stránek a aplikací (viz obrázek 2.5). Nejčastěji se zde nachází videa z platformy *YouTube*, která spadá také pod společnost Google, najdeme zde ale i videa z různých novinářských a bulvárních stránek, jako *Blesk*, *New York Post* atd.

2.1.3 Shrnutí

V době psaní této práce jsou webové vyhledávače velmi oblíbené a hojně používané, nicméně jejich nástupci v podobě vyhledávacích agregátorů se už také začínají dočkávat oblíbenosti a slávy, zejména v odvětví nakupování. Oba dva vyhledávací systémy mají své výhody a nevýhody.

Webové vyhledávače jsou relativně jednoduché na implementaci a údržbu, mají však malé pokrytí stránek a pro rozšíření tohoto pokrytí je zapotřebí velkého množství paměti na disku a použití efektivních nástrojů jako je cachování či indexace, aby dotazování netrvalo příliš dlouho.

Vyhledávací agregátory jsou složitější na implementaci. Jak je uvedeno v úvodu článku [6], je nutné při vyhledávání zjistit, jakých vyhledávačů se má agregátor zeptat pro získání relevantních

[1] https://www.heureka.cz/
[2] https://www.zbozi.cz/
[3] https://www.google.cz/

výsledků, jak změnit uživatelův dotaz tak, aby pro dané vyhledávače byl co nejvíce srozumitelný, a jak má získané výsledky seřadit. Avšak pokud je správně naimplementovaný, pokrývá takový vyhledávací agregátor mnohem větší množinu webových stránek a díky tomu může vracet více relevantních informací než webový vyhledávač.

Vyhledávací systém WhereIS bude z těchto důvodů na pomezí webového vyhledávače a vyhledávacího agregátoru (blíže bude mít k agregátoru). WhereIS bude vyhledávat nad databází dat nebo aktuálně obdrženými daty z určitých zdrojů. Tato práce se nebude zabývat využíváním crawlerů ani jiných vyhledávačů k získání dat, bude využívat webové služby (API) již existujících webových aplikací. To však nebrání jejímu dalšímu rozšíření.

2.2 Zdroje dat

Aby aplikace mohla agregovat výsledky vyhledávání nad předměty, učiteli a místnostmi, je potřeba získat data o těchto entitách. K tomuto účelu lze využít již existující webové služby na FIT ČVUT.

„*Webová služba (Web Service) je zapouzdřená logika (funkcionalita) jedné aplikace, kterou využívá jiná aplikace. Jedná se tedy o strojovou interakci – webové služby NEJSOU určeny pro lidskou interakci.*“[7]. Webové služby (zkráceně lze říct jen služby) jsou tedy takové části aplikace, které jsou používané jinými aplikacemi a ne uživateli. Mezi takovéto služby patří například *Sirius*, *KOSapi* nebo *Umapi*.

Služba Sirius zprostředkovává data, které využívá aplikace *TimeTable*, služba KOSapi zprostředkovává data aplikace *KOS* a služba Umapi zprostředkovává data aplikace *UserMap*. Všechny tyto služby využívají autorizační protokol *OAuth 2.0* a jsou *RESTfull*. Aby vývojář získal přístup k těmto službám, je potřeba zaregistrovat svůj projekt ve službě *Apps Manager* (detailní popis v sekci 3.1.3.1).

2.2.1 Sirius

Sirius je API, které mimo jiné využívá aplikace TimeTable. Jedná se o REST API dostupné na adrese `https://sirius.fit.cvut.cz/api/v1`, které poskytuje svá data ve formátu JSON. V dokumentaci služby Sirius na adrese `http://cvut.github.io/sirius/docs/api-v1.html` jsou uvedeny všechny zdroje, ze kterých lze touto službou získat data. Jedná se zejména o události předmětů, místností a učitelů, do kdy se v jaké místnosti učí/zkouší, jaké předměty se v jakých místnostech vyučují apod. V rámci každého požadavku je možné v hlavičce požadavku specifikovat a konfigurovat určité dotazovací parametry, například o jaký typ události se má jednat, od/do jakého data a hodiny mají události začínat/končit, kolik událostí chceme maximálně vrátit apod.

Pro vyhledávací systém WhereIS jsou vhodné následující požadavky:

`GET /courses/{courseCode}/events` vrátí všechny události (kalendář) předmětu,

`GET /teachers/{username}/events` vrátí všechny události (kalendář) učitele, které učí, organizuje, zkouší…(zde je nutné vyžádat práva pro čtení),

`GET /rooms/{kosId}/events` vrátí všechny události v dané místnosti.

Poslední 2 uvedené požadavky lze také využít pro vyhledání „aktuální polohy“ učitele — v jaké místnosti právě učí — nebo k vyhledání aktuálně volných místností, do kterých se mohou jít studenti učit.

Služba Sirius je tedy vhodná na získávání a zpracování dat o událostech předmětů, učitelů či místností. Lze tak velmi jednoduše získat přehledný a informativní kalendář ke každé z těchto entit nebo lze pomocí dotazovacích parametrů získat pouze ty události, které nás jako uživatele zajímají.

Mezi nevýhody této služby se řadí zejména nepropojenost s jinými službami FIT ČVUT. Jak v datech poskytovaných službou Sirius, tak ani v samotné aplikaci TimeTable není například možnost prokliku na stránky předmětu (např. Courses). Jediné propojení, které aplikace nabízí, je se službou UserMap, kdy se po kliknutí na jméno učitele zobrazí možnost otevření jeho profilu v aplikaci UserMap. Další nevýhoda této služby je absence možnosti zobrazení aktuálně volných místností, do kterých by se studenti mohli chodit učit.

2.2.2 KOSapi

KOSapi je API aplikace KOS. Jedná se o REST API dostupné na adrese `https://kosapi.fit.cvut.cz/api/3/`, které poskytuje svá data ve formátu Atom nebo XML. Jak je uvedeno v [8], Atom je název formátu pro spravování webového obsahu a metadat založeného na XML a k jeho publikování a úpravám na webových stránkách. Jedná se o nástupce RSS formátu.

API neposkytuje data ve formátu JSON, což může vést ke ztížení práce s tímto zdrojem kvůli nutnosti implementace parseru pro jiný formát. V dokumentaci KOSapi na adrese `https://kosapi.fit.cvut.cz/projects/kosapi/wiki/` lze vyčíst všechny zdroje, ze kterých lze získat data. Jedná se o data učitelů, předmětů, místností, zkoušek, semestrů, paralelek a dalších entit, které jsou součástí ČVUT. Služba tak nevrací pouze data týkající se FIT ČVUT, ale data celé univerzity ČVUT. V požadavcích lze však využít dotazovacích parametrů, na základě kterých můžeme vyfiltrovat pouze taková data, která spadají pod FIT ČVUT.

Pro vyhledávací systém WhereIS jsou vhodné následující požadavky:

`GET /teachers` vrátí všechny vyučující na ČVUT,

`GET /rooms` vrátí všechny místnosti na ČVUT,

`GET /courses` vrátí všechny předměty na ČVUT.

Tyto požadavky lze využít hlavně na získání, předzpracování a uložení základních dat, pomocí kterých se může WhereIS dotazovat v dalších službách. Jedná se zejména o `kosID` místností, `username` učitelů a `courseCode` předmětů.

Služba KOSapi je tedy vhodná na získání a uložení základních dat o entitách, které lze pak využít na jiné, podrobnější dotazy v dalších službách. Výhodou je také možnost filtrování dat, které chceme získat, což je vhodné zejména proto, že služba vrací data ze všech fakult ČVUT.

Nevýhodou KOSapi je opět nepropojenost s jinými službami. Částečné propojení funguje u předmětů, kde k většině předmětů služba vrátí jejich nastavenou domovskou stránku, která ve většině případů vede na Courses, v některých případech na Moodle. U učitelů však není žádné propojení s UserMapem a u předmětů, učitelů a místností není možnost zobrazit si jejich rozvrh pomocí TimeTable. Další nevýhodou je neaktuálnost či neúplnost dat, kdy například některé domovské stránky předmětů nebyly aktualizovány na nové nebo místnosti pro FIT ČVUT vrácené přes KOSapi neobsahovaly všechny místnosti, ve kterých se učí. Poslední nevýhodou je absence formátu JSON, kvůli které je nutná implementace parseru pro jiný datový formát.

2.2.3 Umapi

Umapi je API aplikace UserMap. Jedná se o REST API dostupné na adrese `https://kosapi.fit.cvut.cz/usermap/v1`, které poskytuje svá data ve formátu JSON. V dokumentaci Umapi na adrese `https://kosapi.fit.cvut.cz/usermap/doc/rest-api-v1.html` lze vyčíst všechny zdroje, ze kterých lze získat data. Služba poskytuje informace o všech zaměstnancích ČVUT (nejen zaměstnancích FIT). Mezi tyto informace patří jejich fotka, kancelář, kontaktní údaje (e-mail, telefonní čísla…), pod jakou katedrou jsou zaměstnaní nebo jaké mají přidělené role. Do aplikace UserMap lze odkazy na profily generovat ve formátu `https://usermap.cvut.cz/profile/{username}`. Nicméně i přes tento fakt je lepší informace o zaměstnancích agregovat pomocí

Umapi, protože lze pak tato data více propojit s aplikací WhereIS (například propojení kanceláře s vyhledáním místnosti).

Pro vyhledávací systém WhereIS jsou vhodné následující požadavky:

`GET /people/{username}` vrátí informace o zaměstnanci — kontaktní údaje (e-mail, telefonní číslo…), kancelář, katedru a role zaměstnance,

`GET /people/{username}/photo` vrátí fotku učitele.

Služba Umapi (a aplikace UserMap) je tedy vhodná na získání podrobnějších informací o všech zaměstnancích ČVUT. Mezi všemi zaměstnanci, fakultami a katedrami lze jednoduše filtrovat a vyhledávat. Další výhodou je propojení rolí zaměstnanců s filtrem všech rolí na dané fakultě a katedře.

Nevýhodou Umapi (a aplikace UserMap) je již několikrát zmíněná nepropojenost s existujícími systémy. Výsledky z této služby již neodkazují na žádné jiné zdroje dat (KOS, TimeTable…), odkazují pouze na jiné stránky v rámci této služby/aplikace.

2.2.4 Nedostupné služby

V době psaní této práce neexistovala žádná služba poskytující ucelená zdrojová data pro místnosti, maximálně jen informace jednoho typu (události, kódové označení…). Umapi poskytuje pouze data o lidech a rolích, KOSapi poskytuje data zejména o předmětech a základní údaje o učitelích a Sirius poskytuje data o událostech předmětů, místností a učitelů.

KOSapi poskytuje pouze seznam kódových označení místností (např. T9:349), jejich název (NBFIT PC učebna) a lokaci v podobě zkratek DEJ (Dejvice), THA (Thákurova ulice, budova A), DIST (distanční) nebo VŠE (místnosti v budovách VŠE). Podrobnější informace o lokaci místnosti, jejího typu, volnosti přístupu či plánku, kde se místnost nachází, se musí najít v jiných informačních systémech.

Tento problém chybějící služby se práce snaží alespoň částečně vyřešit. Z pomocné stránky[4] pro studenty lze zjistit místnosti, do kterých mají studenti volný přístup (bez nutnosti dozoru). Navíc se na těchto stránkách nachází plánky většiny pater školy. Bohužel nejsou v těchto pláncích zaznamenány všechny místnosti, ve kterých se na FIT ČVUT učí, nicméně je lze použít alespoň pro základní orientaci, kde by se místnosti mohly nacházet. Bohužel v současnou chvíli není taková data možné jednoduše strojově zpracovávat a je nutné je zpracovat manuálně.

2.2.5 Další služby

Do výsledného prototypu vyhledávacího systému WhereIS je možné zapojit i více služeb než předchozí zmíněné a tím lze získat přístup k dalším datům. Jedním z takových systémů je systém Agáta[5] od SUZ, který poskytuje informace o jídelníčcích ve všech menzách ČVUT. Díky napojení na tento systém by mohli uživatelé vyhledávat menzy, ve kterých v daný den vaří jejich oblíbené jídlo (např. řízek).

Další službou, která by se dala napojit na WhereIS, je ISKAM[6]. Díky této službě ve spojení s dalšími informačními systémy SUZ by WhereIS mohlo poskytovat informace o kolejích ČVUT, například kde se kolej nachází, kolejné, odkazy na stránky kolejí apod.

Práce se implementaci dalších služeb nevěnuje. V první fázi vývoje je cílem agregovat základní informace o místnostech, učitelích a předmětech. Zároveň je kladen důraz na jednoduchou rozšiřitelnost o další druhy entit, aby bylo možné v budoucích verzích agregovat větší množství webových aplikací a informačních systému. To by mělo ulehčit orientaci v těchto systémech.

[4] https://help.fit.cvut.cz/rooms/classrooms/list.html
[5] https://agata.suz.cvut.cz/jidelnicky/index.php
[6] https://web.suz.cvut.cz/Home/Index

2.2.6 Možné problémy při rozšiřování aplikace

Při budoucím rozšiřování aplikace o dříve zmíněné služby (v kapitole 2.2.5) bude nutné analyzovat jejich data, funkčnost, zabezpečení a rozhraní. To však může přinést několik problémů:

- jiný formát, ve kterém jsou data poskytována — nutnost implementace nového parseru,
- nedostatečná oprávnění, kvůli kterým nebude zajištěn přístup k datům,
- jiný autorizační protokol než OAuth 2.0 — nutnost implementace nové metody autorizace a získání autorizačních údajů,
- nekonzistentní data používaných služeb — např. v KOSapi nemusí data souhlasit se současností (neaktuální stránky předmětů, nevyplněné veškeré místnosti apod.).

Z těchto důvodů nemusí být rozšíření aplikace tak jednoduché a i přes dobrý návrh a implementaci samotného vyhledávání (agregátoru) může být potřeba implementovat další pomocné nástroje pro integraci nových entit (dat).

2.3 Analýza požadavků

Požadavky na software lze chápat jako potřebná kritéria, která má software splňovat, a kvality, kterými jich lze dosáhnout. Podle toho požadavky dělíme na funkční a nefunkční.

2.3.1 Funkční požadavky

Funkční požadavky říkají, jaké funkčnosti má daná aplikace mít a co má umět. V rámci vyhledávacího systému WhereIS tak nejdůležitějšími funkčními požadavky jsou vyhledávání předmětů, učitelů a místností, vyhledávání volných místností, vyhledávání nejbližších událostí a řazení výsledků dle relevance.

Vyhledávání předmětů, učitelů, místností Prvním požadavkem na prototyp je vyhledávání informací o předmětech, učitelích a místnostech. Jedná se o jeden z nejdůležitějších požadavků aplikace, protože pomáhá studentům k lepší orientaci na škole a rychlému nalezení často potřebných informací. Ke splnění tohoto požadavku lze využít data ze služby KOSapi, která poskytuje základní informace o těchto entitách. Z hlediska uživatele se stačí do systému pouze přihlásit a zadat hledaný údaj. Systém poté vrátí všechny relevantní výsledky související s hledaným údajem.

Vyhledávání volných místností Tento požadavek je velmi užitečný pro všechny studenty na FIT ČVUT, nikoliv hlavně pro nováčky. Výsledkem požadavku je množina volně přístupných místností pro studenty rozdělených podle typu místnosti (seminární či počítačová), které jsou od doby vyhledání po dobu delšího časového úseku (např. 45 minut) volné.

Vyhledávání nejbližších událostí Tento požadavek je součástí předchozích dvou zmíněných požadavků. Při vyhledávání předmětů, učitelů či místností se ke každé z těchto entit zobrazí 2 nejbližší události, které danou entitu čekají. Každá z těchto událostí nese informace o jaký předmět se jedná, od kdy do kdy se událost koná, kdo z učitelů v události figuruje a kde se událost koná.

V rámci vyhledávání volných místností je využito tohoto požadavku ke zjištění, jestli je daná místnost volná či nikoliv.

Řazení výsledků dle relevance Při zobrazování všech výsledků na konkrétní dotaz se může stát, že ve výsledcích budou různé entity (např. předměty s učiteli). V rámci tohoto požadavku je řešeno řazení těchto dat dle relevance na základě dotazu, tedy přiřazení určitého čísla každému výsledku a následné seřazení výsledků podle těchto čísel sestupně. Toto číslo bude generováno např. na základě počtu shod ve slovech názvu/jména či shoda nějakého slova s kódem/uživatelským jménem. Díky tomuto požadavku tedy bude zajištěno, že žádný druh/typ entity nebude vždy upřednostňován před druhým a že uživatel nalezne žádaný výsledek rychleji.

2.3.2 Nefunkční požadavky

Nefunkční požadavky jsou všechny požadavky, které nejsou funkční. Mezi takové požadavky patří například:

- vzhled (GUI),
- technologie,
- výkonnost,
- cílová skupina,
- zjednodušení práce se systémem/systémy.

Mezi hlavní nefunkční požadavky aplikace patří zejména zjednodušení práce s již existujícími informačními systémy a aplikacemi FIT ČVUT. Dále lze vymezit cílové skupiny uživatelů aplikace (studenti a učitelé). Od aplikace je také vyžadována určitá výkonnost, aby vyhledávání nezabralo příliš mnoho času. To může být splněno například cachováním či jinou formou ukládání dat a výsledků, protože získávaná data se často nemění a díky cachování budou výsledky zobrazeny rychleji. Cachování je důležité také z toho důvodu, aby aplikace zbytečně nezahlcovala služby fakulty. Po výkonnosti je také vyžadována bezpečnost. V rámci bezpečnosti je nutné autorizovat a autentizovat uživatele vyhledávacího systému, aby k datům nezískal přístup uživatel mimo ČVUT. Posledním velmi důležitým nefunkčním požadavkem, který se od aplikace očekává, je jednoduchá rozšiřitelnost, díky které bude jednoduché aplikaci rozšířit o další webové služby či entity bez nutnosti úprav již napsaného kódu.

2.4 Analýza procesů

Procesy softwaru lze chápat jako sadu operací, které vedou ke splnění určitého cíle či požadavku. Z hlediska WhereIS lze vytyčit tři nejdůležitější procesy aplikace:

- získání a případné zpracování a uložení dat,
- přihlášení uživatele (tedy jeho autentizace a autorizace),
- vyhledávání informací a jejich řazení dle relevance.

2.4.1 Získání a uložení dat

Z hlediska uživatele a samotného používání aplikace není požadována žádná interakce navíc, jelikož se tento proces již stal nebo se děje během vyhledávání na pozadí.

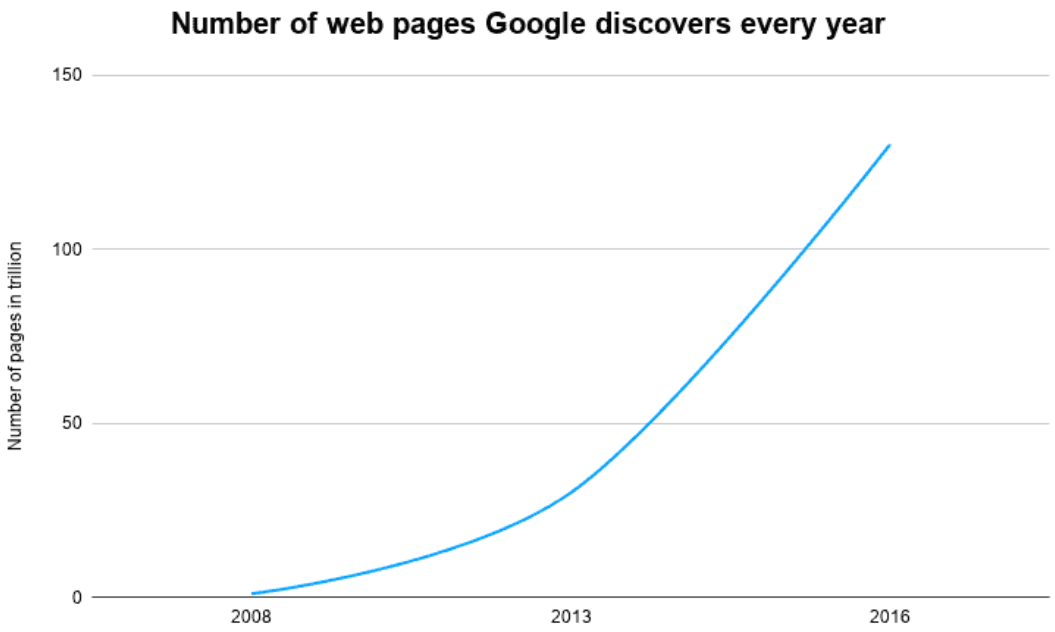
Statická data, která jsou velká a je u nich málo pravděpodobné, že dojde k jejich změnám, je lepší předem ze zdrojů získat, zpracovat a uložit. Vyhledávání nad již staženými, připravenými daty v databázi či v souborech na disku je rychlejší, než jejich neustálé stahování. Tento proces lze také implementovat tak, že se bude provádět automaticky např. vždy začátkem dalšího dne nebo začátkem nového semestru.

2.4.2 Přihlášení uživatele

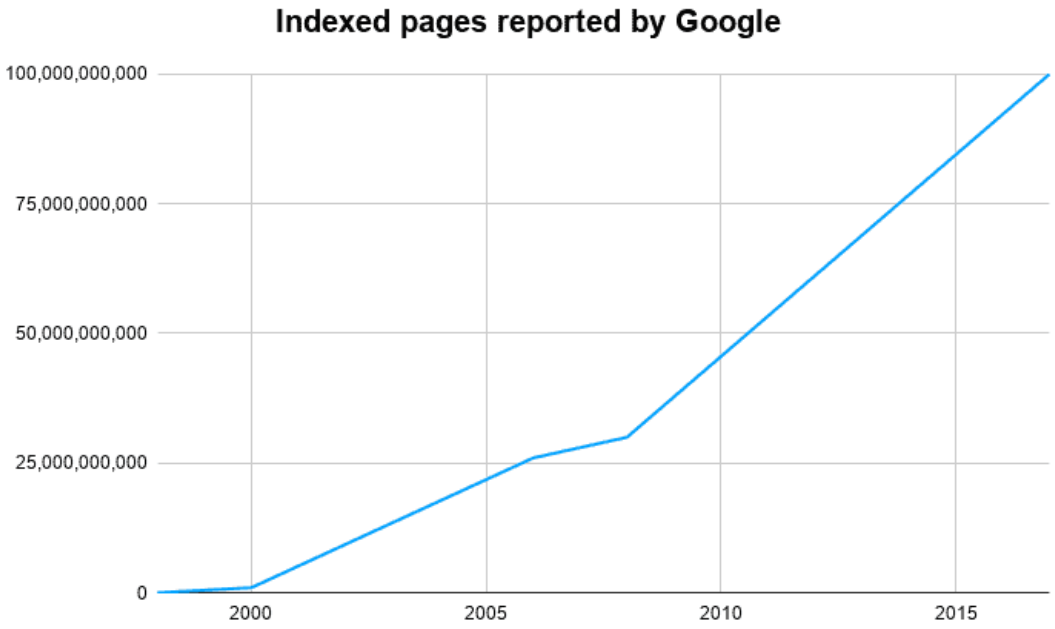
Přihlášení uživatele se od ostatních aplikací a systémů ČVUT nijak neliší. Pro přihlášení je potřeba mít fungující ČVUT účet. Při otevření webové stránky aplikace WhereIS klikne uživatel na tlačítko „Přihlásit“, poté bude přesměrován na autorizační server ČVUT, kde zadá přihlašovací údaje ke svému účtu ČVUT. Pokud uživatel aplikaci WhereIS používá poprvé, nebo došlo v rámci aplikace k úpravám vyžadovaných *scopes*, musí uživatel vyjádřit souhlas aplikaci, že aplikace může používat uživatelovy scopes (více v sekci 3.1.2.1). Po úspěšném přihlášení a případném vyjádření souhlasu je uživatel přesměrován na vyhledávací stránku systému WhereIS.

2.4.3 Vyhledávání informací

Jedná se o nejdůležitější proces ze strany uživatele i aplikace. Uživatel pomocí jednoho či více klíčových slov vyhledává potřebné informace o učitelích, předmětech, místnostech či aktuálně volných místnostech. Na velikosti písmen zadávaného dotazu nezáleží, na diakritice ano. Po zadání všech chtěných klíčových slov dojde k zobrazení všech relevantních výsledků, které souvisí s hledanými slovy. Tyto výsledky jsou navíc uživateli seřazeny podle relevance, tedy výsledky více nahoře budou více relevantní než výsledky pod nimi. Uživatel v době psaní této práce nemá možnost definovat si svůj vlastní způsob řazení výsledků.

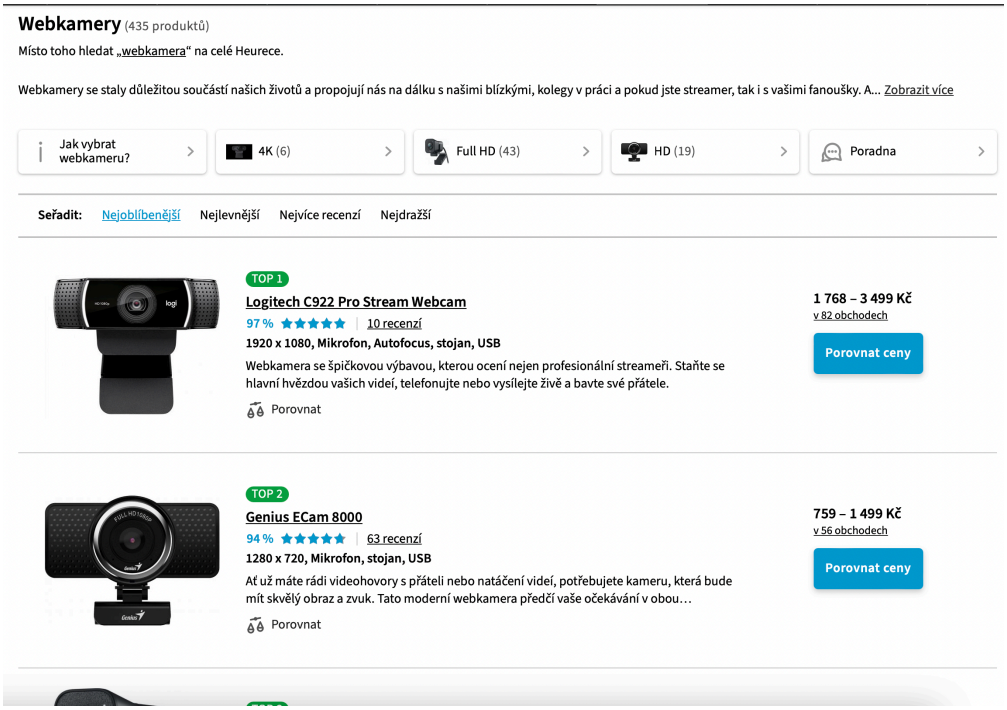


(a) Počet webových stránek, které Google objevil v daný rok

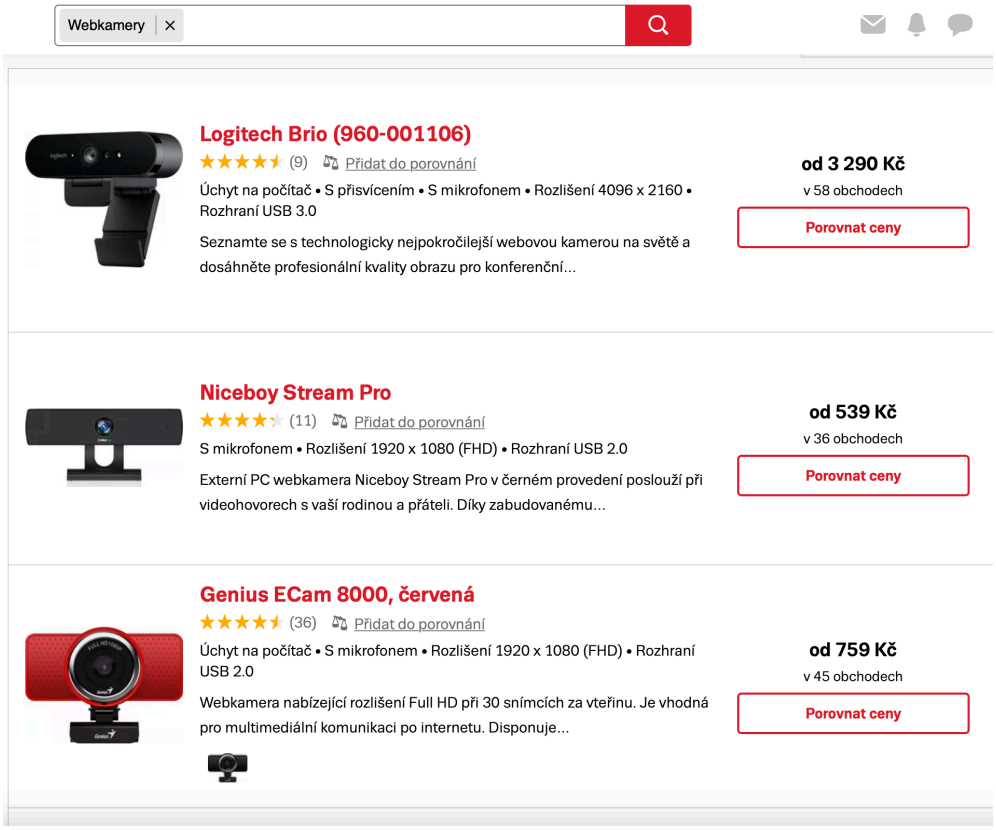


(b) Počet webových stránek, které Google indexoval v daný rok

Obrázek 2.1 Grafy srovnávají počet objevených a počet indexovaných webových stránek společností Google, převzato z [4]



(a) Heureka — webkamery



(b) Zboží.cz — webkamery

Obrázek 2.2 Porovnání výsledků vyhledávání agregátorů Heureka a Zboží.cz

Obrázek 2.3 Novinové články o Johnnym Deppovi jako výsledek agregace na Google Search

Obrázek 2.4 Box agregující informace o Johnnym Deppovi jako výsledek agregace na Google Search

Obrázek 2.5 Videa související s Johnnym Deppem jako výsledek agregace na Google Search

Kapitola 3
Návrh a implementace

3.1 Technologie

Součástí návrhu nové aplikace je nutné zvolit, jaké technologie bude aplikace využívat, případně mezi jakými technologiemi měl vývojář na výběr a které z nich nakonec zvolil. V rámci této kapitoly popisuji použité a potřebné technologie pro správné fungování vyhledávacího systému WhereIS a diskutuji případné možnosti v jejich volbě.

3.1.1 REST

REST, jak uvádí Richards v [9], je architektonický styl využívající běžné technologie a protokoly, které jsou v tomto případě použity k implementaci a využití RESTové webové služby. Tento architektonický styl byl navržen v roce 2000 Royem Fieldingem v jeho disertační práci [10]. V této práci Fielding píše o tom, že každá informace či koncept na internetu, který může být pojmenován (tzv. *resource*, zdroj), je identifikován svým identifikátorem (tzv. *resource identifier*), jako je například URI. URI je jednoznačný identifikátor objektu na internetu, který může být buď URL, nebo URN (viz obrázek 3.1). Poté různé komponenty v této architektuře provádí různé akce na zdroje použitím své reprezentace. Například použijeme webový prohlížeč na vyžádání zdroje, server předá aktuální stav webové stránky prohlížeči. Prohlížeč je pak schopný provést akci vykreslení dané reprezentace. Akce na tyto zdroje lze určovat pomocí HTTP metod `GET`, `HEAD`, `POST`, `PUT`, `DELETE`. Pichlík v [11] zmiňuje, že v dnešní době se tyto HTTP metody používají k realizaci CRUD[1] operací nad každým zdrojem, které slouží k manipulaci se zdroji. Navíc každý zdroj může mít jiné reprezentace (JSON, XML, PDF…). REST je v současné době velmi populární a hojně využívaný nástupce SOAP.

3.1.2 OAuth 2.0

Jak je řečeno v [12], OAuth 2.0 je standardizovaný autorizační framework (protokol) využívaný k autorizaci uživatelů a klientů. Tento framework umožňuje dočasný a limitovaný přístup k zabezpečeným webovým službám. Singh v [13] definuje framework (v českém jazyce *rozhraní*) jako pomocnou úroveň abstrakce pro programátora, která nabízí řadu implementovaných funkcionalit na nižších úrovních, o které se programátor nemusí starat.

Na FIT ČVUT se OAuth 2.0 používá zejména pro zabezpečení RESTových webových služeb a webových aplikací. FIT ČVUT má svůj vlastní autorizační server OAuth 2.0 s názvem

[1] Create, Read, Update, Delete

Zuul OAAS, který je open-source. Tento server běží na adrese `https://auth.fit.cvut.cz`, kde dochází k veškeré autorizaci.

3.1.2.1 Autorizační povolení (authorization grants)

Pro přístup k datům ze systémů Sirius, KOSapi a Umapi se potřebuje aplikace autorizovat právě pomocí protokolu OAuth 2.0 a získat *access token*, díky kterému získá určitá práva ve službách FIT ČVUT. V [12] je *access token* definován jako řetězec znaků sloužící jako přístupové (přihlašovací) údaje k chráněnému zdroji. Tokenem jsou určeny oblasti (tzv. *scopes*), do kterých má daný uživatel/klient přístup, a dobu trvání tohoto přístupu (tzv. *duration of access*).

K tomu je potřeba také autentizovat a autorizovat samotné uživatele, kteří budou klienta používat. K tomuto účelu slouží *autorizační povolení* (tzv. *authorization grants*). Tato povolení se od sebe liší např. potřebnými údaji k autorizaci nebo samotným použitím (autorizace klienta, uživatele…).

První typ autorizačního povolení je *client credentials*, který slouží k autorizaci klienta (aplikace). Zde se nerozlišují práva jednotlivých uživatelů, kteří klienta používají, ale pouze práva klienta samotného. K autorizaci dojde zadáním *client id* a *client secret* a po úspěšné autorizaci vrátí autorizační server *access token*. Zjednodušeně lze tento typ chápat jako způsob autorizace klienta jako uživatele vůči serveru (stroj se autorizuje stroji).

Druhý potřebný typ autorizačního povolení je *authorization code*. Jak je popsáno v [12], tento typ povolení slouží k autorizaci samotného uživatele a vyžaduje uživatelské jméno — nejčastěji přezdívka nebo e-mailová adresa — a heslo. Přihlášení uživatele probíhá opět skrz autorizační server, nikoliv přes samotný server či zdroj, ke kterému se uživatel snaží získat přístup. Po úspěšném přihlášení uživatele dojde k přesměrování na vývojářem specifikovanou stránku (nejčastěji hlavní stránku serveru/zdroje). V URL adrese přesměrování se nachází *code*, který slouží k autentizaci a autorizaci uživatele vůči dalším webovým službám (podobně jako *access token*), které jsou klientem používány, nebo také k přenosu *access tokenu* přímo samotnému klientovi bez rizika odposlechu. Kromě *code* se v URL adrese nachází také *state*. Tento řetězec slouží k určení, jestli se jedná o pravého aktuálně autentizovaného a autorizovaného uživatele, či nikoliv. Před přesměrováním uživatele na autorizační server dojde k vygenerování unikátního *state* na straně aplikace a tento řetězec se pošle autorizačnímu serveru v URL adrese přesměrování. Po úspěšné autorizaci pošle autorizační server tento řetězec zpět, aby mohla aplikace ověřit, že se jedná o stejného uživatele a že *code* nezískal někdo jiný. Tento typ autorizace slouží tedy k rozlišení práv přihlášených uživatelů.

OAuth 2.0 nabízí ještě další typy autorizačního povolení, jak je zmíněno v [12]. Mezi nimi je například *implicit*, což je zjednodušený a optimalizovaný proces autorizace pro klienty implementovaných ve webových prohlížečích, které využívají skriptovací jazyky jako je *JavaScript*. Další je *resource owner password credentials*, který může být použit přímo jako typ autorizačního povolení k získání *access tokenu*. Používá se zejména v tu chvíli, kdy je mezi stranami vysoká důvěryhodnost.

Poslední věc, kterou je potřeba v rámci této podkapitoly zmínit, je *refresh token*. Refresh token je částečně typem autorizačního povolení, nedá se totiž použít k prvotní autorizaci. V [12] uvádí, že tento token je poskytnut klientovi autorizačním serverem a slouží jako přístupové údaje k získání nového *access tokenu* poté, co *access token* vyprší nebo je nevalidní.

3.1.3 Apps Manager

Apps Manager je webová aplikace, díky které mohou vývojáři získat přístup k webovým službám FIT ČVUT. V této aplikaci si vývojář může zažádat o vytvoření client id a client secret pro typ povolení client credentials. Dále může specifikovat přesměrování pro typ povolení authorization code, kam bude uživatel po úspěšné autentizaci a autorizaci přesměrován. Navíc lze explicitně určovat služby, ke kterým bude mít uživatel/aplikace přístup a i konkrétní práva (čtení, zápis,

mazání). Práva pro čtení jsou ve většině služeb dostupná automaticky a lze je ihned zapnout/-vypnout. Ostatní práva je však nutné si prostřednictvím aplikace vyžádat.

3.1.3.1 Návod k získání přístupu k datům/API

Tento návod slouží vývojářům, kteří chtějí získat přístup ke službám FIT ČVUT. Popisuje získání *access tokenu* pomocí typu povolení client credentials.

1. Přihlaste se svým účtem ČVUT v aplikaci Apps Manager[2].
2. Na této stránce si vytvořte nový projekt (nejlépe s názvem vaší aplikace).
3. Vlevo klikněte na růžový papír (stránka „Apps“).
4. Na této stránce klikněte na „Create next App“. V této části můžete specifikovat typ aplikace (povolení), který potřebujete. Pro účely tohoto návodu zvolte „Service Account“ (typ client credentials).
5. Po vytvoření aplikace na stránce uvidíte vygenerované client id a client secret.
6. Nyní stačí použít jednoduchý HTTP klient k získání *access tokenu*. V požadavku vyplňte client id, client secret a typ povolení. Viz 3.1.

Výpis kódu 3.1 Získání access tokenu pomocí HTTP klienta Guzzle v PHP

```
use GuzzleHttp\Client;

$client = new Client();
$response = $client->post('https://auth.fit.cvut.cz/oauth/token', [
            'form_params' => [
                'grant_type' => 'client_credentials'
            ],
            'auth' => [$CLIENT_ID, $CLIENT_SECRET]
        ]);
$responseContent = json_decode($response->getBody()->getContents(),
                    true);
return $responseContent['access_token'];
```

7. Nyní pouze stačí použít *access token* při získávání dat z webové služby. Viz 3.2.

Výpis kódu 3.2 Získání dat přes Umapi pomocí HTTP klienta Guzzle v PHP

```
use GuzzleHttp\Client;

$client = new Client();
$response = $client->get(
            "https://kosapi.fit.cvut.cz/usermap/v1/people/xvagner", [
                'headers' => [
                    'Authorization' => 'Bearer ' . $accessToken
                ]
            ]);
$data = json_decode($response->getBody()->getContents(), true);
```

3.1.4 Programovací jazyky

Ve volbě programovacích jazyků mají v dnešní době vývojáři velkou volnost. Pro vývoj webové aplikace WhereIS lze vybírat mezi 4 nejoblíbenějšími a nejpoužívanějšími programovacími jazyky.

[2] https://auth.fit.cvut.cz/manager/user/home.xhtml

PHP Jak je řečeno na oficiální stránce [14], PHP je open source dynamicky typovaný skriptovací jazyk, který se používá zejména pro vývoj webových stránek a může být přímo zakomponován do HTML. Díky dynamickému typování není nutné předem uvádět, o jaké datové typy se jedná, interpreter toto vyřeší za programátora. Interpreter je program, který provádí dané instrukce/příkazy okamžitě bez nutnosti překladu kódu. Čím se PHP například od JavaScriptu na straně klienta liší, je fakt, že PHP kód je spuštěn na straně serveru, nikoliv na straně klienta. Vygenerovaná HTML stránka je pak klientovi poslána, ale klient není schopný vidět, jaký kód danou stránku vygeneroval. Jeho největší výhodou je vysoká úroveň abstrakce, kdy programátor nemusí řešit nízkoúrovňové záležitosti aplikace, a je velmi přívětivý a jednoduchý pro začátečníky. Pro složitější aplikace lze využít balíčkovacího nástroje určeného pro PHP s názvem *Composer*. Balíčkovací nástroj je program, který se stará o stahování a správu závislostí (balíčků, knihoven) používaných v projektu. Pro PHP existuje také řada frameworků, kterých lze využít při psaní webových aplikací. Mezi nejznámější patří framework *Symfony*, *Laravel* nebo český framework *Nette*. Největší nevýhodou tohoto jazyka je bezpečnost. PHP je open source (otevřený zdrojový kód), takže jej může kdokoliv upravovat a do zdrojového kódu může kdokoliv nahlížet, je to dynamicky typovaný jazyk a je interpretován za běhu. To znamená, že případné chyby se mohou objevit až v době používání aplikace a nelze je odchytit během překladu.

Python Jak je řečeno v oficiální dokumentaci [15], Python je interpretovaný objektově orientovaný vysokoúrovňový programovací jazyk, který je navíc dynamicky typovaný. Podobně jako PHP je Python relativně jednoduchý na naučení a je tedy dobrou volbou pro začátečníky. Vývojáři v Pythonu mohou využívat velkého množství modulů a balíčků, díky kterým bude psaní jejich aplikací či skriptů jednodušší. Ohledně balíčkovacích nástrojů mají Python vývojáři na výběr hned ze 2 oblíbených možností: *Conda* a *Pip*. Mezi největší nevýhody Pythonu patří výkonnost. Jak je řečeno v článku [16], Python je pomalejší než jiné programovací jazyky (např. C++, Java, PHP…), což je pro programátory nevhodné zejména při psaní obsáhlých programů a skriptů. Další nevýhodou je velké využití paměti při běhu kvůli flexibilitě datových typů (dynamické typování).

JavaScript Jak je popsáno v [17], JavaScript je vysokoúrovňový interpretovaný dynamicky typovaný skriptovací jazyk, který se používá zejména na skriptování na webových stránkách a ve webových aplikacích. V současné době se tento jazyk rozšiřuje i mimo internetové prostředí, používá se například v *Node.js* nebo v *Adobe Acrobat*. JavaScript je konkrétní realizací *ECMAScript*, což je specifikace ECMA skriptovacích jazyků. JavaScript je vhodný pro začínající programátory díky své jednoduchosti, je také multiplatformní, protože běží ve webovém prohlížeči, který lze v dnešní době nainstalovat na každý operační systém. Díky tomu, že JavaScript běží v prohlížeči, je úzce spjatý s *DOM*. DOM lze chápat jako strom objektů, ze kterých se skládá (X)HTML nebo XML dokument, nebo jako API těchto dokumentů. Mezi nevýhody JavaScriptu v [18] řadí například různou kompatibilitu napříč webovými prohlížeči. Každý dnes používaný webový prohlížeč podporuje JavaScript, nicméně kompatibilita různých funkcí tohoto jazyka se v každém prohlížeči liší. Další nevýhodou je spouštění JavaScriptového kódu na straně klienta, kde může dojít k zahlcení pomalejšího počítače klienta nebo k ohrožení bezpečnosti v případě spouštění škodlivého kódu.

Java Jak je popsáno v [19], Java je objektově orientovaný třídně založený programovací jazyk designovaný za účelem menšího množství implementačních závislostí. Jedná se o spolehlivý a bezpečný programovací jazyk. Java na rozdíl od předchozích zmíněných jazyků není dynamicky typovaný jazyk, ale staticky typovaný — datové typy se u proměnných musí vždy uvádět. Další výhodou tohoto jazyka je fakt, že Java běží v *JVM*, což lze chápat jako virtuální prostředí, ve kterém je kód spuštěn. Díky tomu je možné spustit programy napsané v Javě na všech zařízeních, na kterých je možné JVM nainstalovat. To také znamená, že je platformě nezávislá. Na rozdíl od programovacího jazyka C/C++, který se kompiluje do strojového kódu, se Java kompiluje

do *bajtkódu* (v anglickém jazyce *bytecode*). Bajtkód lze chápat jako seznam instrukcí, které se mají efektivně provést. Mezi nevýhody Javy podle [20] patří například rychlost. Ve srovnání se zmíněným C/C++ Java značně zaostává. Další nevýhodou je fakt, že při psaní aplikací v Javě vznikají často relativně komplexní a hůře čitelné kódy. Java sice přináší vyšší úroveň abstrakce, než například C/C++ (proto se tento jazyk na psaní webových aplikací nepoužívá), ale tato úroveň není tak velká na to, aby vznikaly „pěkné“ kódy jako například v PHP nebo Pythonu.

Shrnutí Z výše uvedených jazyků použiji skriptovací jazyk PHP. JavaScript není vhodným jazykem pro psaní větších webových aplikací z důvodu nutnosti řešení kompatibility napříč webovými prohlížeči a spouštění kódu na straně klienta. Java neobsahuje dostatečně velkou úroveň abstrakce, díky které by se aplikace psala snáz a kód by byl lépe udržovatelný a čitelný. Python je s PHP v této volbě velmi srovnatelný. Důvod, proč nevolím jazyk Python, je menší množství dostupných frameworků pro psaní webových aplikací a slabší výkonnost ve srovnání s PHP. Navíc hostingové služby pro PHP jsou zpravidla levnější než hostingy jiných programovacích jazyků a PHP k tomu disponuje mnoha dlouhodobě vyvíjenými frameworky (více v kapitole 3.1.5), které řeší některé problémy s bezpečností.

3.1.5 Frameworky

Vzhledem k tomu, že pro vývoj vyhledávacího systému WhereIS byl zvolen programovací jazyk PHP, popisuji v této části několik frameworků. Vzhledem k rozšířenosti jazyka PHP a opravdu velkému množství různých frameworků následující sekce popisuje tři a to Symfony, Laravel a Nette.

Symfony V [21] uvádí, že Symfony je open source framework pro PHP, který je založený na MVC návrhovém vzoru. Tento návrhový vzor slouží zejména k oddělení logické vrstvy aplikace od prezenční. Podporuje také CLI, což znamená, že typické úkony lze provádět a nastavovat pomocí příkazové řádky. Samotný projekt je po vygenerování strukturován do příslušných složek, z jejichž názvů je jasné, co do které složky patří. Další výhodou je, že je multiplatformní, podporuje více operačních a databázových systémů. Z hlediska ORM, což je mapování tříd v kódu do entit (tabulek) v databázi (1 objekt = 1 řádek v tabulce), podporuje Symfony v základním nastavení *Doctrine*, což implementuje návrhový vzor *Data Mapper*. Data Mapper je návrhový vzor, který se používá pro perzistenci dat. Má oddělenou datovou a logickou vrstvu, tedy objekt uchovává pouze svá data a o logiku ukládání dat do databáze se stará jiný nezávislý objekt (tzv. *mapper*). Pro šablonování HTML stránek používá Symfony nástroj *Twig*. Co se týče bezpečnosti, tak Symfony je velmi silný framework, nicméně veškeré její nastavení je relativně komplikované. Další nevýhodou je, že ve srovnání s relativně podobným frameworkem Laravel je Symfony ve výkonnostních testech pomalejší.

Laravel V článku [21] říkají, že Laravel je také open source framework pro PHP založený na MVC návrhovém vzoru. Také podporuje CLI a po vygenerování projektu vznikne opět patřičná struktura projektu, ze které je vypovídající, co kam patří. Stejně jako Symfony i Laravel je multiplatformní a podporuje řadu operačních a databázových systémů. Laravel v rámci ORM podporuje v základním nastavení *Eloquent*, který je založen na návrhovém vzoru *Active Record*. Active Record je návrhový vzor používaný pro perzistenci dat a je přesným opakem Data Mapperu. Active Record totiž spojuje datovou vrstvu a logiku ukládání dat, takže 1 objekt v rámci kódu reprezentuje i 1 řádek v tabulce v rámci databáze, protože objekt má metody pro práci s databází. Pro šablonování HTML stránek používá Laravel nástroj *Blade*. Ohledně bezpečnosti Laravel poskytuje jednodušší přístup, než Symfony. To znamená, že míra bezpečnosti je zde menší, než u Symfony, ale její nastavení je o to snazší. Jak už bylo řečeno v odstavci o Symfony, Laravel je ve výkonnostních testech rychlejší, než Symfony.

Nette Jak je uvedeno na webu frameworku [22], Nette je český open source framework pro PHP založený na MVC návrhovém vzoru. Také podporuje CLI a po vygenerování projektu vznikne opět předdefinovaná struktura projektu. Stejně jako předchozí frameworky i Nette je multiplatformí a podporuje řadu operačních a databázových systémů. V rámci ORM lze v Nette pomocí různých rozšíření (např. *Nettrine*) použít nástroj *Doctrine* a tedy i návrhový vzor *Data Mapper*. Pro šablonování používá Nette český nástroj *Latte*. Nette disponuje zejména velkou bezpečností a jednoduchým a silným nástrojem na debugování s názvem *Tracy*. Jednou z nevýhod Nette je fakt, že není tak rozšířený jako Laravel či Symfony, takže komunita a celková podpora frameworku není tak silná jako u předchozích frameworků. Na druhou stranu je tato komunita tvořena zejména Čechy a Slováky, takže pokládání dotazů a řešení problémů bude jednodušší, než v případě zahraničních komunit.

Shrnutí Z výše uvedených frameworků použiji Symfony, protože i přes menší výkonnost ve srovnání s Laravelem poskytuje větší možnosti konfigurace, zejména co se týče bezpečnosti, a v základu poskytuje návrhový vzor Data Mapper, který je dle mého názoru pro ORM lepší a přehlednější, než Active Record. Obzvláště s ohledem na další rozšiřování projektu. Nette nevolím z toho důvodu, že není tak rozšířené a v komunitě je méně používané, a tedy i samotná podpora frameworku Nette je menší, než například u Symfony či Laravelu.

3.1.6 Databázové systémy

Pro správné fungování vyhledávacího systému WhereIS není nutné používat databázový systém. Všechna data se dají získat v moment dotazu uživatele ze zdrojů. Nicméně z hlediska výkonnosti aplikace a potřeby nezahlcení webových služeb a aplikací FIT ČVUT je použití databáze žádané.

Jak bylo řečeno v popisu procesu 2.4.1, statická data, která jsou velká a nemění se moc často či vůbec, je vhodné předem zpracovat a uložit. Pro ukládání těchto dat lze využít databázový systém. Naopak malá data a data, která se často mění, je lepší ze zdrojů získávat vždy při každém vyhledávacím dotazu. Režie na získání a zpracování těchto dat bude menší, než kdybychom tato data vždy získávali, zpracovávali, upravovali v databázi a poté nad nimi vyhledávali. V tomto ohledu lze pro efektivnější fungování aplikace využít *cachování*. *Cache (keš či mezipaměť)* je takový typ paměti, do které se dočasně ukládají zpravidla nejčastěji hledaná nebo nejnovější data za účelem rychlejšího dodání těchto dat. Z hlediska webových aplikací se *cache* hodí zejména na ukládání odpovědí ze serveru či jiných aplikací, aby se aplikace na stejná data nemusela opakovaně dotazovat.

Meire a Kaufmann ve své knize [23] rozdělují databázové systémy na 2 typy a to *SQL* a *NoSQL*. V této knize také popisují rozdíly mezi těmito databázovými systémy a na co se jaký typ nejvíce hodí. Tyto 2 typy jsou podrobněji popsány v následujících paragrafech.

SQL SQL databázové systémy jsou tradiční relační databázové systémy, které využívají *SQL*, což je dotazovací jazyk pro dotazování nad daty a pro práci s nimi. Základním principem relačních databázových systémů je používání tabulek a relací k jejich propojení a vyjádření vztahu mezi nimi. Jednu tabulku v tomto databázovém systému lze chápat jako jednu entitu. Tyto databázové systémy jsou vhodné v tu chvíli, kdy nepracujeme s velkým množstvím dat (tzv. *Big data*) a chceme, aby data v databázi nezabírala zbytečně mnoho místa na disku (díky dekompozici dat do tabulek). Problémem u velkého množství dat v těchto databázových systémech je velká režie na spojování (*joinování*) tabulek, díky kterému získáme z tabulek a relací požadované informace. Nejznámější a hojně používané relační databázové systémy jsou například *PostgreSQL*, *Oracle*, *MySQL* či *SQLite*.

NoSQL NoSQL databázové systémy jsou novějším typem databázových systémů. I přes název těchto systémů používají některé z nich SQL pro dotazování nad daty, standardizovaným dotazovacím jazykem pro tento typ databází, jak zmiňují Kempe a Williams ve svém článku [24],

je *UnQL (unstructured query language)*, který má podobné konstrukty jako SQL a je vhodný pro práci s daty ve formátu JSON. NoSQL databázové systémy jsou vhodné pro práci s velkým množstvím dat a pro rychlé získávání informací z těchto dat. Data nejsou dekomponována do mnoha tabulek za účelem nejnižší duplikace, naopak jsou zde „tabulky“, které obsahují velké množství duplicitních dat za účelem rychlého čtení. Nejznámější NoSQL databázové systémy jsou například *MongoDB* nebo *Cassandra*.

Pro vyhledávací systém WhereIS bohatě postačí SQL relační databázový systém, jelikož s velkým množstvím dat tato aplikace pracovat nebude. K implementaci funkčního prototypu zvolím relační databázový systém *SQLite*. Jedná se o velmi jednoduchou databázi, která nevyžaduje žádnou formu autentizace a autorizace uživatele pro přístup k datům. Při inicializaci se vytvoří 1 soubor, ve kterém jsou všechna data uložena, a celková práce s touto databází je velmi snadná a jednoduchá. Tento databázový systém je proto vhodný pro lokální vývoj a testování. Pro nasazení aplikace do produkce bude nutné zvolit jiný databázový systém, například *PostgreSQL* nebo *Oracle*. Díky návrhu aplikace a využití třívrstvé architektury bude migrace na jiný databázový systém velmi jednoduchá, postačí pár změn v konfiguračních souborech a samotné nastavení databáze, jiný zásah (například do zdrojového kódu) nebude nutný.

3.2 Architektura aplikace

Architektura aplikace popisuje návrh z pohledu aplikace jako celku a z pohledu její funkčnosti. Většina těchto architektur se ještě navíc doplňuje o další potřebné věci. Z pohledu komunikace s aplikací WhereIS je implementována dvouvrstvá architektura klient-server a z pohledu rozdělení tříd je implementována třívrstvá architektura.

3.2.1 Dvouvrstvá architektura

Dvouvrstvá architektura se zabývá pohledem funkčnosti aplikace. Existují dvě základní dvouvrstvé architektury, které se v dnešní době používají, a to *thin-client / smart-server* a *thick-client / dumb-server*.

thin-client / smart-server Tato dvouvrstvá architektura se používá u tradičních aplikací, kde veškerá logika aplikace probíhá na straně serveru a klient se serveru pouze dotazuje, přijímá a zobrazuje výsledky.

thick-client / dumb-server Tato dvouvrstvá architektura reprezentuje moderní přístup dnešních aplikací. Jak už je z názvu patrné, aplikace běží na straně klienta a server slouží pouze pro poskytování dat (zdroj, API). Nevýhodou tohoto přístupu je závislost výkonu aplikace na výkonu zařízení uživatele.

V případě vyhledávacího systému WhereIS zvolím přístup *thin-client / smart-server*, kde klientem bude webový prohlížeč uživatele, který bude sloužit pro dotazování a zobrazování výsledků, a server bude samotný vyhledávací systém WhereIS, ve kterém se bude provádět veškerá logika.

3.2.2 Třívrstvá architektura

Třívrstvá architektura se zabývá pohledem aplikace jako celku. Tato architektura slouží zejména k separaci odpovědnosti, kdy každá vrstva je zodpovědná pouze za jednu věc, a ke snížení závislosti, kdy mezi sebou komunikují vždy jen sousední vrstvy (vyšší vrstva volá nejbližší nižší vrstvu). Existuje také rozvolněná varianta třívrstvé architektury, kdy vrstvy, které mezi sebou

komunikují, nemusí být nejbližší. Třívrstvá architektura se skládá ze tří vrstev a to datové, logické a prezentační.

Datová vrstva Jedná se o nejnižší vrstvu, která se stará o ukládání dat a práci s daty. Odděluje od business logiky aplikace logiku perzistence dat. Součástí této vrstvy je systém/nástroj, který přímo komunikuje s databází (např. *Doctrine*), nebo souborem, do kterého jsou data zapisována, apod. Datovou vrstvu v aplikaci WhereIS reprezentuje právě nástroj *Doctrine* (třídy `Repository` a třída `EntityManager`).

Logická vrstva Jedná se o prostřední vrstvu, která obsahuje veškeré funkcionality aplikace. Zpracovává vstup uživatele a na základě tohoto vstupu mění data v datové vrstvě. Logickou vrstvu v aplikaci WhereIS reprezentují třídy koordinující běh aplikace (`DefaultController` a `SearchController`), servisy (`SearchService`, `CourseItem`, `FreeRoomItem`, `RoomItem` a `TeacherItem`) a samotné entity (`Course`, `Teacher` a `Room`).

Prezentační vrstva Jedná se o nejvyšší vrstvu obsahující tu část aplikace, která je viditelná pro uživatele. Stará se o zobrazování informací, dat, výsledků a zajišťuje přijímaní vstupu uživatele. Je platformě závislá, tedy na různých zařízeních může vypadat jinak (počítač, mobil, tablet…).

Prezentační vrstvu v aplikaci WhereIS reprezentují vygenerované HTML stránky z *Twig* šablon (složka `templates`) s použitím CSS souborům definující vzhled a grafiku stránek (složka `assets/styles`), pohledy (views) reprezentující obálku nad konkrétními daty entity, která se má zobrazit, (`CourseView`, `FreeRoomView`, `RoomView` a `TeacherView`) a třídy koordinující běh aplikace (`DefaultController` a `SearchController`). Tyto 2 třídy patří do obou vrstev, protože určují, jaká šablona se má zobrazit (prezentační vrstva), a volají potřebné metody a zpracovávají vstup uživatele (logická vrstva).

3.3 MVP

Model-View-Presenter (MVP) je návrhový vzor, který mluví o třech vrstvách v aplikaci za účelem oddělení odpovědnosti. Tento vzor je odvozený z návrhového vzoru Model-View-Controller (MVC). Ve srovnání s MVC, jak je uvedeno v článku [25], je presenter v MVP více izolovaný — než controller v MVC — od systému pod ním a nic o něm neví. Model má stejnou zodpovědnost jako v případě MVC. View v MVP zpracovává i akce, které manipulují komponentami systému nebo uživatelským rozhraním, a presenteru přenechává pouze akce, které manipulují se samotnými daty a entitami v modelu. Pokud chce presenter provést vizuální změny uživatelského rozhraní nebo interagovat se systémem pod ním, musí zavolat metodu ve view, která danou akci implementuje. Z těchto důvodů se návrhový vzor MVC více používá v desktopových a mobilních aplikací, zatímco návrhový vzor MVP je více zastoupený ve webových aplikací.

Model Tato vrstva reprezentuje stav aplikace (model), to znamená, že obsahuje samotné entity, repository a další nástroje pro komunikaci s databází. Součástí této vrstvy je také business logika.

V rámci aplikace WhereIS do této vrstvy patří:

- Doctrine
  - Repository
    - `CourseRepository`
    - `TeacherRepository`
    - `RoomRepository`
  - `EntityManager`

- Entity
  - `Course`
  - `Teacher`
  - `Room`
- Service
  - Items
    - `CourseItem`
    - `FreeRoomItem`
    - `RoomItem`
    - `TeacherItem`
  - SearchService

View Tato vrstva reprezentuje pohled na stav aplikace, takže obsahuje pouze pohledy (views), šablony, HTML stránky a grafickou stránku těchto stránek.

V rámci aplikace WhereIS do této vrstvy patří:

- Šablony (templates)
  - Items
    - `course`
    - `event`
    - `free_room`
    - `room`
    - `teacher`
  - `base`
  - `index`
  - `search`
  - `search_result`
- Pohledy (views)
  - `CourseView`
  - `FreeRoomView`
  - `RoomView`
  - `TeacherView`

Presenter Tato vrstva zprostředkovává pohled na stav aplikace a obsahuje logiku interakce s uživatelem, tedy zpracovává vstup uživatele.

V rámci aplikace WhereIS do této vrstvy patří:

- Třídy koordinující běh aplikace (controller)
  - `DefaultController`
  - `SearchController`

3.4 Další části aplikace

Součástí aplikace WhereIS jsou i takové části aplikace, které svou povahou přímo nepatří do žádné vrstvy. Jedná se konkrétně o část starající se o získání, zpracování a uložení statických dat do databáze a o část komunikující se zdroji při vyhledávání.

Získání, zpracování a uložení statických dat Jedná se o oddělenou část aplikace, která není přímou součástí webové aplikace WhereIS. Je realizována pomocí *Symfony Command*, jehož úkolem je získat a uložit data o učitelích, místnostech a předmětech do databáze. Data jsou získána přes KOSapi. Tento proces je možné v budoucnu rozšířit tak, že se bude spouštět automaticky sám po nějaké době (po jednom dnu, na začátku semestru…).

Tuto část aplikace reprezentuje několik tříd, mezi nimi je třída `FetchDataCommand` ve složce `Command`. Tato třída obsahuje implementaci *Symfony Command*, aby mohl být tento proces volán jako příkaz z konzole. Další třída je `FetchManager` ve složce `FetchManager`. Tato třída se stará o postupné získání a uložení dat o jednotlivých entitách, které má v sobě zaregistrované, a drží si informace o tom, která data se podařilo uložit a která ne. Poslední částí tohoto procesu jsou třídy `CourseFetcher`, `RoomFetcher` a `TeacherFetcher` ve složce `Utils/Fetcher`, které se starají o získání, zpracování a uložení dat o stejnojmenných entitách. Všechny tyto třídy musí implementovat rozhraní `ItemFetcher` ve stejné složce.

Proces se spouští příkazem `php bin/console fetch` z konzole ve složce projektu.

Dotazování na zdroje při vyhledávání Při vyhledávání uživatelem požadovaných informací dochází často k posílání požadavků na různé webové služby FIT ČVUT, například při zobrazování dvou nejbližších událostí pro předmět, učitele (chybějící práva) a místnost dojde k poslání požadavku na službu *Sirius*, které vrátí 20 událostí od dne vyhledávání, a z této odpovědi dojde k nalezení dvou aktuálně nejbližších událostí pro danou entitu. Stejný proces se děje při vyhledávání volných místností, akorát se zjišťuje, jestli nejbližší událost místnosti začíná až za aktuální čas + určitý čas (aktuálně nastaveno na 45 minut). K dalšímu takovému dotazování dochází u učitelů, kdy se aplikace dotazuje přes *Umapi* na informace o učiteli (kancelář, katedra, kontaktní údaje…). O všechna tato dotazování a zasílání požadavků na služby FIT ČVUT se starají třídy `SiriusResource`, `UserMapResource` a `ZuulOAASResource` ve složce `Service/Resource`. V této složce je také třída starající se o *cachování* výsledků získaných z těchto služeb s názvem `CacheResource`.

Součástí všech těchto procesů je autorizace aplikace WhereIS přes autorizační server školy *ZuulOAAS*, který vrátí aplikaci *access token* (viz sekce 3.1.2).

3.5 Bezpečnost

Bezpečnost je v dnešní době důležitým aspektem každé aplikace, u webových aplikací to platí obzvlášť. Některé prvky bezpečnosti jsou poskytovány automaticky použitými technologiemi, například většina databázových systémů vyžaduje autorizaci a autentizaci uživatelů, frameworky nabízí ochranu proti XSS, CSRF, poskytují vývojáři nástroje pro jednodušší implementaci autorizace a autentizace uživatelů atd. V této kapitole je probrán útok XSS, autorizace a autentizace uživatelů, validace formulářů a zabezpečené připojení pomocí protokolu HTTP.

Cross Site Scripting (XSS) XSS, jak je uvedeno v [26], je typ útoku, kdy je škodlivý kód (skript) vložen na stránku, které uživatelé důvěřují. K útoku dojde v tu chvíli, kdy útočník použije webovou aplikaci k poslání škodlivého kódu (nejčastěji v podobě skriptu, který prohlížeč spustí) koncovým uživatelům aplikace. Webový prohlížeč uživatele není schopný rozeznat, zda se jedná o skripty původní stránky, nebo o skripty podstrčené, a proto veškeré obdržené skripty

provede. Díky tomu se pak může útočník dostat ke všem *cookies* souborům, *session tokenům* či jiným citlivým informacím a je pak schopný se například za uživatele vydávat. Útok je graficky popsán na obrázku 3.2.

Jak už bylo zmíněno na začátku sekce o bezpečnosti, ochranu proti takovému útoku poskytuje samotný framework Symfony prostřednictvím automatického escapování nevhodného vstupu prostřednictvím šablonovacího systému Twig, který aplikace WhereIS používá.

Autentizace a autorizace Pro autentizaci a autorizaci není potřeba téměř nic implementovat, protože vše za aplikaci vyřeší OAuth 2.0 a autorizační server ČVUT *ZuulOAAS*. Jediné, co je nutné zařídit pro úspěšnou autorizaci a autentizaci, je přesměrování na autorizační server a následné přesměrování zpět na vyhledávací stránku v aplikaci, kdy po přesměrování dojde ke kontrole, jestli se jedná o opravdu aktuálně autentizovaného uživatele, nebo obdržený *code* byl podvrh. Poslední nutností pro správné fungování je zabezpečení všech stránek (krom úvodní, kde se uživatel může přihlásit) tak, aby se nepřihlášený uživatel nemohl na tyto stránky dostat. To je vyřešeno kontrolou v controllerech, kde se před zobrazením požadované stránky zkontroluje, zda je uživatel přihlášený. Následně je ještě nutné umožnit uživateli odhlásit se.

Validace formulářů Jelikož formuláře představují jeden ze vstupů webových aplikací, kam můžou uživatelé zadávat (pokud není ošetřeno) co chtějí, představují nebezpečí, jelikož se dají využít na útok proti aplikaci a například tak získat citlivá data z databáze, nebo data z databáze smazat. Z těchto důvodů je nutné uživatelský vstup ve formulář escapovat a kontrolovat na straně serveru, aby nedošlo k takovému zneužití a útoku.

Framework Symfony escapování a kontrolu formulářů na straně serveru nabízí a provádí ji, tedy aplikace WhereIS je proti tomuto zneužití chráněna.

Zabezpečené připojení (HTTPS) Po úspěšné autentizaci a autorizaci uživatele dojde k přesměrování zpět na webovou aplikaci WhereIS, kde se v odkazu přesměrování nachází autentizační kód uživatele. Z důvodu přenosu těchto citlivých dat je nutné, aby webový server, na kterém WhereIS běží, podporoval zabezpečené připojení pomocí protokolu HTTPS a k tomu je nutné, aby používal vhodný a bezpečný certifikát. Samotná aplikace je automaticky nastavena na používání protokolu HTTPS. Protokol HTTPS je zabezpečená verze protokolu HTTP.

3.6 Integrace nových entit

V požadavcích na návrh aplikace byla požadována jednoduchost rozšiřitelnosti aplikace o nové entity a služby bez nutnosti úprav stávajícího kódu. Nevyhneme se vytváření nových objektů, které budou potřeba na integraci nových entit, nicméně by jich nemělo být velké množství a s využitím frameworku Symfony dojde k vytvoření některých objektů za nás.

Pokud budeme chtít, aby se data o nové entitě z dané služby ukládala do databáze, využijeme frameworku Symfony na vytvoření třídy entity, repository a samotné tabulky entity v databázi. Vytváření entity spustíme konzolovým příkazem v hlavní složce projektu `php bin/console make:entity`. Poté už nás návod provede samotným vytvářením entity. Po vytvoření entity je potřeba změny propsat do databáze, toho docílíme příkazy `php bin/console make:migration` a `php bin/console doctrine:migrations:migrate`. Poslední příkaz `php bin/console doctrine:migrations:migrate` může smazat všechna data z databáze, pokud jsou vytvořené migrace špatně navržené (nemělo by se stát, pokud do nich nebudeme zasahovat). Poté máme vše připraveno na integraci nové entity do databáze.

Následně bude třeba vytvořit třídu pro získávání a ukládání dat do databáze o dané entitě. K tomu slouží rozhraní `ItemFetcher` ve složce `src/Utils/Fetcher`. Je potřeba, aby daná třída toto rozhraní implementovala, což jsou metody:

`fetch(string $accessToken, EntityManagerInterface $entityManager): void` slouží k získání dat ze služby, jejich zpracování a následnému uložení,

`parse(mixed $data): mixed` slouží k vytvoření 1 objektu entity ze získaných dat,

`prepareDB(EntityManagerInterface $entityManager): void` slouží k přípravě databáze na získání/aktualizaci dat (smazání tabulky a její znovuvytvoření).

Příklad implementace metod takové třídy je vidět v 3.3.

Výpis kódu 3.3 Zjednodušený příklad implementace metod třídy Fetcher (CourseFetcher)

```
private const KOSAPI_COURSES = "https://kosapi.fit.cvut.cz/api/3/courses";
private const ENTITY = Course::class;

// Ziskani dat (z KOSapi) a jejich zpracovani
public static function fetch(string $accessToken,
                             EntityManagerInterface $entityManager)
{
    self::prepareDB($entityManager);

    $client = new Client();
    $response = $client->get(self::KOSAPI_COURSES, [
        'headers' => [
            'Authorization' => 'Bearer ' . $accessToken
        ]
    ]);

    $courses = $response->getBody()->getContents();
    foreach ($courses as $course) {
        $entityManager->persist(self::parse($course));
    }

    $entityManager->flush();
}

// Format je Atom (da se s nim pracovat jako s XML)
public static function parse(mixed $data): mixed
{
    $content = $data->xpath("atom:content");
    $name = $content[0]->name;;
    $code = $content[0]->code;

    return new Course($code, $name);
}

// Bude stejne (nebo velmi podobne) pro vsechny entity
static function prepareDB(EntityManagerInterface $entityManager): void
{
    $metadata = $entityManager->getClassMetadata(self::ENTITY);
    $schemaTool = new SchemaTool($entityManager);
    $schemaTool->dropSchema([$metadata]);
    $schemaTool->createSchema([$metadata]);
}
```

Po implementaci potřebného fetcheru je nutné třídu zaregistrovat ve třídě `FetchManager` ve složce `src/FetchManager`. Toho docílíme tak, že v poli `ENTITIES` přidáme řádek `{NováEntita}::class => {NováEntita}Fetcher::class`. Pole se zaregistrovanými entitami ve třídě `FetchManager` je vidět v 3.4.

Výpis kódu 3.4 Pole zaregistrovaných entit (FetchManager)

```
// Tyto entity ziskaji sva data pres patricny Fetcher
private const ENTITIES = [
    Course::class => CourseFetcher::class,
    Teacher::class => TeacherFetcher::class,
    Room::class => RoomFetcher::class
];
```

Poté je potřeba vytvořit třídu, která bude implementovat rozhraní `Item` ve složce `src/Service/Items`. Tato třída bude implementovat metodu `find(string $searchedValue): array`, která bude sloužit k vyhledávání všech relevantních výsledků dané entity na uživatelův dotaz a k jejich rankingu. Tato metoda musí vracet pole, které bude obsahovat `{NováEntita}View`. Tento nově vytvořený `Item` je nutné zaregistrovat v konfiguračním souboru `services.yaml` ve složce `config` stejným způsobem, jako jsou zaregistrované ostatní třídy `Item`. Příklad implementace takové metody (ze třídy CourseItem) je znázorněn v 3.5 a registrace v konfiguračním souboru `services.yaml` je vidět v 3.6.

Výpis kódu 3.5 Zjednodušený příklad implementace metody find (CourseItem)

```
public function find(string $searchedValue): array
{
    // Zpracovani zadaneho dotazu
    $searchedValue = strtoupper($searchedValue);
    $words = explode(' ', $searchedValue);

    $queryBuilder = $this->courseRepository->createQueryBuilder('c');

    // Vyhledavani v databazi na zaklade podobnosti v zadanych slovech
    $i = 1;
    foreach ($words as $word) {
        $queryBuilder = $queryBuilder
            ->orWhere('c.code LIKE :value' . $i)
            ->orWhere('UPPER(c.name) LIKE :value' . $i)
            ->setParameter('value' . $i++, '%' . $word . '%');
    }

    // Ziskani vysledku z databaze
    $select = $queryBuilder->getQuery();

    // Serazeni nalezenych predmetu podle urcitych pravidel
    $courses = self::rankCourses($select->getArrayResult(), $words);

    // Vytvoreni pole CourseView (Course preveden na CourseView)
    return array_map(function (array $course)
        {
            return new CourseView($course, $course['rank'],
                                  new DateTime($course['approvalDate']));
        }, $courses);
}
```

Výpis kódu 3.6 Registrace CourseItem v services.yaml

```
CourseItem:
    class: App\Service\Items\CourseItem

App\Service\SearchService:
    class: App\Service\SearchService
```

```
        autowire: true
        arguments:
            - ['@CourseItem']
```

`{NováEntita}View` slouží k zobrazování (renderování) entity v šablonách a musí implementovat rozhraní `ItemView` ve složce `src/View`, což jsou metody:

`template(): string` slouží k získání názvu šablony, která se použije k zobrazení entity,

`data(): array` slouží k získání samotných dat entity,

`rank(): int` slouží k získání ranku entity (větší = lepší),

`lastUpdate(): DateTime` slouží k získání data poslední úpravy dat (pro případ, že ranky některých entit jsou stejné).

Příklad implementace takové třídy je vidět v 3.7 (CourseView).

Výpis kódu 3.7 Implementace třídy CourseView

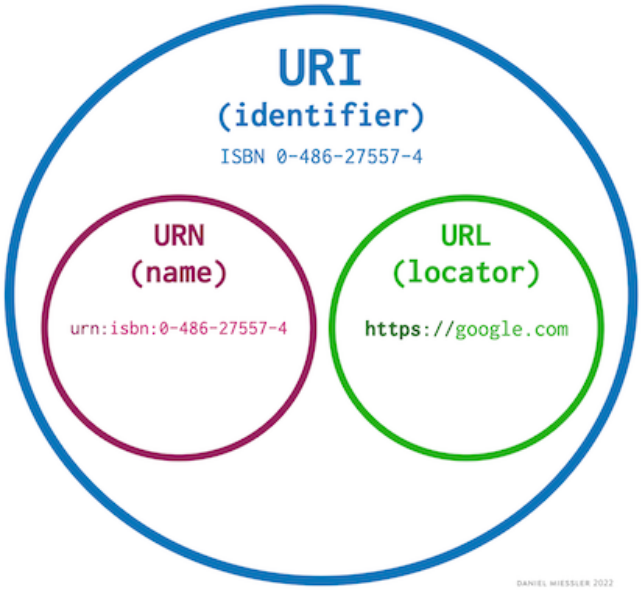
```
    private const TEMPLATE_NAME = "Items/course.html.twig";
    private array $itemData;
    private int $rank;
    private DateTime $lastUpdate;

    public function __construct(array $itemData, int $rank,
                                DateTime $lastUpdate)
    {
        $this->itemData = $itemData;
        $this->rank = $rank;
        $this->lastUpdate = $lastUpdate;
    }

    public function template(): string
    {
        return self::TEMPLATE_NAME;
    }

    public function data(): array
    {
        return $this->itemData;
    }

    public function rank(): int
    {
        return $this->rank;
    }

    public function lastUpdate(): DateTime
    {
        return $this->lastUpdate;
    }
```

Poté už stačí vytvořit samotnou šablonu entity ve složce `templates/Items`. Příklad takové šablony je vidět v 3.8 (šablona pro zobrazení předmětu).

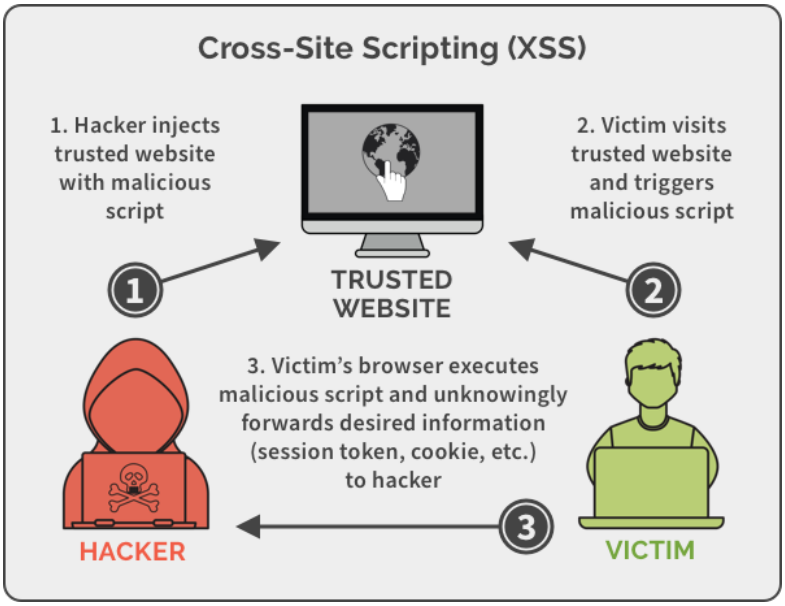
Výpis kódu 3.8 Příklad šablony pro zobrazování předmětu

```
<section class="card box">
    <header class="card-header">
        <h1 class="card-header-title is-1 mb-0">
            [{{ data.code }}] - {{ data.name }}
        </h1>
    </header>
    <div class="card-content">
        {% if data.homepage != null %}
            <p class="mb-0">
                <a href="{{ data.homepage }}">
                    Stranka predmetu
                </a>
            </p>
        {% else %}
            <p class="mb-0">Predmet nema svou stranku</p>
        {% endif %}
        <a href="{{ data.timeTableUrl }}">Rozvrh predmetu (TimeTable)</a>
        <section>
            <h3 class="subtitle is3 mb-0 mt-2">
                Nejblizsi 2 udalosti predmetu
            </h3>
            {{ include('Items/event.html.twig', {events: data.events}) }}
        </section>
    </div>
</section>
```

Pokud bude entita využívat novou službu, nebo chceme vytvořit novou službu, vytvoříme třídu ve složce `src/Service/Resource`. Pro získání *access tokenu* lze využít statickou metodu `ZuulOAASResource::getAccessToken(): string`.



Obrázek 3.1 Rozdíl mezi URI, URL a URN, převzato z `https://danielmiessler.com/study/difference-between-uri-url/`.



Obrázek 3.2 Diagram znázorňující kroky útoku XSS, převzato z `https://www.business2community.com/business-innovation/cross-site-scripting-xss-web-based-application-security-part-3-02204503`

Kapitola 4
Testování prototypu

Testování patří mezi důležitou součást každého vývoje aplikací. Způsobů testování je velké množství, záleží na tom, co přesně chceme v rámci aplikace testovat a jaké části chceme testovat.

Pro testování aplikace WhereIS se hodí uživatelské testování. Prototyp aplikace byl nasazen a poskytnutím aplikace určité skupině studentů lze nejlépe zjistit, jestli aplikace funguje dle očekávání, tedy jestli vrací žádané informace, jestli vyhledané volné místnosti jsou opravdu volné apod. Krom uživatelského testování by se pro zhodnocení prototypu dalo využít také heuristické vyhodnocení.

4.1 Uživatelské testování vs heuristické vyhodnocení

V rámci hodnocení funkčnosti celé aplikace lze krom uživatelského testování, kde výsledky závisí na spokojenosti uživatelů a jejich zpětné vazbě k testování, využít také heuristické vyhodnocení, což je hodnocení aplikace jako celku a je vhodné pro hodnocení prototypů. Takových heuristických vyhodnocení je velké množství, jedna z nejznámějších je *Nielsenova heuristika*.

V článku [27] uvádí, že při porovnávání efektivity uživatelského testování a heuristického vyhodnocení webových aplikacích bylo zjištěno, že obě metody nalezly jiné problémy v používání a fungování aplikace. Tyto zásadní rozdíly naznačují, že metody by se měly považovat za komplementární a neměly by mezi sebou soutěžit. Uživatelské testování je závislé na tom, co v aplikaci aktuálně existuje, jak jsou uživatelé spokojení s používáním těchto funkcionalit a jestli funkcionality fungují podle očekávání. Heuristické vyhodnocení se na druhou stranu opírá především o odborné znalosti a zkušenosti inženýrů, kteří by stránku vyhodnocovali na základě určitých faktorů.

Pro testování a zhodnocení prototypu aplikace WhereIS lze použít jak uživatelské testování, tak heuristické vyhodnocení pro nejlepší a nejefektivnější zhodnocení.

4.2 Heuristické vyhodnocení

Nielsen a Molich v [28] definují heuristické vyhodnocení jako metodu analýzy použitelnosti rozhraní aplikace. Tato analýza se tedy snaží přijít na to, co v rozhraní je dobré a co špatné. K tomuto vyhodnocení dojde na základě určitých faktorů, které se v průběhu této analýzy vyhodnocují. V roce 1990 Nielsen a Molich představili v [28] 9 základních faktorů, na základě kterých dochází k vyhodnocení jejich heuristiky (známá jako *Nielsenova heuristika*). V roce 1994 Nielson v [29] popisuje, že pro pokrytí 90 % případů a problémů je potřeba použít kolem 53 faktorů, což je pro praktické použití obrovské množství. Proto vymezil top 10 nejdůležitějších faktorů, na základě kterých by se mělo vyhodnocovat. Tyto faktory jsou:

1. Visibility of system status — uživatel by měl vědět a být informován o tom, co se děje a kde se nachází,
2. Match between system and real world — přirovnání věcí z opravdového světa k akcím/věcem v aplikaci/systému (např. ikonka koše = nástroj na mazání věcí),
3. User control and freedom — uživatelé musí mít možnost „dělat si, co se jim zlíbí“, zároveň to pro ně musí být bezpečné a musí mít možnost vždy své akce vrátit zpět (zrušit, *undo*, *redo*),
4. Consistency and standards — stejné věci vždy vyjadřovat/zobrazovat stejným způsobem, vše by mělo být konzistentní mezi sebou a nemělo by se to měnit,
5. Error prevention — zamezit chybám, podrobné a jasné chybové hlášky, použití výchozích hodnot, jazyka uživatele,
6. Recognition rather than recall — nevyžadovat od uživatele, aby si pamatoval spoustu věcí, vždy vše zobrazovat a připomínat, co se děje, co daná věc udělá, znovu se zeptat a potvrdit akci,
7. Flexibility and efficiency of use — systém by měl být flexibilní pro různé typy uživatelů (málo/hodně zkušení — pro málo zkušené jsou pokročilá nastavení schována), neměl by používat rozptylující prvky, používání zkratek,
8. Aesthetic and minimalist design — použití jednoduchého, minimalistického designu, zvýrazňovat důležité prvky, nepoužívat rozptylující prvky,
9. Help users recognize, diagnose, and recover from errors — pomoct uživateli rozpoznat, co se děje, k jaké chybě došlo a jak chybu opravit nebo se jí vyvarovat,
10. Help and documentation — používání nápověd, dokumentace, návodů k usnadnění práce s aplikací/systémem/funkcionalitou.

Na základě těchto 10 faktorů vyhodnotím aplikaci WhereIS. Hodnocení nebude samozřejmě tak objektivní, jako v případě třetí nezávislé strany, ale pokusím se objektivitu co nejvíce udržet. Heuristickým vyhodnocením začínám z toho důvodu, že v reálném použití je levnější, než uživatelské testování, protože nejsou potřeba další lidé k provedení tohoto vyhodnocení.

1. Visibility of system status — přímo navigace k dispozici není, uživateli ale pomáhají nadpisy stránek, nyní chybí status načítání výsledků při vyhledávání (bude doplněno v dalších verzích),
2. Match between system and real world — pár ikon použito (domovská stránka, varovný vykřičník při nezadání dotazu), mohou být doplněny další (ikona k odhlášení, ikona k vyhledávání),
3. User control and freedom — pocit kontroly a bezpečí splněn, uživatel zde nemůže něco změnit/smazat/rozbít,
4. Consistency and standards — konzistence v zobrazování věcí zajištěna, dodržen stejný formát,
5. Error prevention — jediná chybová hláška je při nezadání dotazu a snaze vyhledávat, jiné chybové hlášky potřeba nejsou (problém při autorizaci řeší autorizační server ČVUT),
6. Recognition rather than recall — mohlo by být zlepšeno tím, že na stránce s výsledky bude zopakován uživatelův dotaz, jinak splněno, protože se v aplikaci moc jiných podobných prvků nevyskytuje,
7. Flexibility and efficiency of use — neexistují pokročilá nastavení pro zkušenější uživatele, v aplikaci by si nenašla využití (šlo by implementovat vyhledávání na základě pokročilejších dotazů — speciální znaky, regulární výrazy…),

8. Aesthetic and minimalist design — splněno, vše důležité je dostatečně viditelné, žádné rušivé elementy, jednoduchý design,
9. Help users recognize, diagnose, and recover from errors — chybová hláška při nezadání dotazu poprosí uživatele o vyplnění vyhledávacího pole,
10. Help and documentation — dokumentace v aplikaci není, jednoduchý návod k použití je k dispozici v A.2, dle mého názoru nejsou v aplikaci složité prvky, které by potřebovaly podrobné vysvětlení.

Jak je z hodnocení patrné, nejsou splněny všechny důležité faktory Nielsenovy heuristiky. U některých faktorů to nehraje zásadní roli, protože aplikace je v zásadě jednoduchá na používání a nevyžaduje velké množství ovládacích prvků. Přesto však mohou být tyto faktory zlepšeny v dalších verzích aplikace, zejména proto, že se aplikace bude dál rozšiřovat o nová data, nové zdroje a funkcionality a samotné používání aplikace se může při těchto rozšíření ztížit.

4.3 Uživatelské testování

Uživatelské testování, jak už je z názvu patrné, je testování, kde výsledný produkt netestují vývojáři, ale samotní uživatelé produktu. Jak uvádí Lewis v [30], během tohoto testování pozoruje jeden či více pozorovatelů skupinu účastníků/uživatelů, kteří vykonávají určité úkoly. Tyto úkolu jsou předem pečlivě připravené. Uživatelské testování má za cíl otestovat konkrétní funkcionality systému, k tomu je potřeba zvolit správné účastníky pro testování, nejlépe různé typy lidí, často se také určuje, kolik iterací bude toto testování mít. Na konci testování dojde k zaznamenání výsledků. Prvním typem výsledků jsou popisy problémů používání aplikace, kdy se sepíše „příběh“, během kterého k problému dojde. Druhým typem výsledků je navrhnutí doporučení a úprav na základě předchozích popisů problémů, které by dané problémy mohly vyřešit.

V rámci uživatelského testování aplikace WhereIS byla zvolena skupina pěti studentů FIT ČVUT, kteří se testování zúčastnili. Tito studenti v rámci několika dní používali aplikaci každý den a snažili se nalézt chyby či aplikaci rozbít. Zároveň také poskytli zpětnou vazbu o tom, co se jim na aplikaci (ne)líbilo, co by do aplikace přidali či co by na aplikaci změnili. Nejčastěji se opakovaly tyto poznatky:

- vyhledávání by mohlo být dostupné i na stránce s výsledky, aby uživatel nemusel překlikávat zpět na vyhledávací stránku,
- vyhledávání pro málo specifické dotazy občas trvalo relativně dlouho (nejspíš kvůli velkému množství nových výsledků a jejich *cachování*, velké množství zobrazovaných výsledků),
- v rámci vyhledávání by mohlo být možné specifikovat chtěný výsledek (např. „PA2 předmět“, kde slovem „předmět“ uživatel specifikuje, že chce pouze předměty s daným kódem a ne nic dalšího,
- při prokliku na rozvrhy v aplikaci TimeTable se musí uživatel znovu přihlásit a poté jít zpět do aplikace WhereIS a na odkaz kliknout znovu — aplikace WhereIS za tento problém nemůže,
- aplikace je přehledná, jednoduchá k použití a na dotazy vracela pro studenty žádané výsledky.

Chyby v aplikaci nebyly během testování nalezeny žádné. V jeden den (sobota 30.04.2022) se však testerům nedařilo autorizovat přes autorizační server školy a aplikace (v debug režimu) vyhazovala výjimky. Tento problém ale už v žádný jiný den testování nenastal, jednalo se tedy pravděpodobně o chybu či výpadek na straně autorizačního serveru ČVUT.

Kapitola 5
Závěr

Na závěr této práce bych rád připomněl cíle práce a zhodnotil, jak cíle byly splněny a jestli došlo k nesplnění určitých cílů a případně z jakých důvodů. Po tomto zhodnocení se zastavím nad možnostmi budoucího vývoje a rozšíření aplikace WhereIS.

5.1 Zhodnocení

Hlavním cílem práce bylo navrhnout a vytvořit funkční prototyp vyhledávacího systému WhereIS, který bude sloužit studentům i učitelům ke snadnějšímu vyhledávání potřebných informací o předmětech, místnostech a učitelích. Z hlediska návrhu byla hlavním cílem jednoduchá rozšiřitelnost aplikace a integrace nových entit a zdrojů dat. Tyto hlavní cíle jsem úspěšně splnil. Prototyp aplikace je k dispozici k testování studenty FIT ČVUT, což je také potřebné k splnění cíle o testování aplikace, a k přidání nových entit a zdrojů dat stačí pouze vytvořit pár nových tříd a zaregistrovat je v konfiguračním souboru, není třeba upravovat již existující kód. V rámci implementace byl také zvolen vhodný způsob ukládání a práce s daty získaných ze zdrojů a služeb školy, aby nedocházelo k dlouhému vyhledávání výsledků a zahlcování služeb školy.

V rámci teoretické části jsem analyzoval již existující řešení / vyhledávací systémy a jejich výhody a nevýhody. Díky tomu jsem byl schopný zvolit přístup, kterým jsem se řídil během návrhu a implementace prototypu. Následně jsem se zabýval i potřebnými službami a zdroji dat, nad kterými aplikace výsledky vyhledává a agreguje. K těmto službám jsem získal přístup a pro budoucí vývojáře jsem popsal návod, jak by měli postupovat, pokud chtějí služby školy pro své aplikace používat.

V rámci cílů nebyla splněna jedna část požadavku k vyhledávání učitelů, místností a předmětů. Součástí tohoto požadavku byla mimo jiné možnost najít, kde se učitel vůči rozvrhu aktuálně nachází (tedy výpis 2 nejbližších událostí učitele, ze kterých lze informaci vyčíst). Tato část požadavku nebyla splněna z důvodu nedostatečných práv, která jsou potřebná k získání kalendáře učitele pomocí služby Sirius. O práva bylo během implementace prototypu požádáno, nicméně v době psaní práce nebyla práva udělena. Prototyp je na tuto funkcionalitu připraven, jakmile budou data poskytnuta k dalšímu zpracování, je možné doimplementovat pouze službu, která data získává.

5.2 Budoucí vývoj

V rámci budoucího vývoje je spousta možností, kam aplikaci směřovat a jaké nové funkcionality implementovat. Aplikaci je možné rozšířit o nové entity a zdroje dat, nad kterými bude agregovat. Student tak bude mít možnost vyhledávat například menzy, ve kterých vaří jeho

oblíbené jídlo, nebo najít informace o koleji, na které bydlí. Pro již existující entity lze přidat více informací a lépe propojit jednotlivé entity mezi sebou v rámci samotného vyhledávacího systému. K předmětům lze například doplnit často používané nástroje a programy (ProgTest, MARAST…), u učitelů lze doplnit jejich fotografie nebo seznam předmětů, které učí.

V rámci vyhledávání volných místností by bylo možné přidat možnost, díky které by student zadal datum a čas, od kdy chce volnou místnost nalézt, a systém by mu vrátil seznam všech volných místností v daný čas. Podobným způsobem by student mohl ovlivňovat také počet nejbližších událostí dané entity, který je v době psání práce nastaven na 2.

Nakonec by aplikace mohla pracovat s přihlášenými uživateli zvlášť. Mohla by například zobrazovat uživatelovy nejbližší události v dané místnosti či předmětu, nebo by si uživatel mohl definovat seznam oblíbených jídel a v případě, že jedno z těchto jídel bude v daný den v menze, tak mu to aplikace oznámí.

Možností na rozšíření je zkrátka mnoho a záleží pouze na tom, co by v danou chvíli bylo pro studenty a učitele nejvíce užitečné.

Příloha A

Návod na spuštění a používání aplikace

A.1 Návod na spuštění

1. Nainstalujte si nejnovější verzi Dockeru[1][2].
2. Spusťte Docker.
3. Otevřete příkazovou řádku ve složce projektu.
4. V příkazové řádce spusťte příkaz `docker-compose build --no-cache --pull`. Pokud tento příkaz neuspěje, zkuste v souboru „Dockerfile“ změnit verzi PHP (např. na 8.0.2 — nejnižší možná verze pro fungování Symfony).
5. V příkazové řádce spusťte příkaz `docker-compose up [-d]`. Volitelný přepínač `-d` slouží pro běh na pozadí.
6. Po chvilce bude aplikace dostupná na adrese localhost[3].
7. Pro vypnutí aplikace stačí použít zkratku CTRL + C (pokud neběží na pozadí), nebo příkaz `docker-compose down` (pokud běží na pozadí).

A.2 Návod k použití

Používání samotné aplikace je velmi jednoduché. Po přesměrování na úvodní stránku je po kliknutí na tlačítko „Přihlásit“ uživatel přesměrován na autorizační server ČVUT, kde se přihlásí svým platným ČVUT účtem. Po úspěšném přihlášení je uživatel přesměrován zpět do aplikace WhereIS, konkrétně na hlavní vyhledávací stránku, kde uživatel může zadávat své dotazy (vždy musí zadat alespoň jeden znak). Po zadání dotazu stačí kliknout na tlačítko „Vyhledat“ nebo stisknout klávesu „Enter“. Poté budou výsledky uživateli zobrazeny. Pro opětovné vyhledávání musí v aktuální verzi uživatel kliknout na tlačítko „Hlavní stránka“ v levém horním rohu stránky. Pro odhlášení stačí kliknout na tlačítko „Odhlásit se“ v pravém horním rohu.

Dotazy uživatele mohou být:

[1] https://www.docker.com/products/docker-desktop/
[2] Testováno na Docker Desktop 4.7.1 (77678).
[3] https://localhost/

- jméno, příjmení nebo username učitele (či jakákoliv část těchto řetězců),
- jméno či kód předmětu (či jakákoliv část těchto řetězců),
- kód místnosti (či jakákoliv část tohoto řetězce),
- řetězce ve stylu ”voln[á,é,ý] místnost(i), učebn[a,y]”pro vyhledávní volných laboratorních a seminárních místností.

Bibliografie

1. *Basic Information about all Search Engine which exist in the world.* [Online]. Mumbai: Tech ADR Group, 2021 [cit. 2022-04-17]. Dostupné z: https://www.onionride.com/p/onionride-how-search-engines-operate.html.
2. Introduction. In: *Web Crawling* [online]. 3. vyd. Hanover: Now Publishers Inc., 2010, s. 1–4 [cit. 2022-04-18]. ISBN 978-1-60198-322-0. Dostupné z: https://books.google.cz/books?id=CRS_GMjF5gwC%5C&lpg=PA1%5C&ots=_m1djb27PN%5C&dq=crawling%5C&lr%5C&pg=PA1%5C#v=onepage%5C&q=crawling%5C&f=false.
3. CHRIS, Alex. *Top 10 Search Engines In The World (2022 Update)* [online]. Nicosia: MACC MEDIA LTD, 2002 [cit. 2022-05-09]. Dostupné z: https://www.reliablesoft.net/top-10-search-engines-in-the-world/.
4. INDIG, Kevin. *Google's index is smaller than we think - and might not grow at all* [online] [cit. 2022-04-18]. Dostupné z: https://www.kevin-indig.com/googles-index-is-smaller-than-we-think-and-might-not-grow-at-all/.
5. MENG, Weiyi. Metasearch Engines. In: *Encyclopedia of Database Systems*. Ed. LIU, LING; ÖZSU, M. TAMER. Boston, MA: Springer US, 2009, s. 1730–1734. ISBN 978-0-387-39940-9. Dostupné z DOI: 10.1007/978-0-387-39940-9_217.
6. GLOVER, Eric; LAWRENCE, Steve; BIRMINGHAM, William; GILES, C. Architecture of a Metasearch Engine that Supports User Information Needs. 2000. Dostupné z DOI: 10.1145/319950.319980.
7. *Webová služba (web service)* [online]. Wilmington: MANAGEMENTMANIA.COM LLC, 2012 [cit. 2022-04-11]. Dostupné z: https://managementmania.com/cs/webova-sluzba.
8. *What is Atom 1.0* [online]. Telangana: Tutorials Point India Private Limited, 2013 [cit. 2022-04-11]. Dostupné z: https://www.tutorialspoint.com/rss/what-is-atom.htm.
9. RICHARDS, Robert. Representational State Transfer (REST). In: *Pro PHP XML and Web Services*. Berkeley, CA: Apress, 2006, s. 633–672. ISBN 978-1-4302-0139-7. Dostupné z DOI: 10.1007/978-1-4302-0139-7_17.
10. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, 2000. Disertace. University of California.
11. PICHLÍK, Roman. A REST. In: *Dagblog* [online]. Medium, 2002 [cit. 2022-04-11]. Dostupné z: https://dagblog.cz/a-rest-c5156313d79e.
12. HARDT, Dick. The OAuth 2.0 Authorization Framework. In: [RFC 6749]. RFC Editor, 2012, s. 1–10. Request for Comments, č. 6749. Dostupné z DOI: 10.17487/RFC6749.
13. SINGH, Vijay. *What is a Framework? [Definition] Types of Frameworks* [online] [cit. 2022-04-20]. Dostupné z: https://hackr.io/blog/what-is-frameworks.

14. *What is PHP?* [Online]. PHP Group [cit. 2022-04-20]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php.
15. *What is Python? Executive Summary* [online]. Python Software Foundation [cit. 2022-04-20]. Dostupné z: https://www.python.org/doc/essays/blurb/.
16. JOY, Ashwin. *5 Main Disadvantages of Python Programming Language* [online]. 2019 [cit. 2022-04-20]. Dostupné z: https://pythonistaplanet.com/disadvantages-of-python/.
17. *JavaScript* [online]. Mozilla Foundation [cit. 2022-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
18. *Pros and Cons of JavaScript – Weigh them and Choose wisely!* [Online]. DataFlair [cit. 2022-04-20]. Dostupné z: https://data-flair.training/blogs/advantages-disadvantages-javascript/.
19. HARTMAN, James. *What is Java? Definition, Meaning & Features of Java Platforms* [online]. Guru99 [cit. 2022-04-20]. Dostupné z: https://www.guru99.com/java-platform.html.
20. *Advantages and Disadvantages of Java* [online]. TechVidvan [cit. 2022-04-20]. Dostupné z: https://techvidvan.com/tutorials/pros-and-cons-of-java/.
21. CHOJRIN, Mauro. *Laravel vs. Symfony: A Side by Side Comparison* [online]. Adeva, 2015 [cit. 2022-04-21]. Dostupné z: https://adevait.com/laravel/laravel-vs-symfony-comparison.
22. *Nette* [online]. Nette Foundation, 2008 [cit. 2022-04-23]. Dostupné z: https://nette.org/cs.
23. MEIER, Andreas; KAUFMANN, Michael. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management* [online]. 1. vyd. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, part of Springer Nature, 2019 [cit. 2022-04-28]. ISBN 978-3-658-24549-8. Dostupné z DOI: 10.1007/978-3-658-24549-8.
24. KEMPE, Shannon; WILLIAMS, Paul. *UnQL: A Standardized Query Language for NoSQL Databases* [online]. Studio City: DATAVERSITY, 2011 [cit. 2022-04-28]. Dostupné z: https://www.dataversity.net/unql-a-standardized-query-language-for-nosql-databases/.
25. DAOUDI, Aymen; ELBOUSSAIDI, Ghizlane; MOHA, Naouel; KPODJEDO, Sègla. An exploratory study of MVC-based architectural patterns in Android apps. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* [online]. 2019-04-08, s. 1711–1720 [cit. 2022-04-28]. ISBN 9781450359337. Dostupné z DOI: 10.1145/3297280.3297447.
26. KIRSTEN, S. *Cross Site Scripting (XSS)* [online]. Wakefield: OWASP Foundation, 2001 [cit. 2022-04-27]. Dostupné z: https://owasp.org/www-community/attacks/xss/.
27. TAN, Wei-siong; LIU, Dahai; BISHU, Ram. Web evaluation: Heuristic evaluation vs. user testing. *International Journal of Industrial Ergonomics*. 2009, roč. 39, č. 4, s. 621–627. ISSN 0169-8141. Dostupné z DOI: https://doi.org/10.1016/j.ergon.2008.02.012. Special issue: Felicitating Colin G. Drury.
28. NIELSEN, Jakob; MOLICH, Rolf. Heuristic evaluation of user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90* [online]. 1990, s. 249–256 [cit. 2022-05-02]. ISBN 0201509326. Dostupné z DOI: 10.1145/97243.97281.
29. NIELSEN, Jakob. Enhancing the explanatory power of usability heuristics. *Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94* [online]. 1994, s. 152–158 [cit. 2022-05-02]. ISBN 0897916506. Dostupné z DOI: 10.1145/191666.191729.

30. LEWIS, James R. Usability Testing. In: *Handbook of Human Factors and Ergonomics*. John Wiley & Sons, Ltd, 2012, kap. 46, s. 1267–1312. ISBN 9781118131350. Dostupné z DOI: https://doi.org/10.1002/9781118131350.ch46.

Obsah přiloženého média

- `README.txt` ..... stručný popis obsahu média
- `app` ..... adresář se soubory aplikace
  - `assets`
    - `styles` ..... adresář s CSS soubory
  - `config` ..... adresář s konfiguračními soubory frameworku a balíčků
  - `migrations` ..... adresář s databázovými migracemi
  - `public`
    - `room_plans` ..... adresář s plánky místností
  - `src` ..... adresář se zdrojovými kódy
  - `templates` ..... adresář se šablonami stránek
  - `README.pdf` ..... návod na spuštění a používání aplikace
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF