FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	WhereIS - vyhledávací platforma
Student:	Bc. Tomáš Heger
Vedoucí:	Ing. David Bernhauer, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

V rámci bakalářské práce T. Hegera vznikl prototyp aplikace WhereIS, která má sloužit pro usnadnění navigace nejen v budovách školy, ale i v jiných aspektech studia. Následující diplomová práce navazuje právě na tento prototyp a jejím hlavním cílem je dokončit aplikaci do stavu, který umožní reálné používání. Práce je součástí skupinového projektu WhereIS.

1. Získejte a analyzujte požadavky uživatelů.
2. Proveďte rešerši podobných systémů se zaměřením na vyhledávání.
3. Zanalyzujte existující prototyp a identifikujte problémy a příležitosti pro zlepšení vyhledávání.
4. Proveďte návrh vylepšení vyhledávací platformy.
5. Integrujte a otestujte vybraná zlepšení do existujícího prototypu.
6. Implementujte REST API a backendovou část dokončete do funkční aplikace.
7. Aplikaci zprovozněte a nasaďte.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 31. ledna 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

WhereIS – vyhledávací platforma

Bc. Tomáš Heger

Katedra softwarového inženýrství
Vedoucí práce: Ing. David Bernhauer, Ph. D.

9. května 2024

Poděkování

Chtěl bych poděkovat především svému vedoucímu Ing. Davidu Bernhauerovi, Ph. D., za vedení mé diplomové práce správným směrem a za pomoc, kterou poskytoval v případě problémů. Dále bych rád poděkoval svému příteli Bc. Illiu Brylovovi za výbornou spolupráci a podporu během vývoje a Bc. Jakubu Jabůrkovi za konzultaci řešení autorizace a autentizace uživatelů. Nakonec bych rád poděkoval své rodině a svým přátelům, kteří mi byli celou dobu velkou podporou.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. května 2024 .....

Odkaz na tuto práci

Heger, Tomáš. *WhereIS – vyhledávací platforma.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Tato práce se zabývá zlepšením vyhledávacích schopností existujícího prototypu aplikace WhereIS a dokončením backendové části aplikace za cílem jejího nasazení do produkce. V rámci rešerše je kladen důraz na existující vyhledávací nástroje, které je možné použít pro zlepšení vyhledávání, a na vytyčení problémů v existujícím prototypu. V rámci analýzy se autor zaměřuje na sběr a analýzu požadavků získaných od uživatelů. Návrh a implementace jsou cíleny na integraci zvoleného vyhledávacího nástroje, uživatelských požadavků a na zlepšení nedostatků a oprav problémů nalezených v původním prototypu. Výsledná aplikace je následně otestována pomocí unit testů a jednoduše nasazena na server pomocí Docker kontejnerů.

Klíčová slova vyhledávací platforma, backend, Sphinx, teorie vyhledávání, automatizace aktualizace dat, informační systém, školní systém, WhereIS

Abstract

This thesis deals with improving the search capabilities of an existing prototype of WhereIS application and completing the backend part of the application in order to deploy it into production. The focus of the research is on existing search tools that can be used to improve the search and to highlight problems in the existing prototype. As part of the analysis, the author focuses on the collection and analysis of the requirements received from users. The design and implementation are aimed at integrating the chosen search tool, the user requirements and at improving the deficiencies and fixing the problems found in the original prototype. The resulting application is then unit-tested and simply deployed to the server using Docker containers.

Keywords search platform, backend, Sphinx, information retrieval, automation of data update, information system, school system, WhereIS

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Tato práce navazuje na bakalářskou práci [1] autora, díky které vznikl první prototyp aplikace WhereIS. Na aplikaci pracoval autor poté společně s dalšími dvěma kolegy v rámci předmětu „Návrh uživatelského rozhraní“ na FIT ČVUT. Při této týmové práci došlo ke zlepšení a rozšíření funkcionalit aplikace v backendové části a také k přepracování celé frontendové části. Vymezení práce v rámci týmového projektu bylo následující:

- Bc. Tomáš Heger – práce na backendu:
  - rozšíření aplikace o menzy a jejich jídelníčky,
  - implementace oblíbených menz,
  - přidání nejbližších událostí přihlášeného uživatele,
  - implementace REST API využívané frontendovou částí,
  - přepracování původního kódu prototypu.
- Bc. Illia Brylov – práce na frontedu:
  - nahrazení všech původních HTML Twig šablon a CSS Reactem,
  - vytvoření wireframe prototypu,
  - napojení na REST API,
  - překlad do anglického jazyka.
- Bc. Jakub Jabůrek – práce na backendu a frontendu:
  - přepracování způsobu přihlašování uživatelů a vytváření session pomocí Symfony Security balíčku,
  - implementace směrování (routingu) a tlačítka zpět mezi stránkami,
  - orchestrace spuštění frontendu a backendu jako jedné aplikace.

Na systému WhereIS se v době psaní této práce pracuje odděleně. Autor této práce se zabývá zlepšením a dokončením backendové části, zatímco práce kolegy Bc. Illii Brylova se zabývá zlepšením a dokončením frontendové části aplikace.

Cíle práce

Hlavním cílem této diplomové práce je tedy zlepšit a dokončit backendovou část aplikace WhereIS, aby se dala nasadit a byla dostupná k používání. V rámci zlepšení existující aplikace bude kladen důraz především na schopnost vyhledávání mezi daty uloženými v databázi.

Teoretická část práce se nejprve zaměřuje na teorii vyhledávání, o kterou se autor bude v rešerši opírat, a následně jaká hotová řešení, která se zabývají vyhledáváním dat a informací, již existují. V rámci těchto existujících řešení se autor zaměří především na dostupné vyhledávací nástroje (knihovny, enginy, servery...), které by mohl využít a integrovat do systému WhereIS. Dalším důležitým krokem pro zlepšení backendové části je analýza kódu existujícího prototypu aplikace. Cílem této analýzy je vytyčit problémy a nedostatky, které původní implementace má, aby mohl autor v návrhu a v praktické části práce tyto problémy odstranit a nedostatky vylepšit. Následně autor společně s kolegou Bc. Illiou Brylovem vypracuje dotazník, jehož cílem je sběr požadavků uživatelů na aplikaci. Dotazník bude sbírat požadavky jak pro frontendovou, tak i pro backendovou část. Získané požadavky následně autor zanalyzuje a vybere ty, které je možné do prototypu integrovat a které studentům dokáží ulehčit vyhledávání informací, a zanalyzuje nové zdroje dat, které je potřeba přidat pro splnění požadavků. Posledním cílem teoretické části je důkladnější popis a vysvětlení nového a zaktualizovaného doménového modelu aplikace.

Cílem praktické části je zejména integrace zvoleného vyhledávacího nástroje, který zlepší samotné vyhledávání v rámci aplikace. Dalším cílem je návrh a integrace oprav problémů a zlepšení nedostatků původního prototypu, které byly nalezeny v teoretické části, a také integrace vybraných uživatelských požadavků společně s potřebnými novými zdroji dat. Následně je nutné upravit REST API, aby všechny nově integrované entity a data byly dostupné frontendové části aplikace. Posledním cílem praktické části je výsledné změny otestovat a následně celou aplikaci nasadit na server.

Struktura práce

Kapitola 1 se zabývá nejdůležitějšími částmi teorie vyhledávání (Information Retrieval). V rámci této teorie jsou popsány hlavní kroky při vyhledávání ve full-textových datech/informacích, jako jsou získání dat, zpracování textu, vyhledávání v textu a ranking nalezených výsledků.

Kapitola 2 se zabývá rešerší. V první části se autor zabývá rešerší existujících řešení, přesněji existujících vyhledávacích nástrojů a jak přesně tyto nástroje fungují. Po této rešerši následuje její shrnutí, kde autor ve zkratce popíše společné rysy těchto vyhledávacích nástrojů. Druhá část této kapitoly se zabývá rešerší existujícího prototypu WhereIS, ve které autor vytyčí problémy a nedostatky v kódu tohoto prototypu.

Kapitola 3 se zabývá analýzou. První část této kapitoly je zaměřena na tvorbu dotazníku, sběr uživatelských požadavků a následnou analýzu těchto požadavků, ve které autor popíše, co je pro daný požadavek potřeba, a rozhodne, jestli bude požadavek integrován do systému. Na základě těchto požadavků následně autor zanalyzuje nové zdroje dat, které je potřeba integrovat do

systému za cílem splnění získaných požadavků. Poté autor popíše nové procesy, které v aplikaci vzniknou, a nakonec popíše nový doménový model aplikace.

Kapitola 4 se zabývá návrhem a implementací. První část této kapitoly popisuje nové technologie, kterých autor využívá pro integraci uživatelských požadavků a pro opravy problémů a nedostatků původního prototypu, společně se zvoleným vyhledávacím nástrojem. Poté následuje popis integrace vybraných uživatelských požadavků a integrace nových zdrojů dat, jehož součástí je také kompletní návod přidávání právě nového zdroje a nové entity do systému. Následně jsou navrhnuty opravy a zlepšení pro problémy a nedostatky nalezené v původním prototypu. Ke konci kapitoly autor popíše integraci zvoleného vyhledávacího nástroje do systému a zmíní pár informací k dokumentaci vytvořeného REST API.

Kapitola 5 se zabývá nasazením a testováním aplikace. Autor v této kapitole popisuje použitý statický analyzátor kódu a použitou metodiku testování pomocí unit testů. Následně se autor zabývá vytvořením CI pomocí GitLab CI, jehož součástí jsou statická analýza kódu a spouštění unit testů. Na konci kapitoly se autor zabývá nasazením aplikace na server pomocí Docker kontejnerů.

Kapitola 1

Teorie vyhledávání informací

V této kapitole popíše autor nejdůležitější části teorie vyhledávání informací (Information Retrieval). Jedná se o zásadní teorii nejen z pohledu textového vyhledávání. Kromě důležitých pojmů popíše autor také jednotlivé kroky, ke kterým během vyhledávání dochází, a na tyto teoretické postupy naváže v kapitole 2.1, kde autor provede rešerši, jak některé vyhledávací knihovny tyto kroky konkrétně aplikují.

Vyhledávání/získávání informací (častěji označováno anglickým souslovím „Information Retrieval (IR)“) je disciplína, kterou Ceri a spol. [2] definují jako hledání relevantních dokumentů na základě shody s lexikálními vzory v zadaném dotazu. Je důležité podotknout, že slovo „relevantní“ je v tomto ohledu:

subjektivní – dva různí uživatelé hledající stejné informace mohou mít jiný názor na výsledky vrácené systémem (dokumenty),

dynamické – dokumenty vrácené uživateli v daném čase mohou ovlivnit posouzení relevance dokumentů, které budou vráceny později,

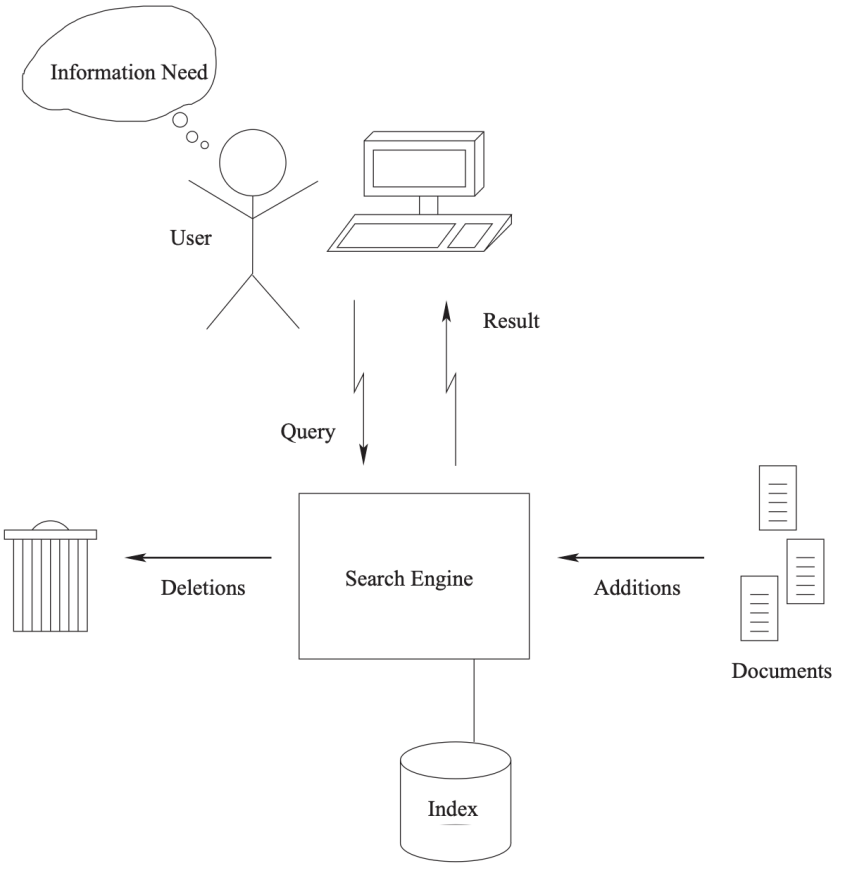
mnohostranné – relevance není určena pouze obsahem dokumentu, ale také dalšími aspekty (autorita, důvěryhodnost, specifičnost…).

Klíčovým aspektem a vlastností vyhledávání informací je podle [2] práce s nestrukturovaným typem dat – nejčastěji s textem. Nestrukturovaných dat je na internetu extrémní množství a neustále přibývají další, proto je velmi důležité v těchto datech umět vyhledávat. To je zásadní rozdíl od podobné disciplíny, a to získávání dat (nebo také „Data Retrieval“), kdy získáváme všechny objekty/data, která odpovídají přesně definovaným podmínkám a kritériím (např. pomocí SQL v relačních databázích, nebo pomocí XPath v XML dokumentech).

1.1 Základní architektura informačního systému

Na začátku, jak uvádí Buttcher a spol. [3], je uživatel, který má *potřebu vyhledat určitou informaci*. Na základě této potřeby zformuluje *dotaz* (anglicky „query“) pro informační systém, který je v drtivé většině případů textový. Tento dotaz se skládá z malého množství klíčových slov – termů. Dotaz je zpracován vyhledávacím strojem, jehož cílem je pomocí *invertovaného indexu*

(vizte 1.4) najít relevantní dokumenty pro zadaný dotaz. Dokumentem se v této oblasti myslí jakákoliv samostatná jednotka s určitým obsahem (nejčastěji textovým), která může být vrácena jako výsledek dotazu. Po získání kolekce relevantních dokumentů dochází k jejich *ohodnocení* (anglicky „ranking"). Díky tomuto ohodnocení by uživatel měl najít hledané informace v několika prvních dokumentech/výsledcích. Některé primitivní vyhledávací systémy ohodnocení neprovádí, pouze vrátí všechny relevantní výsledky. Náčrt takovéto architektury (bez ranking fáze) je možné vidět na obrázku 1.1.



Obrázek 1.1: Základní architektura informačního systému (bez ranking fáze), převzato z [3]

1.2 Získávání dat pro vyhledávání

Na obrázku architektury 1.1 jsou také vidět dvě další operace, a to „additions" – přidávání dokumentů do udržované kolekce – a „deletions" – odstranění dokumentů z udržované kolekce. Jedná se o klíčové operace, protože data se neustále mění, přibývají nová a naopak stará zanikají, či již nejsou relevantní.

Z toho důvodu je potřeba udržovat tato data co nejvíce aktuální, aby se z nich daly získat a vyhledávat co nejnovější informace.

Způsobů, jak tato data získat, je hned několik. Nejstarší a nejméně efektivní možností je manuální práce, kdy data/dokumenty jsou do kolekce přidávány člověkem. Tento způsob je vzhledem k rychlosti tvorby nových dat v současné době nepoužitelný, nicméně dá se stále využít pro získání takových dat, která nejsou na internetu dostupná.

Další možností je využití API webových služeb. V době psaní této práce převládá zejména *REST API*, které využívá HTTP metod pro různé operace nad zdroji/daty. Díky `GET` operaci lze tak získat různá data o různých zdrojích, které dané API nabízí. Tato možnost také nabízí velmi jednoduchý způsob aktualizace dat – znovudotázání na konkrétní zdroj.

V neposlední řadě je velmi hojně využívanou možností *scrapping* spojený s *crawlingem*. Jedná se o automatizované procesy, kde *crawleři* procházejí webové stránky, ty indexují a pomocí URL odkazů objevují nové, a *scrapeři*, kteří naopak extrahují data z těchto objevených webových stránek. Stránky, ze kterých se data těmito způsoby získávají, jsou navštěvovány po určité době znovu, aby došlo k aktualizaci dat v indexu (přidání nových, odebrání starých).

1.3 Zpracování textu

Aby získávání informací bylo rychlé a vracelo relevantní výsledky na zadané dotazy, je potřeba, aby tyto informační systémy dobře zpracovaly texty obsažené v udržované kolekci dokumentů. Textových formátů v době psaní této práce existuje velké množství (Microsoft Word, XML, HTML, RTF, PDF, LaTeX...) a je potřeba vědět, jak jednotlivé formáty fungují pro správnou *extrakci* textu z nich.

Po extrakci textu následuje další důležitý krok a tím je *tokenizace*. Dle [3] je tokenizace kritickým krokem v indexačním procesu textu, protože účinně omezuje třídu dotazů, které může systém zpracovávat. Výsledkem tokenizace jsou jednotlivé tokeny (slova) s přiřazeným celým číslem reprezentující pozici slova v daném textu. V tomto procesu je také důležité dávat pozor na *kódování* jednotlivých textů (ASCII, UTF-8, UTF-16...), které se napříč kolekcí může u různých dokumentů lišit. Například anglické texty mohou být kódovány čistě pomocí ASCII, zatímco české texty musí být kódovány jiným kódováním (UTF-8, UTF-16, Windows 1250...) kvůli většímu počtu znaků v jazyce.

Se získanými tokeny lze pak následně provádět další operace, jako je *stematizace*, *lemmatizace*, *odstranění interpunkce*, *převedení na malá/velká písmena* či *odstranění stop slov*. Cílem těchto operací je zmenšení množství získaných tokenů a tím pádem zmenšení náročnosti vyhledávání relevantních výsledků. Jelikož tyto operace mohou mít zásadní vliv na schopnosti vyhledávání informačního systému, rozebere je autor v samostatných podsekcích.

Stematizace (stemming) Stematizace, jak uvádí Manning a spol [4], je proces, při kterém dojde k odstranění části slova (předpony či přípony) za cílem získat kmen slova. Je důležité zmínit, že z hlediska českého jazyka se opravdu jedná o kmen, nikoliv o kořen.

Lemmatizace (lemmatization) Lemmatizace je podle [4] proces, při kterém dojde k převedení slova na základní tvar. Podstatná jména tedy převede do prvního pádu, slovesa převede na infinitiv apod.

Odstranění interpunkce (punctuation) Jak je z názvu patrné, odstranění interpunkce je proces, při kterém dojde k odstranění teček, čárek, apostrofů, uvozovek atd. Tento proces však může uškodit schopnostem vyhledávacího systému, např. zkratky nebudou obsahovat tečky (s. r. o. – s r o, č. j. – č j – čj...) a může tak dojít k záměně významu (jako u zkratky č.j. kde místo čísla jednotného můžeme myslet český jazyk). Tato operace je ještě více problematická u angličtiny, zejména kvůli častému používání apostrofu („I'll" – I ll, „Don't" – Don t). Navíc apostrof či interpunkce je také někdy součástí názvů či samotných slov v jazyce, jak uvádí Buttcher a spol. [3]. Mezi taková slova či názvy patří „o'clock" či název kapely „Panic! At the Disco".

Převedení na malá/velká písmena (capitalization) Tato operace je také zřejmá z názvu. Cílem je sjednocení slov, která normálně nezačínají velkým písmenem, ale vyskytují se na začátku věty. Nicméně tento proces může mít také špatný vliv na schopnosti vyhledávání, a to zejména u zkratek. Jak v českém, tak i v anglickém jazyce se nacházejí akronymy, které samy o sobě mohou v daném jazyce reprezentovat slovo (například SPOLU – spolu, ANO – ano, POINT – point, US – us) a převedením na malá písmena ztratíme informaci o tom, že se jednalo právě o takový akronym. Nicméně podle [3] se tento problém dá vyřešit pomocí dvojité indexace, ve které se budou indexovat obě varianty těchto slov.

Odstranění stop slov (stopping) Stop slova jsou slova, která sama o sobě nemají žádný význam či nenesou žádnou informaci. Mezi taková slova můžeme řadit členy, zájmena, předložky, spojky...Jelikož taková slova jsou relativně častá v českých i anglických větách, dojde pomocí této operace ke značné redukci získaných tokenů. V některých případech však tato eliminace může vést k nepřesným výsledkům, protože někdy jsou stop slova součástí názvů či jmen.

1.4 Invertovaný index

Po extrakci tokenů a jejich normalizaci pomocí operací zmíněných výše vzniknou *termy*. Množina všech různých termů z celé kolekce dokumentů je *slovník*. Právě termy jsou ta slova, která se indexují a používají pro vyhledávání. Nejčastějším a velmi efektivním způsobem, jak tyto termy indexovat, je právě *invertovaný index*.

Pibiri a Venturini [5] definují invertovaný index jako množinu *invertovaných listů* termů t slovníku. Invertovaný list pak definují jako seřazenou posloupnost identifikátorů dokumentů (celých čísel), ve kterých se daný term nachází. Kromě ID dokumentů může invertovaný list pro každý term obsahovat další informace, jako jsou množina pozic, na kterých se term v dokumentu nachází, či četnost (frekvence) termu v dokumentu. Tento způsob je hojně využívaný zejména díky paměťové efektivitě a rychlosti vyhledávání.

Invertovaný index můžeme tedy chápat jako ADT, kde může existovat několik různých typů takovýchto indexů. Všechny tyto indexy by však měly podle Buttchera Clarka a Cormaka [3] implementovat následující 4 operace:

`first(t)` – vrátí první pozici/dokument, na které / ve kterém se term `t` vyskytuje,

`last(t)` – vrátí poslední pozici/dokument, na které / ve kterém se term `t` vyskytuje,

`next(t, current)` – vrátí první pozici/dokument, na které / ve kterém se term `t` vyskytuje za `current` pozicí/dokumentem,

`prev(t, current)` – vrátí poslední pozici/dokument, na které / ve kterém se term `t` vyskytuje před `current` pozicí/dokumentem.

Samotný způsob vyhledávání pomocí invertovaného indexu pak závisí na jeho konkrétní implementaci. Může se jednat například o mapu spojových seznamů či hash tabulku. Každý způsob je vhodný na konkrétní případy použití a velikosti slovníku. Obecný způsob použití je ale stejný:

1. Uživatel zadá textový dotaz.
2. Vyhledávací systém zpracuje uživatelův dotaz, provede tokenizaci s normalizací a vzniknou termy.
3. Tyto termy se pak hledají v invertovaném indexu jako klíče.
4. Dojde k vrácení seřazených dokumentů, které jsou relevantní k zadaným termům.

1.5 Získání a ohodnocení relevantních výsledků (retrieval a ranking)

Nejjednodušší způsob, jakým lze výsledky získávat (tedy označovat za relevantní), je pomocí přesné textové shody všech termů v dotazu. Pokud tedy dokument obsahuje všechny termy v dotazu, je vrácen jako relevantní výsledek. Podle [3] přestane však tento způsob velmi rychle fungovat pro delší dotazy, které mohou jen více specifikovat žádanou informaci, nebo pro dotazy se synonymy. Z těchto důvodů se používají jiné způsoby a metriky, mezi něž patří například *frekvence termů v dokumentu* či *blízkost termů mezi sebou*.

Frekvence termu Frekvence termu je číslo, které udává počet výskytů daného termu t v daném dokumentu d. Dokumenty s vyšší frekvencí všech termů z dotazu by měly mít vyšší váhu než dokumenty s frekvencí nižší. Otázkou je, jakým způsobem hodnotit dokumenty, které mají součty frekvencí termů stejné, ale jednotlivé frekvence termů rozdílné – například term t_1 je v dokumentu d_1 3x a t_2 4x, zatímco term t_1 je v dokumentu d_2 6x a t_2 1x. V takovém případě se může využít dalších metrik pro přepočítání váhy.

Blízkost termů Blízkost termů, jak už je z názvu patrné, si všímá, jak blízko sobě jednotlivé termy jsou v daném dokumentu. Obecně platí, že čím blíže si termy jsou, tím vyšší váhu dokument má. Tento způsob však nebude dobře fungovat v případě názvů či jmen, např. spousta dokumentů by měla velmi stejnou váhu pro dotaz „William Shakespear".

Z výše popsaných metrik je jasné, že využití pouze jedné, navíc takto základní, není dostatečné a v drtivé většině případů je nutné použít kombinaci různých metrik pro přesnější vážení. Hojně využívanými metrikami v této oblasti jsou *TF-IDF* a *BM25*.

1.5.1 TF-IDF

TF-IDF kombinuje frekvenci termu v dokumentu (TF) a převrácenou frekvenci termu ve všech dokumentech (IDF), přesněji:

frekvence termu i v dokumentu j – $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, kde $n_{i,j}$ je počet výskytů termu i v dokumentu j a $\sum_k n_{k,j}$ je délka dokumentu j (počet termů v dokumentu),

převrácená frekvence termu i – $idf_i = \log \frac{|C|}{|j:t_i \in d_j|}$, kde $|C|$ je počet všech dokumentů v kolekci a $|j : t_i \in d_j|$ je počet dokumentů, které obsahují term i – měří tedy, jak moc informace term i poskytuje (jak běžný či vzácný je napříč kolekcí).

Výsledná váha (skóre) podle TF-IDF se vypočítá jako součin $tf_{i,j} \cdot idf_i$. Tato metoda tedy udává váhu slovům (termům) na základě toho, jak často se vyskytují v dokumentu a jak unikátní jsou v rámci celé kolekce dokumentů.

1.5.2 BM25

Turnbull [6] uvádí, že BM25 (neboli „Best Match 25") vylepšuje TF-IDF. Má své kořeny v pravděpodobnostním modelu, ve kterém je určování relevance dokumentu založeno na pravděpodobnosti.

BM25 mění způsob počítání IDF, které může mít nově i negativní skóre pro term s vysokou frekvencí v dokumentu. Počítání TF části je také jiné, více tlumí vliv frekvence termu než TF-IDF. Vliv četnosti termů je vždy rostoucí, ale asymptoticky se blíží k určité hodnotě. Pokud nebude zvažována délka dokumentu, bude vzorec pro TF vypadat takto: $\frac{(k+1)\cdot tf}{k+tf}$. k je v tomto případě hodnota, ke které se výsledky asymptoticky přibližují, ale nikdy ji nepřekročí. Tato hodnota tedy slouží k „ladění" tohoto modelu a nejčastěji se nastavuje na hodnotu 1,2.

Předchozí vzorec se však v takovéto formě nepoužívá – neuvažuje délku dokumentu. Typicky je BM25 TF skóre ovlivněno tím, jestli má dokument délku větší nebo menší než je průměrná délka všech dokumentů v kolekci. Do vzorce jsou tedy přidány dvě nové proměnné: konstanta b (umožňuje definovat vliv délky L na vážení) a délka L (jak relativně je dokument dlouhý k průměrné délce všech dokumentů: $L = \frac{|d|}{avg|D|}$). Výsledný vzorec pro počítání BM25 TF

je tedy $\frac{(k+1)\cdot tf}{k\cdot(1.0-b+b\cdot L)+tf}$. Kratší dokumenty tak dosáhnou asymptoty mnohem rychleji než dokumenty delší.

Výsledné skóre BM25 se počítá následovně: $IDF \cdot \frac{(k+1)\cdot tf}{k\cdot(1.0-b+b\cdot\frac{|d|}{avg|D|})+tf}$

Kapitola 2

Rešerše

V předchozí práci [1] se autor zaměřil na typy vyhledávacích systémů (webový vyhledávač, agregátor, metavyhledávací systém...). V rámci této rešerše se autor zaměří především na funkcionalitu vyhledávacích knihoven a enginů, tedy jak přesně fungují, jak zpracovávají a ukládají text a jaké způsoby využívají pro získávání relevantních výsledků na zadané dotazy. Na konci této kapitoly provede autor rešerši původního prototypu WhereIS a identifikuje problémy, které se v prototypu nacházejí.

2.1 Rešerše podobných vyhledávacích systémů

Aplikace, ve kterých jsou hlavní funkcionality realizovány vyhledávacím enginem, se nazývají anglickým termínem *Search-Based Applications (SBA)*. Jak uvádí Foglia [7], tyto aplikace nemusí být na první pohled čistě o vyhledávání, může se jednat například o aplikace umožňující čtení e-mailů či koukání na videa. Důležité však je, že na pozadí jsou data uložená v indexu a jsou získávána pomocí jednoho nebo více vyhledávacích dotazů. Tyto aplikace zároveň poskytují dobrou výkonnost nezávisle na velikosti dat. V době psaní této práce existuje velké množství takovýchto aplikací, které využívají již hotová řešení. Může se jednat jak o zpoplatněné služby, tak o volně dostupné vyhledávací enginy a knihovny. Jelikož se dává přednost již hotovým řešením za účelem ušetření času, rozhodl se autor v rámci této rešerše soustředit na dostupné vyhledávací knihovny či enginy, zejména na to, jak zpracovávají text, indexují či jak určují, že daný výsledek je relevantní, protože to je pro zlepšení vyhledávacích schopností systému WhereIS to nejdůležitější. Je důležité také zmínit, že všechny vyhledávací knihovny a enginy zmíněné v této kapitole, jsou open-source. Je to z toho důvodu, že existují publikace detailně popisující jejich fungování a je možné si kód otevřít a zanalyzovat. Existuje samozřejmě i velké množství populárních enterprise řešení. Mezi nejznámější patří například *Elasticsearch*.

2.1.1 Lucene

Jak uvádí Białecki, Muir a Ingersoll [8], Lucene je moderní open-source vyhledávací knihovna, která poskytuje jak relevantní výsledky na dotazy, tak i vysokou výkonnost. Poskytuje API na provádění typických operací jako jsou indexace, dotazování, jazyková analýza a další. Lucene se stará o vyhledávání

ve velkém množství populárních aplikací a zařízení, jako jsou například *Twitter (X)*, *Netflix* či *Instragram*. Hlavní schopnosti Lucene jsou orientovány na vytváření, udržování a přistupování k jejímu invertovanému indexu. Popularita této knihovny je zřejmá také z velkého množství jiných vyhledávacích knihoven a enginů, které vychází z Lucene, či ji čistě používají. Mezi ně patří například *Elasticsearch* či *Solr*.

Zpracování textu Lucene jak samotné dokumenty v kolekci, které indexuje, tak dotazy zpracovává a konvertuje do interní reprezentace. Během indexace vytváří tokeny, které ukládá do svého invertovaného indexu. Pro dotazy slouží tokeny k vytvoření požadované vnitřní reprezentace, která umožní dotazování nad invertovaným indexem. Samotné zpracování sestává ze tří částí:

1. Nepovinná filtrace znaků a normalizace (např. odstranění diakritiky, převod na malá/velká písmena…).
2. Vytvoření tokenů.
3. Filtrace tokenů (stematizace, lemmatizace, odstranění stop slov…).

Indexace a úložiště V tomto ohledu jmenují Białecki a spol. [8] například tyto schopnosti:

- indexace uživatelsky definovaných dokumentů, kde se dokumenty mohou skládat z více různých polí, ve kterých mohou mít obsah (nadpis, podnadpis, obsah dokumentu…),
- téměř real time indexace umožňuje vyhledávat dokumenty ihned po dokončení jejich indexace,
- podpora transakcí pro přidávání a odebírání dokumentů,
- podpora pro různé typy termů, dokumentů a korpusů, což umožňuje různé možnosti rankingu.

Dotazování Lucene nabízí hned několik způsobů dotazování včetně podpory filtrování, stránkování, řazení nalezených výsledků a také možnost poskytnout jakousi zpětnou vazbu na relevanci vrácených výsledků. Pro samotné dotazování nabízí přes 50 různých reprezentací dotazů včetně různých parserů. Obsahuje také předpřipravené typické vyhledávací modely (vektorový, jazykový, informačně založený…).

Konečný stavový automat Kromě již zmíněného zpracování textu je také důležitou součástí samotné architektury *konečný stavový automat (FST)*. Jedná se o *in-memory* implementaci invertovaného indexu, která podporuje konstrukci minimálního stavového automatu v lineárním čase, kompresi či vážení (ranking).

Dokumentový model Dokumenty jsou reprezentovány jako seřazený list polí, které mají jméno, obsah (data), přidělenou váhu (důležité pro počítání výsledné váhy) a další atributy. Dokumenty mohou mít pole se stejnými názvy, protože během indexace jim je přiděleno unikátní celočíselné ID, a jsou zpracovávány sekvenčně. Lucene definuje 2 základní typy polí:

indexové – obsah pole je nejdříve zpracován do tokenů a ty jsou následně uloženy do in-memory segmentů,

ukládací – obsah pole je uložen tak, jak je.

Index a jeho aktualizace on-line Vždy, když jsou vloženy nové dokumenty, jsou zpracovány stejným způsobem, jak autor popsal výše. Výsledná reprezentace je uložena do *segmentů*. Tyto segmenty jsou následně periodicky ukládány do perzistentního úložiště a tím se také aktualizuje samotný invertovaný index. Jednotlivé segmenty jsou *write-once*, tedy po prvotním zápisu je již nelze měnit. Pro úpravu již takto zapsaného segmentu je nutné vytvořit nový. O tyto operace se stará třída `IndexWriter`, která se kromě samotného zápisu segmentů stará také o udržování historie zapsaných segmentů do paměti (tzv. *commity*).

Vyhledávání Lucene pro vytváření interních dotazů využívá `Query`. Tyto objekty mohou pomocí *chainování* vytvářet komplexní dotazy s různou striktností. Podporuje také využívání zástupných znaků či regulárních výrazů. Typicky jsou pak dotazy zpracovány třídou `QueryParser` a převedeny do stromové struktury (není to však nutné). Následně je spuštěno samotné vyhledávání, během kterého je každý segment invertovaného indexu zpracován sekvenčně, a pro každý takovýto segment je vypočítáno skóre (rank). Po zpracování všech segmentů invertovaného indexu `Collector` třída zpracuje všechna tato skóre a pomocí zvolené strategie (např. top-N) vrátí seřazené relevantní výsledky.

Počítání váhy (scoring, ranking) O počítání váhy (relevance) se v Lucene podle [8] stará `Similarity` třída. Ta bere v potaz statistiky termu, globální statistiky indexu a specifikace samotného dotazu (např. vzdálenosti mezi termy, počet odpovídajících termů...). Lucene také nabízí předpřipravené klasické skórovací modely jako jsou TF-IDF s různými normalizacemi, BM25, jazykový model a další. Sharma [9] také uvádí, že Lucene nepočítá skóre na základě dokumentů, nýbrž na základě jednotlivých polí (fields), a že základním skórovacím modelem, který Lucene využívá, je právě TF-IDF.

2.1.2 Indri

Strohman a spol. [10] zmiňují, že Indri je vyhledávací engine, který je součástí Lemur projektu[1] a byl vyvinut tak, aby splňoval následující cíle:

- dotazovací jazyk by měl podporovat komplexní dotazy s možností specifikovat velké množství pravidel,

[1] Projekt Lemur vyvíjí vyhledávače, panely nástrojů pro prohlížeče, nástroje pro analýzu textu a datové zdroje, které podporují výzkum a vývoj softwaru pro vyhledávání informací a vytěžování textu [11].

- vyhledávání by mělo být velmi efektivní napříč různými typy dotazů a typů dokumentů (webové, vícejazyčné...),
- dotazování a vyhledávání by mělo fungovat na různých úrovních granularity (věty, části textu, XML elementy, celé dokumenty...),
- systémová architektura by měla podporovat obrovskou databázi dat s možností rychlého indexování a optimalizovaného dotazování.

Zpracování textu Indri v základu obsahuje parsery známých dokumentových formátů jako XML, HTML či čistě textové. Dokumenty jsou převedeny do `ParsedDocument` reprezentace, kterou indexer ukládá přímo. `ParsedDocument` obsahuje list termů, které dokument obsahuje, a informace o jednotlivých polích dokumentu. Navíc také obsahuje celý původní text a lokaci všech termů v tomto textu. To se používá zejména v případě, kdy uživatel chce vidět zvýrazněné dotazované termy ve vráceném dokumentu.

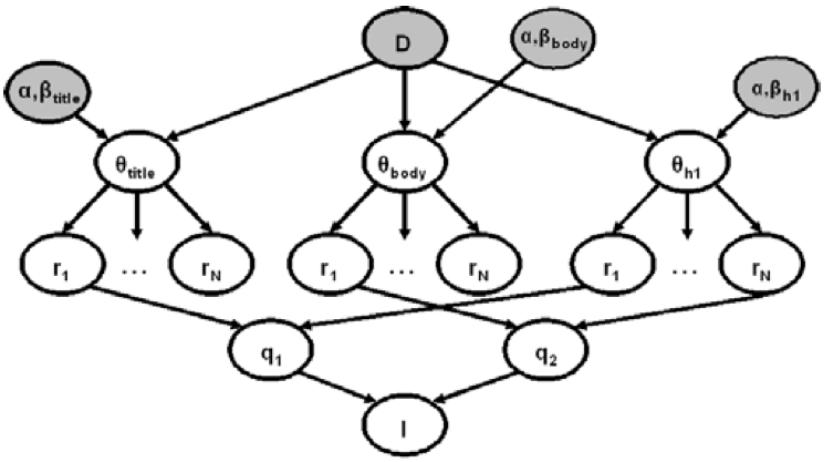
Indexace Indexační systém využívá komprimované invertované listy pro každý term a pole dokumentu v paměti. Periodicky je pak tato paměť zapisována na disk. Každá tato paměťová buňka je soběstačná a dokáže provádět vyhledávání. Indri index lze tedy považovat jako množinu menších indexů.

Vyhodnocování dotazů Indri kombinuje jazykový model s inferenční sítí, což umožňuje vyhodnocovat strukturované dotazy pomocí odhadů jazykového modelování v rámci sítě, nikoliv pomocí TF-IDF. Dokumenty jsou následně seřazeny podle $P(I|D, \alpha, \beta)$, což je pravděpodobnost, že informační potřeba I je splněna, když jako důkaz slouží dokument D a hyperparametry α a β.

Reprezentace dokumentů Dokument je reprezentován jako sekvence termů. Na základě této sekvence je podle [10] odhadnut multinomický jazykový model nad slovní zásobou. Modelace tohoto jazyka se zabývá čistě výskytem slov. Toho lze docílit pomocí binárních vektorů, jejichž dimenze je rovna velikosti slovníku. Každé slovo v dokumentu je pak zakódováno pomocí vektoru, který má jako jedinou nenulovou složku term, který dané slovo reprezentuje. Samozřejmě není tento způsob reprezentace jediný. Může být např. užitečné označovat fakt, jestli se slovo objevuje na konci věty, nebo jestli slovo začíná velkým písmenem. Všechny tyto vlastnosti lze opět reprezentovat pomocí binárních vektorů.

Inferenční síť Při používání inferenčních sítí se předpokládá, že se dotaz skládá z řady konceptů jako mohou být termy, fráze či složitější vazby. Dokument je následně považován za relevantní právě tehdy, když obsahuje koncepty uvedené v dotazu. Strohman a spol. [10] také upozorňují na fakt, že koncept typu term není to samé jako term vyskytující se čistě v dokumentu, např. dokument může obsahovat slovo „terorismus“, ale jeho obsah nemusí mít s terorismem nic společného. Jazykové modely θ v inferenčních sítí jsou odhadovány právě pomocí hyperparametrů α a β. Pro stejný dokument se používají různé dvojice hodnot těchto hyperparametrů podle konkrétní části dokumentu (název, tělo, obsah elementu `h1`...). Z těchto modelů jsou následně

vytvořené *features* dokumentů r, které jsou použity pro určení relevance k dotazu q. Příklad takové inferenční sítě je vidět na obrázku 2.1.



Obrázek 2.1: Příklad inferenční sítě, převzato z [10]

Dotazování Indri nabízí komplexní dotazovací jazyk umožňující vytváření komplexních konceptů. K tomu jsou využívány operátory, každý z nich se dá považovat za dotazovací uzel q v inferenční síti. Pomocí těchto operátorů lze například indikovat důležitost lokace termu v dokumentu, nebo že by se příslušné termy měly objevovat v daném pořadí, či že by se termy měly vyskytovat blízko sebe. Navíc Indri také obsahuje operátory, které berou v potaz strukturu dokumentu, např. že daný term je relevantní, pokud se vyskytuje pouze v příslušném poli/části.

Vyhledávání Dotaz je při vyhledávání převeden do interní reprezentace, která následně podléhá různým transformacím, např. kvůli efektivitě či převedení komplexních operátorů na jednodušší. Vyhledávání pak probíhá paralelně napříč různými menšími indexy. Následně probíhají dvě fáze:

1. fáze – vypočítají se statistiky, kolikrát se daný term či fráze vyskytují v kolekci.

2. fáze – statistiky z první fáze se použijí pro vyhodnocení dotazu proti kolekci.

2.1.3 Sphinx

Dle [12] je Sphinx open-source, plně textový vyhledávací server, jehož hlavními rysy jsou výkonnost, vyhledávací kvalita (relevance) a jednoduchá integrace. Umožňuje vytvářet vyhledávací indexy nad daty uložené v jakékoliv databázi, která podporuje ODBC. Obsahuje typické nástroje pro zpracování textu a nabí-

zí dotazovací jazyk *SphinxQL*, který je velmi podobný tradičnímu SQL. Mezi jeho hlavní funkce patří:

- 2 základní typy indexů – batchový pro offline konstrukci či real-time index pro aktualizace za běhu aplikace,
- podpora pro různé typy atributů, které mohou být i jiné než čistě textové,
- podpora všech SQL databází podporujících ODBC (MySQL, PostgreSQL, Oracle, SQLite...) a Non-SQL databází pomocí streamování dat do XML formátu,
- jednoduchá integrace do existujících aplikací (Java, PHP, Python, C...) pomocí SphinxAPI,
- vylepšený řadící systém, který kromě klasických statistických metrik používá také blízkost termů a řadí bližší shody frází výše, navíc je řazení flexibilní – volba z řady vestavěných funkcí relevance, úprava vah či tvorba úplně nových funkcionalit.

Zpracování textu Sphinx plně podporuje SBCS a UTF-8 textové kódování. Z textu odstraňuje stop slova, využívá stematizace a lemmatizace a také řeší morfologii a synonyma pomocí slovníků slov a již zmíněné stematizace. Sphinx má navíc podle [13] v sobě zabudovaný stemmer i pro český jazyk. Pro anglický jazyk Sphinx také nabízí 2 fonetické algoritmy, a to *Soundex* a *Metaphone*, jak zmiňuje Yantsan [14]. Tyto algoritmy nahradí slova v dotazu speciálními fonetickými kódy, což umožňuje enginu zacházet s těmito slovy jako významově jinými, ale foneticky velmi podobnými až stejnými. To může být vhodné v tu chvíli, když se vyhledává například pomocí jména či příjmení. Sphinx také umožňuje definovat vlastní formy slov. Díky tomu lze například „S02E02“ nahradit během vyhledávání za „season 2 episode 2“.

Indexace a úložiště Jelikož Sphinx pracuje nad databází dat, jsou zde indexy velmi podobné tabulkám v SQL databázích, konkrétně se jedná o polostrukturovanou kolekci dokumentů (lze chápat jako řádek v SQL databázích). Základní datovou strukturou je dle [15] *full-textový index*. Vše ostatní je nadstavbou tohoto indexu. Z hlediska schémat kombinuje Sphinx to nejlepší z obou světů. Pokud je typ daného „sloupce“ známý, lze použít statické typové atributy (lze chápat jako sloupec v SQL databázích) pro největší efektivitu. Pokud však typ není známý, nic se neděje. Pro takové případy lze využít *JSON atribut*.

Architektura Základním stavebním kamenem jsou *indexy*. Poté následují podle [15] *dokumenty*, což jsou čistě listy s pojmenovanými textovými poli a libovolně zadanými atributy. Poté jsou samotná *pole*, což jsou texty, které Sphinx indexuje a umožňuje je vyhledávat na základě klíčových slov. Jsou vždy indexovány jako full-text index. S poli jsou úzce spjaté již zmíněné *atributy*. Atributy ve Sphinx lze chápat jako datové typy a je jich podporováno velké množství (UINT, BIGINT, FLOAT, BOOL, STRING, JSON...). Udávat atributy k jednotlivým polím má velké výkonnostní výhody, protože Sphinx přesně ví velikost daného atributu a uloží jej do přesně tak velkého paměťového prostoru. Jeho cílem je totiž mít veškeré atributy a nejlépe i celá indexovaná data

v paměti RAM. Úplně nejvýše jsou v této architektuře *schémata*. Schéma je seřazený list sloupců (pole + atributy). Často se ve Sphinx stává, že existuje několik různých schémat v rámci jednoho indexu či dotazu. Vždy však musí existovat *indexační schéma*, které definuje všechna indexovaná pole a atributy. Pro `SELECT` dotazy se následně používá *result set schéma*, které udává strukturu vráceného výsledku (jaké sloupce/pole se vrací, jaké mají atributy...). V neposlední řadě existuje také *vkládací schéma*, které se využívá při `INSERT` operaci.

Vyhledávání Full-textové vyhledávání je ve Sphinx bráno jako „bags of words" a všechna klíčová slova v dotazu musí být přítomna i v dokumentu. Provádí se tedy striktní booleanovské AND nad všemi klíčovými slovy. Samozřejmě samotné dotazování je dle [15] mnohem komplexnější. Sphinx má proto svůj vlastní dotazovací jazyk, který se používá uvnitř klauzule `MATCH()` v `SELECT` dotazu. Mezi hlavní koncepty tohoto jazyka patří *operátory* (AND, OR...), které dávají klíčová slova v dotazu do určitého vztahu, a *modifikátory*, které jsou vázané k jednotlivým klíčovým slovům. Mezi takové modifikátory patří například modifikátor přímé shody, modifikátor začátku/konce pole (dané klíčové slovo musí být na začátku/konci pole) či modifikátor IDF zlepšení, který vynásobí IDF klíčového slova o zadanou hodnotu. Vyhledávat lze samozřejmě přímo z aplikace pomocí SphinxAPI, kde se definují klíčová slova vyhledávání, index, ve kterém se bude vyhledávat, a další parametry jako váhy jednotlivých polí či způsob řazení výsledků.

Počítání váhy (ranking) Dle [15] umožňuje Sphinx specifikovat vlastní funkci pro počítání váhy/ranku výsledků. Tomuto mechanismu se říká *expression ranker*. Výrazy (expressions) mají přístup k několika speciálním proměnným, kterým se říká *ranking factors* (či *ranking signals*). Jedná se o velké množství různých vypočítaných hodnot na základě aktuálního vyhledávacího dotazu pro každý dokument a pole v dokumentu. Ve Sphinx existují tři typy (úrovně) těchto faktorů:

dotazovací (query) – hodnoty závisející pouze na vyhledávacím dotazu a ne na dokumentu (např. počet slov v dotazu), jsou vypočítány jednou na začátku vyhledávání,

dokumentové (documents) – hodnoty závisející jak na dotazu, tak na dokumentech relevantních k danému dotazu (např. počet slov v dokumentu), jsou počítány pro každý nalezený relevantní dokument,

faktory pole (field) – hodnoty závisející jak na dotazu, tak na shodě s full-textovým polem (např. počet slov), jsou počítány pro každé pole a proto musí být agregovány do singulární hodnoty pomocí agregační funkce (suma, největší ze všech...).

Jednotlivé faktory z různých úrovní lze pak kombinovat mezi sebou k vytvoření vlastní rankovací (vážící) funkce. Lze samozřejmě využít již předpřipravených rankovacích funkcí:

IDF – několik různých IDF možností,

BM25, BM25F – je nutné udržovat délky polí pro každý dokument a průměrné délky polí,

použití jiných klíčových slov než pro vyhledávání – někdy je potřeba hodnotit podle jiných klíčových slov než podle těch, kterými se vyhledávalo,

a spoustu dalších...

2.1.4 Shrnutí

Autor v rámci rešerše popsal architekturu a způsob fungování 3 vyhledávacích knihoven a enginů. Každý z těchto nástrojů je architektonicky odlišný. Lucene využívá v oblasti vyhledávání tradičnější architektury, Indri využívá architekturu inferenčních sítí pro práci s obrovskou kolekcí velkých dokumentů a Sphinx je vyhledávací server pracující nad daty uloženými v databázi. Ačkoliv se tyto architektury a způsoby používání relativně liší, všechna řešení využívají na pozadí stejných principů:

zpracování textu – všechna řešení zpracovávají text pomocí stejných nástrojů – odstranění stop slov a následná normalizace (stematizace, lemmatizace, odstranění interpunkce, převedení na malá písmena...),

indexace – všechna řešení používají na pozadí invertovaný index pro indexaci jednotlivých termů a dokumentů, protože tento způsob indexace je paměťově efektivní a umožňuje rychlé vyhledávání,

vyhledávání – všechna řešení vyhledávají na základě textových dotazů, na tento text pak aplikují stejné metodiky zpracování textu jako v případě dokumentů v kolekci, následná implementace samotného vyhledávání se liší podle použité architektury (přímé vyhledávání v invertovaném indexu, využití inferenčních sítí, SQL dotazy nad indexy v databázi),

ranking – všechna řešení nabízejí různé předpřipravené rankovací funkce (zejména TF-IDF či BM25), uživatel si však dokáže jednotlivé funkce upravovat podle svých potřeb či dokonce implementovat své vlastní. Použití konkrétní rankovací funkce závisí na konkrétních případech vyhledávání.

2.2 Rešerše existujícího prototypu WhereIS

Aby autor mohl v této práci navrhnout a integrovat zlepšení do původního prototypu systému WhereIS, musí nejprve analyzovat původní implementaci a vytyčit problémy, které se v ní nacházejí, případně místa v kódu, která by se dala zlepšit. Nalezené poznatky autor rozepíše z hlediska jejich povahy. Pokud se bude jednat o chyby, autor uvede způsob, jak je možné chybu zreprodukovat. Pokud se bude jednat o místo zefektivnění či obecné zlepšení, autor popíše, o jaké zlepšení se jedná. Návrh na řešení problémů se nachází v kapitole 4.4.

```
/**
 * @Route("/search", name="search")
 * @param SessionInterface $session
 * @param Request $request
 * @return Response
 */
public function renderSearch(SessionInterface $session,
                             Request $request): Response
{
    if($session->get('authenticated') === null) {
        if($session->get('state') !== $request->get('state'))
            return $this->redirectToRoute('index');
        else
            $session->set('authenticated', true);
    }

    ...
}
```

Výpis kódu 2.1: Chybná implementace autorizace

2.2.1 Autorizace uživatelů

V původním prototypu aplikace se nachází bezpečností chyba při autorizaci uživatelů oproti školnímu autorizačnímu serveru *ZuulOAAS*, která umožňuje uživateli dostat se do systému WhereIS i přes neprovedené nebo neúspěšné přihlášení.

Příčina chyby Chyba je způsobená absencí kontroly hodnoty query parametru `state` v metodě `renderSearch` třídy `SearchController`. Uživateli tedy stačí zkopírovat vygenerovanou hodnotu query parametru `state` a musí vědět, že po přihlášení dochází k přesměrování na stránku */search*. Chybná implementace kontroly je vidět ve výpisu 2.1 a implementace renderování úvodní stránky společně s předgenerováním autorizačního odkazu je vidět ve výpisu 2.2.

Reprodukce chyby Uživatel klikne na tlačítko *Přihlásit se* na úvodní stránce aplikace a je následně přesměrován na autorizační server OAuth 2.0 *ZuulOAAS*, kde je vyzván k zadání svých přihlašovacích údajů. Uživatel si však všimne URL adresy stránky, která vypadá nějak takto `https://auth.fit.cvut.cz/oauth/authorize?client_id=...&response_type=...&scope=cvut:umapi:read&cvut:sirius:personal:read&state=49f20d`. Uživatel si zkopíruje hodnotu query parametru `state` a tu použije pro přihlášení se do systému WhereIS pomocí odkazu v podobě `https://server:port/search/?code=...&state=49f20d`.

2.2.2 Špatné oddělení scopes v URL

V původním prototypu se používá šablona pro generování autorizačního odkazu. V této šabloně se používá k oddělení příslušných `scopes` špatný znak. Z toho důvodu žádá autorizační server při autorizaci uživatele o povolení pouze

```
/**
 * @Route("/", name="index")
 * @param SessionInterface $session
 * @return Response
 */
public function renderIndex(SessionInterface $session): Response
{
    if($session->get('authenticated') === true) {
        return $this->redirectToRoute('search');
    }

    $state = ...
    $authorizationLink = ...
    $session->set('state', $state);

    return $this->render("index.html.twig", [
        'authorizationLink' => $authorizationLink
    ]);
}
```

Výpis kódu 2.2: Implementace renderIndex metody s předgenerováním odkazu

```
/**
 * Generates link on authorization server OAuth 2.0 ZuulOAAS
 * for user authorization.
 * @param string $state - state for authorization used
 * for validity
 * @return string
 */
public static function getUserAuthorizationLink(...): string
{
    return self::DEFAULT_AUTHORIZATION_URL
        . '?client_id=...'
        . '&response_type=...'
        . '&scope=cvut:umapi:read&cvut:sirius:personal:read'
        . '&state=...';
}
```

Výpis kódu 2.3: Špatný znak pro oddělení scopes

jednoho (prvního) `scope` ze 2 požadovaných. Navíc jsou názvy těchto `scopes` součástí kódu, což ztěžuje jejich rozšiřitelnost či znovupoužitelnost.

Příčina chyby Chyba je způsobena použitím špatného oddělovače v autorizačním odkazu, v tomto případě se jedná o znak „&“, jak je vidět v 2.3.

Reprodukce chyby Chybu lze zreprodukovat přihlášením do aplikace, kdy po vypršení potvrzení zažádá autorizační server *ZuulOAAS* znovu o povolení žádaných `scopes` aplikací WhereIS.

```
$rank = 0;
$courseWordsNum = count(explode(' ', $course['name']));
foreach($words as $word) {
    if(str_contains(strtoupper($course['name']), $word))
        $rank++;

    // If one word is substring of the code,
    // then it is considered as wanted.
    if(str_contains($course['code'], $word)) {
        $rank = $courseWordsNum;
        break;
    }
}
```

Výpis kódu 2.4: Implementace počítání váhy (ranku) pro předmět

2.2.3 Chybějící indexy v databázi

V tomto případě se nejedná o chybu, ale o neefektivitu. Celá aplikace WhereIS je založena na vyhledávání a u vyhledávacích platforem je rychlost získání informací klíčová. Bez indexů nad sloupci textů, nad kterými se vyhledává, bude databázový stroj vždy volit *full table scan*. Full table scan, jak už je z anglického názvu patrné, znamená čtení (skenování) celé tabulky bez použití indexu. S použitím indexu lze tak číst mnohem méně datových bloků, než je během vyhledávání nutné. To platí samozřejmě v tom případě, kdy počet datových bloků na udržování indexu není větší, než počet datových bloků nezbytných pro uložení celé tabulky.

2.2.4 Počítání rank hodnoty výsledků

V rámci prototypu je použit pro počítání rank hodnoty, reprezentující relevantnost výsledku k dotazu, jednoduchý způsob. Na začátku platí $rank = 0$. Vezme se každé slovo dotazu jedno po druhém a zkontroluje se, jestli je podřetězcem nějakého ze slov názvu či kódu (username, kód místnosti, kód předmětu) výsledku. Pokud je slovo podřetězcem některého ze slov v názvu výsledku, pak $rank \mathrel{+}= 1$. Pokud je slovo podřetězcem kódu výsledku, pak $rank = len(words)$. Nakonec se hodnota normalizuje, tedy $rank \mathrel{/}= len(words)$. Pokud je tedy jedno ze slov dotazu podřetězcem kódu, má automaticky výsledek největší rank (váhu). Výsledky jsou pak uživateli zobrazeny seřazené sestupně podle ranku. Příklad implementace počítání váhy pro předmět je vidět ve výpisu 2.4. Z popisu a příkladu implementace je jasné, že se jedná o naivní způsob počítání ranku, který přidá velkou hodnotu i nežádoucím výsledkům a v rámci rešerše v 2.1 vyplynulo, že je potřeba využít sofistikovanějších metod pro počítání váhy/ranku.

2.2.5 Propojenost a celistvost dat

Data jsou v prototypu ukládaná do databáze pomocí příkazu `fetch`. Pro všechny tři původní entity (učitel, předmět, místnost) se používá pouze 1 zdroj dat (KOS). UserMap se využívá pouze pro kontaktní údaje učitele a Sirius pro

události. Kvůli tomu nebudou v databázi všechna data, protože systém KOS poskytuje informace pouze o učitelích a místnostech, ve kterých se učí. Z toho důvodu nebudou v databázi ostatní zaměstnanci fakulty (např. studijní referentky) či kanceláře učitelů. Na druhou stranu tohoto kontextu, co KOS nabízí, lze využít a je tak možné si ke každému zaměstnanci či místnosti, který/á byl/a získán/a z KOSu, uložit informaci „je učitel“ / „je učebna“.

Samotná data v databázi nejsou také nijak propojená. Tabulky neobsahují žádné cizí klíče, a tím pádem nelze třeba zjistit, jací učitelé nalezený předmět učí.

2.2.6 Problémy fungování příkazu `fetch`

Prototyp obsahuje implementaci příkazu `fetch`, který je manuálně spouštěn před zapnutím systému WhereIS. Cílem tohoto příkazu je stažení dat ze školních systémů a jejich uložení do databáze aplikace. Příkaz do databáze ukládá data, která jsou aktuální pro běžící semestr. Je implementován použitím balíčku *Symfony Command* a pro každou entitu je implementována třída `Fetcher`, která se stará o získání a uložení dat souvisejících s konkrétní entitou. Architektura tříd příkazu je vidět na obrázku 2.2.

Problémy ve fungování příkazu jsou následující:

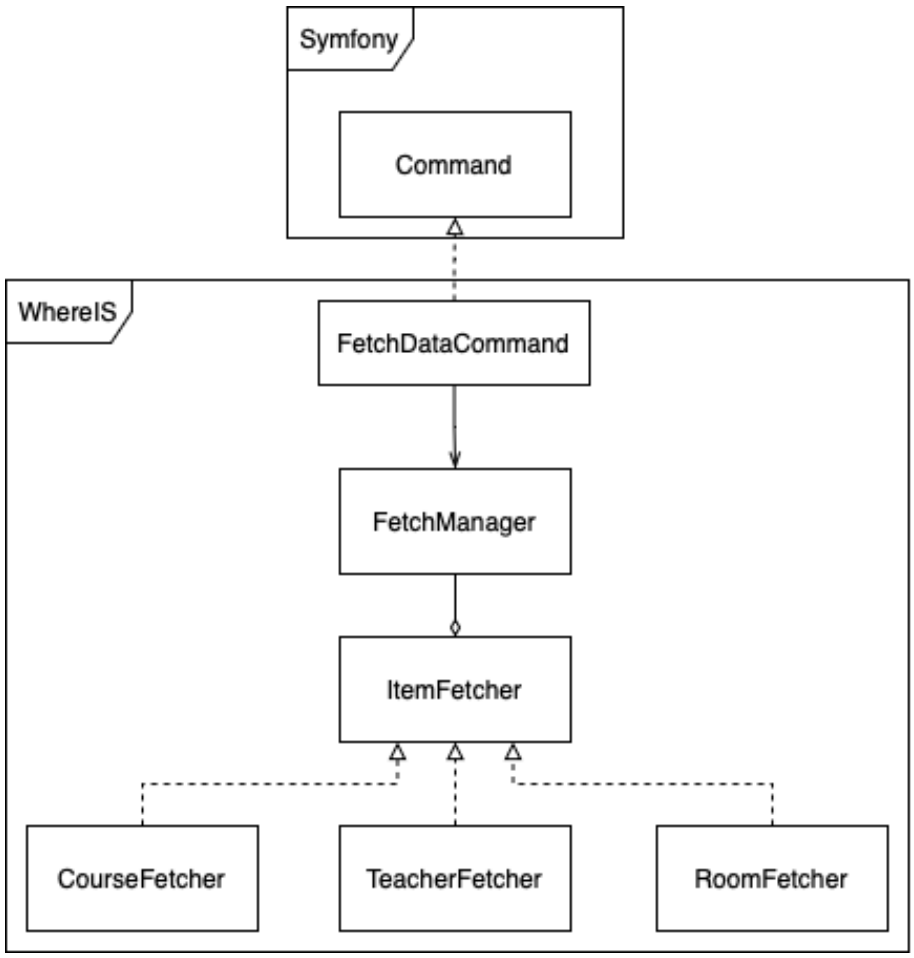
- příkaz je spouštěn manuálně bez žádné automatizace s použitím data či času,
- příkaz je možno spustit pouze v případě, kdy aplikace neběží,
- příkaz je nutné spouštět každý nový semestr, jinak budou v databázi data z minulého semestru.

2.2.7 Příprava databáze pro příkaz `fetch`

Jak je uvedeno na obrázku 2.2, součástí příkazu `fetch` jsou jednotlivé `Fetcher` třídy. Všechny tyto třídy implementují rozhraní `ItemFetcher`, které obsahuje metodu `prepareDB`. Tuto metodu mají všechny třídy stejnou, její implementace je vidět ve výpisu 2.5. Tato implementace není problematická pro aktuální fungování prototypu, protože tabulky či data v tabulkách nejsou navzájem nijak propojené. Situace by se však změnila, kdyby se do tabulek integrovaly cizí klíče. V takovém případě se ztratí provázanost dat v tabulkách a v případě špatného nastavení databáze může dojít i k chybě (když např. nebude použito `CASCADE`). Navíc některá data by se mazat z databáze vůbec neměla (např. uživatelské nastavení). V takovém případě je nutné implementaci metody změnit.

2.2.8 Implementace vyhledávání relevantních výsledků

Vyhledávání relevantních výsledků funguje v prototypu čistě na základě textové shody. V rámci každé `Item` třídy, která implementuje rozhraní `Item` a každá z nich reprezentuje jednu entitu, se zavolá metoda `find`, která dotaz rozdělí na jednotlivá slova podle mezer a pro každé slovo se dotazuje do databáze na shodu pomocí operátoru `LIKE`. Příklad takové implementace metody `find` pro předmět je vidět ve výpisu 2.6. Pro všechny entity se dotazuje ve dvou



Obrázek 2.2: Architektura implementace `fetch` příkazu

sloupcích, a to *name* (název místnosti, předmětu, učitele) a *code*/*username* (kód předmětu či místnosti, uživatelské jméno učitele). Je důležité myslet na to, že porovnávání je *case sensitive*, proto se také jednotlivá slova transformují na *upper case* či *lower case*. Získaným výsledkům z databáze je poté vypočítán *rank* (váha) podle textové shody, vizte 2.2.4, a výsledky jsou následně v `SearchService` spojené do jednoho pole výsledků, které jsou seřazené podle ranku sestupně. Mezi top 5 relevantními výsledky mohou být tedy různé entity.

2.2.9 Způsob fungování vyhledávání

V prototypu WhereIS funguje vyhledávání tak, že uživatel nejdříve zadá celý dotaz do vyhledávacího pole na domovské stránce a poté klikne na tlačítko „Vyhledat“. Po určité chvíli jsou zobrazeny všechny relevantní výsledky na daný dotaz a zároveň ke každému výsledku všechny informace, které aplikace WhereIS k výsledku má. Tento způsob fungování v sobě skrývá několik problémů:

```
static function prepareDB(EntityManagerInterface $entityManager)
{
    $metadata = $entityManager->getClassMetadata(self::ENTITY);
    $schemaTool = new SchemaTool($entityManager);
    $schemaTool->dropSchema([$metadata]);
    $schemaTool->createSchema([$metadata]);
}
```

Výpis kódu 2.5: Implementace metody `prepareDB` ve `Fetcher` třídách

```
$searchedValue = strtoupper($searchedValue);
$words = explode(' ', $searchedValue);

$queryBuilder = ...;

$first = true;
$i = 1;
foreach ($words as $word) {
    if($first) {
        $first = false;
        $queryBuilder = $queryBuilder
            ->where('c.code LIKE :value' . $i)
            ->orWhere('UPPER(c.name) LIKE :value' . $i)
            ->setParameter('value' . $i++, '%' . $word . '%');
        continue;
    }
    $queryBuilder = $queryBuilder
        ->orWhere('c.code LIKE :value' . $i)
        ->orWhere('UPPER(c.name) LIKE :value' . $i)
        ->setParameter('value' . $i++, '%' . $word . '%');
}
```

Výpis kódu 2.6: Část implementace metody `find` v `Item` třídách

- vyhledávání začne až po zadání celého dotazu a kliknutí na tlačítko „Vyhledat“,
- na stránce s výsledky není vyhledávací pole, uživatel tedy nemá možnost zadat nový dotaz či upravit stávající dotaz, a musí tedy vždy přejít na domovskou stránku,
- na stránce s výsledky jsou ihned zobrazeny všechny informace k danému výsledku, což způsobuje následující:
  - uživatel je zahlcen množstvím informací zobrazených na stránce a stránka se tak pro něj stává méně přehledná,
  - ztěžuje to vyhledávání chtěných výsledků na stránce, protože informace jednotlivých výsledků nelze skrýt,
  - jedná se o neefektivní způsob fungování, kdy aplikace vrací a zobrazuje 99 % informací, které uživatel nevyužije a nechce vidět.

Autor práce si je vědom, že se jedná převážně o problém frontendové části aplikace, nicméně jeho řešení ovlivní i aktuální implementaci logiky vyhledávání, proto se autor po domluvě s kolegou Bc. Illiou Brylovem rozhodl zařadit tento problém i do této práce.

2.2.10 Citlivé údaje v kódu

Dalším zásadním bezpečnostním problémem v prototypu je fakt, že citlivé údaje jako `CLIENT_SECRET` jsou napsány přímo v kódu. Jelikož je kód následně verzován na fakultním `Gitlab` serveru, má přístup k těmto údajům pak každý, kdo má platný ČVUT účet. Z toho důvodu je nutné vygenerovat nové hodnoty `CLIENT_SECRET` a případně i `CLIENT_ID`, aby se tyto údaje nedaly zneužít. K resetování těchto údajů lze využít školní aplikaci *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).*“ [1]

Po vygenerování nových hodnot je potřeba tyto hodnoty uložit lepším, bezpečnějším způsobem, aby se nedaly jednoduše číst a nikdo je tak nemohl zneužít.

2.2.11 Konfigurace autowiringu tříd

V prototypu WhereIS se autowiring (automatické spojování) vlastnoručně vytvořených tříd používá ve dvou třídách, a to `SearchService` a `FetchManager`. Ve třídě `SearchService` se autowirují třídy implementující rozhraní `Item` (tedy např. `CourseItem` či `FreeRoomItem`) a konfigurace tohoto autowiringu je definována v konfiguračním souboru `services.yaml` ve složce *config*, kde každá jednotlivá `Item` třída je přidána do pole argumentů třídy `SearchService`. Ve třídě `FetchManager` se také používá pole tříd, a to `Fetch` tříd implementující rozhraní `ItemFetcher` (např. `CourseFetcher` či `RoomFetcher`). V této třídě se navíc kontroluje, jestli třída v daném poli opravdu implementuje potřebnou metodu `fetch`. Oba dva tyto způsoby nejsou ideální zejména z hlediska jednoduché rozšiřitelnosti, protože je každou takto nově vytvořenou třídu, implementující jedno z výše zmíněných dvou rozhraní, nutné zaregistrovat na potřebném místě.

Kapitola 3

Analýza

3.1 Požadavky uživatelů

Prvotní prototyp systému WhereIS umí již některé základní funkcionality. Mezi ně patří:

- vyhledávání předmětů, učitelů a místností,
- vyhledávání volných počítačových a seminárních místností v čase vyhledávání,
- zobrazování autorem definovaných informací k danému výsledku:

  učitel kontaktní údaje včetně jeho kanceláře, dvě nejbližší události (nefunkční kvůli nepřiděleným právům od školy),

  předmět odkaz na rozvrh předmětu a stránku předmětu (Courses či Moodle), dvě nejbližší události,

  místnost odkaz na rozvrh místnosti, plánek patra (na kterém se místnost nachází), dvě nejbližší události,

  volné místnosti reprezentovány jako seznam, kód místnosti funguje jako odkaz na rozvrh místnosti.

Jak je z výše uvedeného vidět, prototyp obsahuje velmi málo informací pro každou entitu. Za nejdůležitější by autor označil nejbližší události, kancelář učitele a plánek patra u místnosti, aby se místnost dala lépe najít. V tomto ohledu se dá systém určitě zlepšit. Pro jednotlivé entity mohou být přidány další užitečné informace, které by nejen pro studenty mohly být zajímavé.

Jedním z možných vylepšení by mohl být seznam učitelů, kteří učí daný předmět, či seznam předmětů, které daný učitel učí, dále odkaz na sylabus předmětu do KOSu či rozšíření entit o další, a to menzy a jejich jídelníčky.

3.1.1 První rešerše uživatelských požadavků

Jak bylo zmíněno v úvodu, na systému WhereIS se po bakalářské práci pracovalo v týmu. V rámci této práce byla také vypracována rešerše o samotné aplikaci a cílových uživatelích. Je nutno podotknout, že tuto rešerši vypracovali

členové týmu na základě svých subjektivních názorů, nedošlo tedy k vypracování žádného dotazníku, který by získal skutečné uživatelské cíle a use casy. Z toho důvodu vypracoval autor se svým kolegou Bc. Illiou Brylovem v rámci obou prací dotazník, jehož cílem je sběr opravdových uživatelských požadavků na aplikaci WhereIS, které by nejen studenti v systému uvítali.

3.1.2 Sběr požadavků

Tyto požadavky reprezentují use casy, které by uživatelé v aplikaci uvítali a které autoři zanalyzují, a ty, které se vyskytují nejčastěji, nebo autorům dávají smysl, integrují do systému WhereIS.

Dotazník byl rozdělen do tří částí. První část sloužila k získání informací o vyplňující osobě, zejména její vztah ke škole (typ studia, doba působení na škole, jestli učí či nikoliv). Tato část sloužila ke zjištění, jestli dotazník vyplnili studenti a učitelé z různých ročníků. Cílovou skupinou jsou zejména studenti prvního ročníku bakalářského studia, protože se na škole moc dobře neorientují a mají požadavky na jiné, obsáhlejší informace a funkcionality, než starší a zaběhlí studenti či učitelé.

Druhá část dotazníku se zabývala existujícími informačními systémy na FIT ČVUT. Mezi ně patří např. KOS, Courses, UserMap či Agáta. Cílem této části bylo zjistit spokojenost uživatelů s těmito systémy, co je na jejich používání v případě vyhledávání informací frustruje a jaké funkcionality by v systémech uvítali.

Třetí a poslední část obsahovala konkrétní nápady autorů dotazníků na funkcionality, které autory napadly v rámci první rozvahy v 3.1.1. Jejím cílem bylo zjistit, jestli by o ně uživatelé měli zájem. Vyplňující hodnotili tyto funkcionality na stupnici od 1 do 5, kde 1 reprezentovala velký nezájem a 5 velký zájem o integraci do systému.

3.1.3 Analýza získaných požadavků

Jelikož jsou požadavky velmi konkrétní, jsou sloučeny do několika obecnějších požadavků. Požadavky se také rozdělují do dvou kategorií – funkční a nefunkční. U jednotlivých požadavků je uvedena zkratka FP či NP, jestli se jedná o funkční či nefunkční požadavek.

3.1.3.1 Vybrané požadavky k implementaci

Vyhledávání volných místností podle času (FP) Vyhledávání volných počítačových a seminárních místností je v prototypu již implementováno. Nicméně neumožňuje definovat čas, od kdy uživatel chce najít volnou místnost, a dobu, po kterou uživatel chce, aby místnost byla volná. Prototyp hledá volné místnosti v čase vyhledávání (zadání dotazu) a místnost označí jako volnou, pokud je od doby hledání volná alespoň 45 minut. Uživatelé by uvítali rozšíření této funkcionality o právě zmíněný čas, od kdy má být místnost volná, a dobu, na jak dlouho má být volná.

Nadcházející události uživatele (FP) Uživatelé by chtěli v aplikaci mít přístup ke svým nadcházejícím událostem, podobně, jak je tomu v systému *Courses*. Tento přehled událostí by jim nabídnul možnost rychlého vyhledávání

předmětu, učitele či místnosti, které souvisí s některou z nadcházejících událostí uživatele. Díky tomu by pak mohli najít potřebné informace k těmto entitám rychleji.

Porovnávání jídelníčků oblíbených menz (FP) Uživatelé by chtěli mít rychlý přístup k jídelníčkům konkrétních (oblíbených) menz a ty jednoduše porovnávat. Tento požadavek v sobě skrývá několik dalších požadavků:

- vyhledávání menz (FP),
- označení menzy jako oblíbená (FP),
- zobrazení aktuálního jídelníčku menzy (FP).

Vyhledávání jídel (FP) Další požadavek úzce spjatý s menzami a jídelníčky je možnost vyhledávat také samotná jídla, tedy jestli je nějaké jídlo vařeno v následující dny alespoň v jedné menze. Uživatel by tedy například zadal dotaz „řízek“ a systém by mu vrátil veškerá jídla obsahující slovo „řízek“, která by se v následujících dnech v menzách vařila.

Podrobnější informace o lokaci místnosti (FP) Prototyp systému WhereIS nabízí pro každou místnost na FIT ČVUT plánek patra (pokud je dostupný), ve kterém se místnosti nachází. Neobsahuje však žádné jiné doplňující informace, jako je číslo patra nebo adresa budovy. Uživatelé by rádi viděli tyto informace o lokaci místnosti.

Oblíbené entity (FP) Uživatelé by také kromě již zmíněných menz chtěli ukládat do oblíbených i jiné entity. Nejvíce takto požadovanou entitou jsou po menzách předměty. Podle výsledků by uživatelé také rádi uvítali rychlejší přístup k volným místnostem než přes vyhledávací pole.

Perzistentní uživatelské preference (FP/NP) Opakovaným požadavkem uživatelů jsou také perzistentní uživatelské preference. U oblíbených entit je toto samozřejmostí, nicméně tento požadavek se týká také volby jazyka či jiných nastavení vzhledu a fungování aplikace.

Informace o kolejích (FP) Uživatelé by rádi vyhledávali i koleje, aby základní informace o kolejích našli co nejrychleji. Na FIT ČVUT v době psaní této práce neexistuje žádný systém, který by data o kolejích poskytoval v jakémkoliv strojovém formátu. Jediným zdrojem dat o kolejích je web SUZ ČVUT, který je dostupný na `https://www.suz.cvut.cz`. Z těchto důvodů se autor rozhodl zařadit tento požadavek do této práce.

Fotky učitelů/zaměstnanců (FP) Uživatelé by také rádi měli u učitelů/zaměstnanců jejich fotky. Podle odpovědí je to vhodné zejména u učitelů/zaměstnanců s velmi podobnými příjmeními (např. Šolcová vs Scholtzová).

Udržitelnost aplikace (NP) Ve všech odpovědích byl tento požadavek zmíněn pouze jednou, nicméně autorovi práce připadá jako velmi důležitý, a proto se jej rozhodl zařadit do vybraných požadavků. Udržitelnost aplikací, zejména těch, které vznikají jako výsledky bakalářských či diplomových prací, je velmi důležitá. Často tyto aplikace vzniknou a následně na nich nikdo dál nepracuje, nejsou udržované, a tím pádem po určité době nepoužitelné. Autor práce nechce, aby stejný osud potkal aplikaci WhereIS. Proto v rámci této i předešlé práce [1] autor přidává návody, jak aplikaci rozšířit (vizte 4.3.1) či jak pracovat s konkrétními technologiemi, které aplikace používá (např. 4.4.8), a snaží se důkladně popisovat, jak aplikace funguje, aby další lidé mohli na práci jednoduše navázat.

Události na fakultě (FP) Uživatelé mají často problém vyznat se ve všech školních událostech pořádaných školou či studentskými spolky/uniemi, proto by byli rádi za možnost vyhledávat v dostupných událostech na fakultě či studentských událostech.

3.1.3.2 Nevybrané požadavky k implementaci

Přidání různých specifických míst (FP) Uživatelé by rádi vyhledávali také toalety, kuchyňky, respiria či místa k sezení na chodbách. Autor by rád splnil tento požadavek v rámci této práce, nicméně na škole bohužel neexistuje systém, který by poskytoval data o těchto místech, a není v autorových schopnostech zmapovat tato místa sám za rozumný časový úsek. Na FIT ČVUT nově vznikl systém *Navigate*, který některá místa jako toalety či respiria na mapě znázorňuje, nicméně nenabízí žádné API, které by tato data poskytovalo.

Integrace školních systémů ProgTest, Marast, DBS-Portál...(FP) Mezi odpověďmi se často vyskytovaly prosby o integraci velmi používaných školních systémů jako jsou ProgTest, Marast, DBS-Portál a další. Uživatelé by rádi viděli přehled všech nadcházejících úkolů/kvízů/kontrolních bodů s datem posledního možného odevzdání. Ačkoliv by autor tyto systémy velmi rád integroval, jedná se o izolované systémy, které neposkytují data přes žádné API. Aby se tedy uživatel k datům dostal, je nutné, aby se přihlásil pomocí svého ČVUT účtu přímo do konkrétního systému a získal tak přehled o svých předmětech a úkolech. Jakmile budou k dispozici API pro tyto systémy, bude jejich integrace do aplikace WhereIS jednoduchým úkolem.

3.2 Nové zdroje dat

Jak uvedl autor v kapitole *2.2 Zdroje dat* ve své předešlé práci [1], existující prototyp využívá celkem tří zdrojů dat: *Sirius*, *KOSapi* a *Umapi*. Pro splnění požadavků uživatelů získaných v kapitole 3.1.3 je však potřeba tyto zdroje dat rozšířit o nové. Nejdůležitějším novým zdrojem bude *Agáta*, což je webová služba, která zprostředkovává data *Správy účelových zařízení ČVUT* o menzách a jejich jídelníčcích. Následně se bude hodit *Courses API*. Jedná se o tradiční *RESTful* službu, která poskytuje informace o zdrojích z webové aplikace *Courses*. Dalšími novými zdroji budou web FIT ČVUT a web SUZ

ČVUT. Nejedná se o standardní zdroje jako v případě předchozích, protože nemají své API. K získávání dat z těchto zdrojů bude použito *scrapování*.

3.2.1 Agáta

Agáta je služba poskytující informace o menzách ČVUT (název, lokace, otevírací doba…) a jejich jídelníčcích, a to jak aktuálních (snídaně, obědy a večeře v aktuální den), tak i týdenních. Nabízí však i další zdroje, jako například odpovědné osoby pro každou menzu a kontakt na tyto osoby, aktuality menz či doplňující informace k jednotlivým menzám (jejich výdejny, kategorie jídel…). Celková dokumentace API je dostupná na `https://agata-new.suz.cvut.cz/jidelnicky/JAPIV2/JAPI-popis.html`. Všechna data získaná ze zdrojů jsou ve formátu JSON. Na stránkách dokumentace si lze všimnout faktu, že pro používání API Agáty je potřeba *klíč uživatele*. Tento klíč byl autorovi práce přidělen správcem webové služby Agáta. Nejedná se o *RESTful* službu, nepoužívají se zde `HTTP` metody pro práci s daty. Naopak se zde používají čistě query parametry pro volání funkcí nad zdroji a pro předávání parametrů.

Pro práci s API je vhodné znát následující query parametry:

`Funkce` reprezentuje funkci, která bude zavolána,

`api` reprezentuje API klíč uživatele, je součástí každého požadavku (z popisů zdrojů níže je tedy vynechán),

`Podsystem` reprezentuje ID menzy v systému Agáta,

`SecondID` reprezentuje ID týdne v systému Agáta (pro týdenní jídelníčky).

Pro požadavky aplikace WhereIS jsou vhodné následující zdroje:

`GET ?Funkce=GetPodsystemy` vrátí všechny menzy včetně jejich ID pro parametr `Podsystem` používaný v dalších volání,

`GET ?Funkce=GetJidla&Podsystem=menza_id` vrátí jídla, která se aktuálně vaří nebo budou vařit v dané menze,

`GET ?Funkce=GetKategorie&Podsystem=menza_id` vrátí názvy kategorií jídel (mapování ID kategorie a názvu kategorie – polévka, hlavní chod…) v dané menze,

`GET ?Funkce=GetOtDoby&Podsystem=menza_id` vrátí otevírací dobu dané menzy,

`GET ?Funkce=GetAdresy` vrátí adresy všech menz,

`GET ?Funkce=GetTydny&Podsystem=menza_id` vrátí následující týdny a jejich ID pro danou menzu,

`GET ?Funkce=GetTydnyDny&SecondID=tyden_id` vrátí zbývající jídla do konce týdne pro daný týden (nevrátí jídla, která už v týdnu byla vařena).

Data získaná z těchto zdrojů tvoří všechny nutné a zajímavé informace ohledně menz a jídelníčků. Umožní to uživatelům zobrazit jídelníček menzy, její lokaci, otevírací dobu, a navíc díky týdenním jídelníčkům umožní uživatelům hledat nadcházející jídla v menzách.

3.2.2 Courses API

Courses API je API webové aplikace *Courses*, která poskytuje informace o předmětech a výukových materiálech předmětů na FIT ČVUT. Jedná se o REST API, jehož dokumentace je dostupná na `https://courses.fit.cvut.cz/api/v1/doc.html`. Data poskytuje ve formátu JSON. API samo o sobě moc zdrojů nemá, v době psaní této práce jsou tyto zdroje čtyři, nicméně jeden z těchto zdrojů je pro aplikace WhereIS vhodný, a to:

GET `/users/me/courses` což vrátí všechny předměty, které aktuálně přihlášený uživatel studuje či učí.

Tyto předměty pak mohou fungovat jako odkazy rychlého vyhledávání v rámci systému WhereIS, které uživateli mohou přinést další informace, než které by získal v samotné aplikaci *Courses*.

3.2.3 Web FIT ČVUT

Web FIT ČVUT obsahuje spoustu dat a informací o škole, studijních programech, událostech a zaměstnancích, které se nikde v žádné strojově čitelné podobě nenachází. Jak už je z názvu patrné, nejedná se o webovou službu, nýbrž o samotné webové stránky dostupné na `https://fit.cvut.cz/cs`, které nemají žádné dostupné API.

Pro systém WhereIS se budou hodit zejména informace o školních budovách a událostech na fakultě a fotky zaměstnanců FIT ČVUT. Čtenář práce by mohl namítat, že tyto fotky lze získat z webové služby *Umapi*, nicméně fotky poskytované touto službou nejsou aktuální, alespoň ne v době psaní této práce, proto se autor rozhodl získat fotografie z webu FIT ČVUT.

3.2.4 Web SUZ ČVUT

Web SUZ ČVUT obsahuje informace o ubytování, tedy koleje a hotely. Je dostupný na adrese `https://www.suz.cvut.cz/cz`. Podobně jako web FIT ČVUT ani web SUZ ČVUT nemá své vlastní API.

Pro systém WhereIS se zde budou hodit informace o kolejích, které by měly pomoci zejména nastupujícím studentům bakalářského či magisterského studia. K těmto informacím patří např. název koleje, adresa (případně mapa s lokací), kontakt na recepci či odkaz na studentskou samosprávu (klub) nebo na systém *ISKaM*, což je aplikace pro správu ubytování na koleji, kde uživatel může sledovat své peníze spojené s ubytováním (ubytovací jistina, zůstatek na kontě…).

3.3 Nové procesy

Autor ve své předešlé práci [1] vytyčil 3 hlavní procesy původního prototypu WhereIS. Jednalo se o:

- získání a případné zpracování a uložení dat,
- přihlášení uživatele,
- vyhledávání informací a jejich řazení dle relevance.

Z těchto 3 původních procesů zůstává stejné *přihlášení uživatele. Získávání, zpracování a uložení dat* a *vyhledávání informací a jejich následné řazení* se nově o něco málo liší. Díky integraci nových požadavků, které byly získány v 3.1.3, přibyly také nové procesy, a to *označování entit jako oblíbené* a *vyhledávání volných místností*, které v prototypu již funguje a bylo součástí procesu *vyhledávání informací*, ale nyní se z toho stává samostatný proces.

3.3.1 Získávání, zpracování a uložení dat

Tento proces z hlediska uživatele stále nepožaduje žádnou interakci. Nově se však spouští sám každý den o půlnoci. Získává, zpracovává a ukládá mnohem více dat a lze jej spouštět i v případě, kdy aplikace běží. Součástí tohoto procesu je nově také automatická aktualizace aktuálních jídelníčků menz, která probíhá každých 5 minut.

3.3.2 Vyhledávání informací

Tento proces, jak již bylo zmíněno [1], je ten nedůležitější proces ze strany uživatele i aplikace. Uživatel pomocí několika klíčových slov zformuluje dotaz a vyhledá potřebné informace. Nově systém vyhledává krom místností, předmětů, učitelů a volných místností také menzy, jídla, koleje, budovy, katedry a události fakulty. Během formulace dotazu jsou uživateli vráceny výsledky, které jsou seřazeny dle relevance na základě zadaného dotazu. Způsob řazení výsledků je definován interně a uživatel nemá možnost tento způsob v době psaní této práce změnit.

3.3.3 Přidání entit do oblíbených

Z analýzy požadavků uživatelů 3.1.3 vyšlo najevo, že by uživatelé nejraději přidávali menzy a předměty do oblíbených, aby k těmto entitám měli rychlý přístup z domovské stránky aplikace a aby tak jednoduše mohli porovnávat jídelníčky oblíbených menz vedle sebe. Uživatel, který chce přidat konkrétní entitu do oblíbených, musí nejdříve danou entitu v systému vyhledat. Následně otevře detail entity a v tomto detailu klikne na možnost „Přidat do oblíbených“. Tato entita se nyní bude uživateli zobrazovat na domovské stránce.

3.3.4 Vyhledávání volných místností

Jak již autor uvedl, vyhledávání volných místností je již v prototypu implementováno a spadá do procesu vyhledávání informací. Nově však vyhledávání volných místností funguje jiným způsobem. Uživatelé při vyhledávání volných místností mohou nově definovat 2 parametry, a to od kdy chtějí, aby byla místnost volná, a na jak dlouho. Tento proces je oddělený od samotného vyhledávání přes textové pole a je dostupný na domovské stránce aplikace.

3.4 Doménový model

Doménový model, jak uvádí Hartinger [16], je náčrt základních entit systému a vztahů mezi nimi. Je nezávislý na platformě (není určen pro konkrétní programovací jazyk) a atributy nemají datové typy. Jedná se o jeden z typů UML

diagramů tříd, kde, jak už je z názvu patrné, je hlavním stavebním kamenem třída. Tento diagram se tedy využívá zejména u objektového programování.

Doménový model prototypu WhereIS vznikl již v předešlé práci autora [1], nicméně model je v rámci této práce rozšířen o nové entity a aktualizován, a proto je v této kapitole podrobněji popsán.

Celý model se skládá ze 2 částí/pohledů:

semestr – pohled z hlediska aktuálního semestru a objektů souvisejících se semestrem (obrázek 3.1),

ostatní – pohled z hlediska ostatních objektů, které nejsou závislé na semestru (obrázek 3.2).

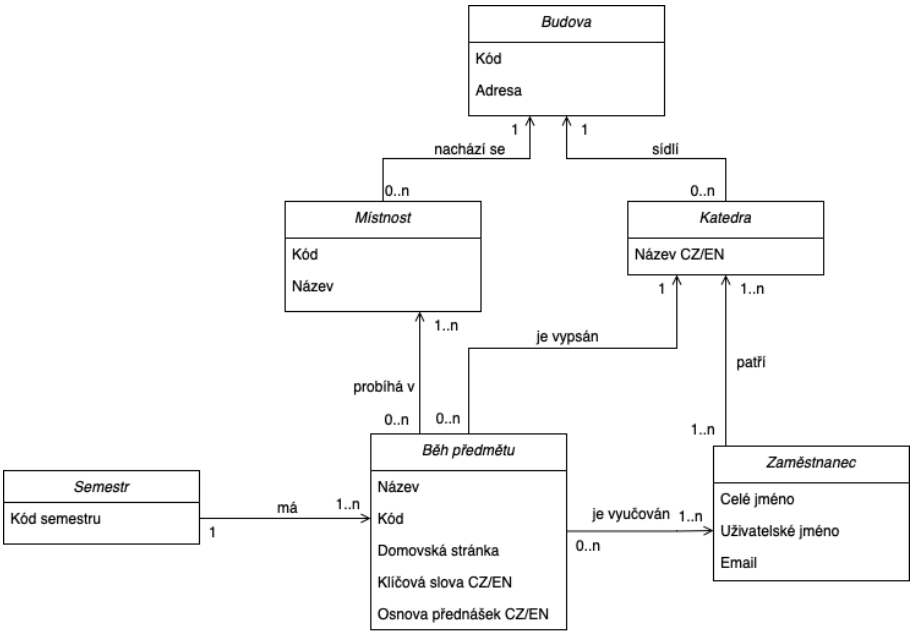
3.4.1 Doménový model semestru

Doménový model semestru obsahuje objekty a vazby, které jsou závislé na aktuálně běžícím semestru. Semestr je cca tří měsíční období výuky na FIT ČVUT a podle části roku, ve kterém běží, se nazývá *zimní* či *letní*. Za každým semestrem následuje zkouškové období, které ale v systému WhereIS není nijak zvlášť znázorněno (lze jej tedy považovat za přímou součást semestru). Jeden akademický rok je tvořen vždy jedním zimním a jedním letním semestrem. Semestr je na FIT ČVUT označován kódem B a třemi číslicemi YY a N, kde YY jsou poslední 2 číslice roku, kdy akademický rok začíná, a N označuje pořadí semestru v akademickém roce. Pro akademický rok 2023/2024 je tedy zimní semestr označen kódem B231 a letní kódem B232.

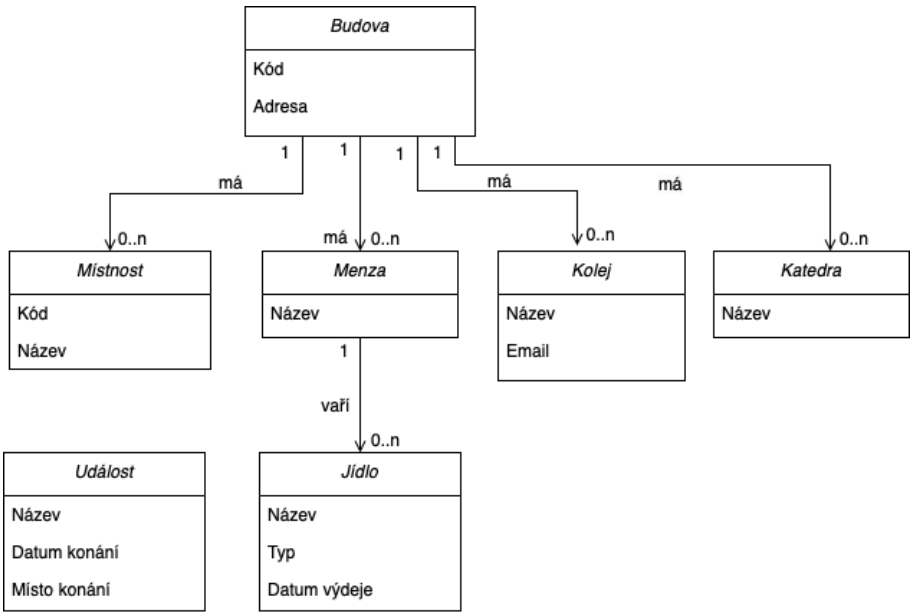
V rámci aktuálního semestru existují běhy předmětů, které se v daný semestr učí. Tyto běhy předmětů učí konkrétní učitelé (zaměstnanci). Tyto předměty jsou vypisovány konkrétními katedrami a zaměstnanci spadají pod katedry. Jeden zaměstnanec může spadat pod více kateder, nicméně předmět může být vypsán pouze jednou katedrou. Předměty jsou v rámci semestru vyučovány podle rozvrhu danými učiteli v daných místnostech na škole. Místnosti a katedry spadají pod konkrétní budovy. Samotný doménový model semestru je možné vidět na obrázku 3.1.

3.4.2 Doménový model ostatních objektů

Tento doménový model obsahuje všechny objekty, které nejsou závislé na semestru. Jedná se hlavně o budovy, místnosti, katedry, menzy, jídlo, koleje a události. Nejdůležitější entitou je v tomto pohledu budova, která má vazbu na menzu, místnost, kolej a katedru. Druhou a poslední vazbou je vazba mezi jídlem a menzou. Tento doménový model je možné vidět na obrázku 3.2.



Obrázek 3.1: Doménový model semestru



Obrázek 3.2: Doménový model ostatních objektů

Kapitola 4

Návrh a implementace

4.1 Technologie

Autor již ve své předešlé práci [1] popsal některé technologie, které byly použity pro vypracování prototypu. Nyní jsou tyto technologie rozšířeny o další potřebné technologie pro splnění uživatelských požadavků a opravy chyb původního prototypu.

4.1.1 Sphinx

Z analyzovaných nástrojů v rešerši 2.1 byl pro integraci do systému WhereIS vybrán Sphinx. Tento nástroj je jednoduše integrovatelný do již existujících aplikací, pracuje přímo nad databází dat, nabízí velmi jednoduché SphinxAPI pro komunikaci s vyhledávacím serverem pro velké množství programovacích jazyků (mimo jiné i pro PHP) a indexy lze jednoduše vytvářet a konfigurovat pomocí jednotného konfiguračního souboru `sphinx.conf`. Sphinx se tedy postará o samotné zpracování textu v databázi (odstranění stop slov, stematizace, odstranění interpunkce...), provede vyhledávání nad jednotlivými indexy a získané relevantní výsledky seřadí podle váhy na základě zvolené rankovací funkce. Autor také uvažoval nad vlastní implementací vyhledávací logiky (zpracování textu, indexace, ranking...), nicméně Sphinx se ukázal jako vhodný vyhledávací nástroj pro účely systému WhereIS a jeho integrace ušetří velké množství práce.

4.1.1.1 Způsob vyhledávání relevantních výsledků

V době psaní této práce nabízí Sphinx 2 základní typy indexů, které se pro full-textové vyhledávání dají použít:

batchové – jedná se o offline typ indexu, který zpracovává data pouze při zavolání indexeru a samotný index je ukládán do souborů na disku,

real time – jedná se o online typ indexu, který zaktualizuje index automaticky při jakékoliv modifikaci dat.

Pro systém WhereIS je nejvíce vhodný real time index, protože se data v databázi aktualizují každý den. Bohužel v době psaní této práce neumožňuje aktuální verze Sphinx používat real time indexy přes SphinxAPI, ale pouze

přes SphinxQL, pomocí kterého se také musí data do indexu vkládat. Real time index se tedy nevytváří z dat uložených v databázi, ale data v real time indexu se musí aktualizovat manuálně právě pomocí SphinxQL. Je tedy nutné použít batchové indexy a zautomatizovat jejich aktualizaci. O vytváření a aktualizaci batchových indexů se ve Sphinx stará program s názvem `indexer`.

O samotné vyhledávání relevantních výsledků se následně stará démon s názvem `searchd`, který přijímá dotazy přes SphinxAPI, SphinxSE a SphinxQL. Dotaz je při vyhledávání v indexech podroben stejné normalizaci, která byla použita u jednotlivých indexů, ve kterých démon vyhledává. Výsledky jsou následně vráceny ze všech vyhledávaných indexů seřazené podle zvolené rankovací funkce od nejvíce relevantních po nejméně relevantní.

4.1.1.2 Počítání váhy (ranku) výsledku

Sphinx nabízí několik již předpřipravených rankovacích funkcí, které se dají ihned využít. Je také možné si pomocí různých metrik a statistik, které Sphinx udržuje a vypočítává, vytvořit svou vlastní rankovací funkci, pokud by předpřipravené funkce nestačily. V době psaní této práce nabízí Sphinx podle [15] tyto rankovací funkce:

`PROXIMITY_BM25` – kombinace BM25 a proximity vyhledávání (jestli jsou vyhledávané termy v určité vzdálenosti od sebe),

`BM25` – pouze BM25,

`NONE` – žádné rankování,

`WORDCOUNT` – počítá výskyt klíčových slov,

`PROXIMITY` – vrací vzdálenost hledané fráze ve výsledku,

`FIELDMASK` – vrací 32 bitovou masku, kde N-tý bit koresponduje s N-tým klíčovým slovem,

`SPH04` – to samé jako `PROXIMITY_BM25`, ale dává větší váhu výsledkům, které se objevují na úplném začátku či úplném konci textového pole,

`EXPR` – umožňuje definovat vlastní rankovací funkci za běhu.

Pro systém WhereIS se z této nabídky rankovacích funkcí dá zvolit několik možností. Autorovi práce se nejvíce líbí použití `WORDCOUNT` funkce.

4.1.2 Security

Pro vyřešení bezpečnosti a správné autentizace a autorizace uživatelů se autor rozhodl využít balíčků *Symfony Security Bundle*, který je součástí samotného frameworku Symfony a nabízí kvalitně zpracovanou dokumentaci, a *OAuth2 Client Bundle*, který usnadňuje práci s OAuth 2.0 autorizačním protokolem.

4.1.2.1 Symfony Security Bundle

Podle dokumentace [17] obsahuje *Symfony Security Bundle* spoustu nástrojů a všechny autentizační a autorizační funkce potřebné pro zabezpečení aplikace. Obsahuje 4 základní elementy, které je potřeba pochopit.

User (uživatel) Objekt `User` představuje samotného uživatele a jsou s ním spojená veškerá jeho práva. Uživatele lze jednoduše vytvořit pomocí konzolového příkazu `php bin/console make:user`, který spustí interaktivní proces jeho vytváření. Lze tak definovat, co se k danému uživateli ukládá. Pro systém WhereIS se hodí:

username uživatelské jméno ČVUT,

accessToken přístupový token uživatele získaný po úspěšném přihlášení přes OAuth 2.0 autorizační server,

refreshToken obnovovací token uživatele pro získání nového přístupového tokenu v případě jeho vypršení,

roles pole rolí uživatele (obsahuje pouze roli „ROLE_USER“ pro WhereIS).

Výše zmíněné údaje o uživateli se ukládají do databáze. Příkaz na vytváření uživatele zaregistruje také podle [17] *user providera*, který se stará o načítání uživatele z databáze podle jeho uživatelského jména a také se stará o opětovné načtení uživatele ze session. Jelikož aplikace WhereIS neřeší samotné přihlašování, není nutné řešit ukládání a hešování hesel uživatelů. Nicméně i tak je stále důležité myslet na další prvky zabezpečení, protože se do databáze ukládají refresh tokeny (obnovovací tokeny).

Firewall Firewall v tomto slova smyslu znamená dle [17] autentizační systém[2]. Definuje, které části aplikace jsou zabezpečené a jakým způsobem se uživatelé budou autentizovat. Lze jej nakonfigurovat v souboru `security.yaml` ve složce *config/packages*. Pro každý požadavek je vždy aktivní pouze jeden firewall. Pro určení, který firewall se má použít, se v konfiguračním souboru používá klíčové slovo `pattern`, který definuje vzor URL odkazů, které daný firewall má obsluhovat. Je důležité zmínit, že všechny stránky, které vyžadují přihlášeného uživatele, musí být pod stejným firewallem (nejčastěji je to firewall `main`).

Autentizace uživatele Autentizace uživatele je proces, při kterém dochází k ověření uživatele (tedy namapování nepřihlášeného uživatele na známého, zaregistrovaného uživatele). *Symfony Security* podle [17] nabízí několik možností, jak autentizovat uživatele, např. pomocí formuláře, HTTP Basic, JSON přihlášení (přes API), přístupový token...Pro systém WhereIS, který využívá autorizační server OAuth 2.0, je nutné použít možnost přístupového tokenu.

Autorizace uživatele Po autentizaci uživatele nastává jeho autorizace, což znamená ověření práv (jestli má uživatel možnost něco vidět, editovat, mazat...). Podle [17] je k úspěšné autorizaci uživatelů potřeba udělat 2 kroky:

1. Přiřazení určitých rolí uživateli (pole `roles` v entitě `User`).
2. Přidání vyžadovaných rolí na příslušná místa do kódu. Tedy uživatel musí mít přiřazenou danou roli, aby se daný kus kódu provedl.

Na základě rolí je možné také omezit v `security.yaml` přístup k určitým URL adresám. Pro zabezpečení `Controller` tříd lze využít anotaci `#[IsGranted('ROLE_NAME')]`, která se píše nad definicí třídy.

[2] Nejedná se o stejný firewall jako v počítačových sítích.

4.1.2.2 OAuth2 Client Bundle

Tento balíček jde ruku v ruce se *Symfony Security* a integruje autentizaci uživatele přes protokol OAuth 2.0. Balíček je dostupný na `https://github.com/knpuniversity/oauth2-client-bundle` a umožňuje nakonfigurovat svého OAuth 2.0 klienta. Nabízí velké množství předem připravených klientů (`Provider`). Jelikož mezi těmito klienty není *ZuulOAAS*, je nutné takového klienta vytvořit manuálně.

4.1.3 Scrapování

K integraci webových stránek je použito *scrapování*. Web scraping (scrapování), jak uvádí Thér [18], je soubor rozličných technik, které slouží k získávání dat z internetových stránek za účelem jejich dalšího zpracování. Nejčastějším typem zpracování je převedení získaných dat z nestrukturovaného formátu (nejčastěji čistý text) do strukturovaného, strojem čitelného formátu. Pro scrapování se používá robot scraper, který za pomoci selektorů získá data z webových stránek. Scrapování jde ruku v ruce s jiným procesem, a to *crawlingem*. V některých publikacích, jako je tomu například v článku Khdera [19], znamenají tyto dva pojmy totéž, a to „extrakce dat z webu s použitím softwaru“. Autor této práce však považuje tyto pojmy za odlišné:

scrapování znamená extrakci dat z webu za použití softwaru (scraper),

crawling znamená procházení webu za pomocí softwaru (crawler), konkrétně procházení webových stránek a objevování nových URL.

Scrapování, jak uvádí Khder [19], se skládá ze 3 částí:

fetching – získání HTML dokumentu pomocí různých nástrojů jako je `curl` nebo `wget`,

extrakce – získání dat z HTML dokumentu za pomocí regulárních výrazů, XPath či CSS selektorů,

transformace – transformace extrahovaných dat do strukturovaného, strojově čitelného formátu.

Symfony nabízí balíček pro crawlování a scrapování webu s názvem *Symfony BrowserKit*. Právě tento balíček autor volí k integraci webových stránek do systému WhereIS. Hlavní důvod volby tohoto balíčku je fakt, že je součástí frameworku Symfony.

4.1.4 Secrets management

Pro správu citlivých údajů se autor rozhodl využít balíček *Symfony's Secrets Management System*, který je již součástí frameworku Symfony. Jak je uvedeno v dokumentaci Symfony [20], využívá tento systém (někdy se také označuje jako „vault“) asymetrického šifrování. Je tedy potřeba vygenerovat *veřejný* a *privátní* klíč. Veřejný klíč slouží k zašifrování a přidání nové citlivé informace do „trezoru“ a je bezpečné jej verzovat. Privátní klíč na druhou stranu slouží k dešifrování citlivých informací a neměl by se v případě produkce nikdy zveřejňovat. Tento balíček podporuje společně se samotným frameworkem

```
*   *   *   *   *  sh /path/to/script/script.sh
|   |   |   |   |              |
|   |   |   |   |      Command or Script to Execute
|   |   |   | Day of the Week(0-6)
|   |   | Month of the Year(1-12)
|   | Day of the Month(1-31)
| Hour(0-23)
Min(0-59)
```

Výpis kódu 4.1: Syntaxe práce pro Cron, převzato z [23]

možnost definovat různá prostředí, a tedy i různé citlivé údaje pro vývoj, testování a produkci.

4.1.5 Databázové transakce

Databázové transakce, jak uvádí Zanini [21], jsou součástí všech populárních relačních databází, jako jsou Oracle, PostgreSQL, SQLite a další. Jedná se o logický celek SQL operací, které se provádějí jedna po druhé. Pokud jedna z těchto operací selže, selže celá transakce a žádné změny se neprovedou – *rollback*. V opačném případě jsou provedeny všechny změny transakce – *commit*. Tran- sakce jsou tak výborným nástrojem pro úpravu dat v databázi (odstranění, přidání, změnění entit) za běhu aplikace, která databázi využívá.

4.1.6 Plánovač práce (Job Scheduler)

Jak uvádí Awati [22], job scheduler je počítačový program, který umožňuje plánovat práci. Stará se zejména o to, aby se daná práce spustila automaticky v konkrétní čas, a také umožňuje monitorovat samotný průběh dané práce. Takovýchto plánovačů existuje v době psaní této práce velké množství, dokonce jsou již součástí UNIXových operačních systémů (Linux, MacOS) a operačního systému Windows. Pro Windows existuje *Windows Task Scheduler*[3] a v UNIXových operačních systémech existuje řešení s názvem *Cron*. Framework Symfony také nabízí své řešení pro plánování uvnitř samotné aplikace s názvem *Scheduler*.

Cron Cron je podle Hiry [23] plánovač práce dostupný v UNIXových operačních systémech. Jedná se o démona (program běžící neustále na pozadí). Cron čte z *cron tabulky*, ve které jsou specifikované skripty/programy, které a kdy má spouštět. Pro automatické spouštění scriptu v určitou dobu je nutné přidat tento script společně s konfigurací automatického spouštění do této tabulky. Konkrétní syntaxe vytváření práce pro Cron je vidět ve výpisu 4.1.

Symfony Scheduler Symfony Scheduler, jak je uvedeno v dokumentaci [24], je komponenta, která se stará o plánování tasků (úkolů) pro PHP aplikace. Mezi tyto úkoly můžou patřit například vyčištění cache, vyčištění databáze nebo procesy běžící na pozadí aplikace jako synchronizace dat nebo periodická

[3] Nejedná se o Task Manager.

```
public function getFreeRoomsByTime(int $minutesRequired = ...,
    DateTimeInterface $desiredDateTime = ...): array
```

Výpis kódu 4.2: Hlavička metody `getFreeRoomsByTime` třídy `FreeRoomItem`

aktualizace dat. Největší výhodou používání této komponenty je to, že o automatizaci se stará přímo samotná PHP aplikace, nikoliv jiný program/démon třetí strany. Tento balíček funguje na základě zasílání zpráv, které jsou reprezentované jako objekty. Tyto zprávy podle definovaného rozvrhu rozesílá `Scheduler` třída. Zprávy jsou následně zachyceny programátorem definovanou `Handler` třídou, která provede požadovaný kus kódu.

4.2 Integrace požadavků uživatelů

V této kapitole autor navrhne způsob integrace uživatelských požadavků vymezených v 3.1.3.1. Některé požadavky autor pokryje v rámci jedné podkapitoly, protože návrh na jejich řešení je velmi podobný či úplně stejný. Nefunkční požadavek „Udržitelnost aplikace“ autor v této kapitole již nerozebírá, jelikož se jedná o obecný požadavek na samotnou aplikaci a její budoucnost a ne na její fungování a rozšíření funkcionalit.

4.2.1 Vyhledávání volných místností podle času

Vyhledávání volných místností, jak již autor několikrát uvedl, je již v původním prototypu implementováno. Rozšíření o vyhledávání podle času, kdy má být místnost volná, a doby, na jak dlouho má být volná, je přidáním 2 parametrů do metody, která se o toto vyhledávání stará. Hlavičku takové metody je možné vidět ve výpisu 4.2.

4.2.2 Nadcházející události uživatele

Nadcházející události uživatele lze jednoduše získat z webové služby *Sirius*. Pro přístup k těmto událostem je však potřeba mít uložený přístupový token přihlášeného uživatele, který byl vygenerován a vrácen společně s obnovovacím tokenem školním autorizačním serverem *ZuulOAAS*. Tyto údaje je vhodné ukládat společně s přihlašovacím jménem uživatele do databáze (jak bylo zmíněno v 4.1.2.1). Po vypršení přístupového tokenu lze pak jednoduše zažádat o nový díky obnovovacímu tokenu. Ze získaných událostí ze Siria je vybráno 8 nejbližších událostí, které jsou uživateli vráceny.

4.2.3 Porovnávání jídelníčků oblíbených menz

Jak již bylo zmíněno v analýze 3.1.3.1, sestává tento požadavek z několika dalších požadavků. Nejprve je nutné do systému WhereIS integrovat novou entitu – menza (canteen). K tomu je potřeba integrovat také novou webovou službu, a to Agáta (vizte 3.2). Statická data o menzách lze získávat, zpracovávat a ukládat v rámci příkazu `fetch`, proto je nutné implementovat třídu `CanteenFetcher` implementující rozhraní `ItemFetcher`. Po přidání menz do systému je nutné implementovat další požadavek, a to možnost označit menzu

jako oblíbenou. K tomu je potřeba si pro každého uživatele v databázi pamatovat, které menzy označil jako oblíbené (vizte požadavek 4.2.6). Nakonec je potřeba získat aktuální jídelníčky pro oblíbené menzy ze systému Agáta. Zobrazování jídelníčků pro jednoduché porovnání je již frontendovou záležitostí. Integrace nové entity a `Fetcher` třídy je popsána v návodu 4.3.1.

4.2.4 Vyhledávání jídel

Ke splnění tohoto požadavku lze opět využít webovou službu Agáta, která nabízí možnost získat jídelníčky pro zadanou menzu na aktuální a následující týden. Tato data je nutné mít uložena v databázi, proto je potřeba vytvořit novou entitu – jídlo (food). Jednotlivá jídla získána ze zmíněného zdroje obsahují informace jako název jídla, hmotnost, menzu (ve které je jídlo vařeno) či datum, kdy má být jídlo vařeno. Je důležité upozornit na fakt, že tyto týdenní jídelníčky se mohou měnit, proto je nutné začlenit získávání týdenních jídelníčků do příkazu `fetch`. Celkový proces integrace této nové entity je velmi podobný integraci menz, jejíž průběh je popsán v návodu 4.3.1.

4.2.5 Podrobnější informace o lokaci místnosti

K implementaci tohoto požadavku je nutné získat některé informace z internetu, protože v době psaní této práce neexistuje webová služba, která by tyto informace o místnostech poskytovala. Adresy budov patřící k FIT ČVUT lze získat z internetu a uložit do kódu či databáze. K adresám lze také přidat odkaz na *Google Maps*, který vede na mapu s lokací dané budovy, kde se místnost nachází. Patro místnosti lze získat z kódu místnosti podle určitých pravidel (pro „T9“: lze vzít první číslici za „:“, pro TH:A- záleží na délce řetězce za „-“ a podle délky vzít první či první dvě číslice…). Kódové značení místností je podrobně popsáno v `https://help.fit.cvut.cz/rooms/index.html`.

4.2.6 Oblíbené entity a perzistentní uživatelské preference

Uživatelské preference lze ukládat různými způsoby. Napříč webovými aplikacemi jsou často používány soubory *cookies* nebo databáze. Soubory cookies nejsou v tomto případě dobrou volbou, protože pokud se uživatel přihlásí z jiného zařízení, nebudou načtena žádná data, protože soubory cookies na tomto zařízení chybí. Proto se autor rozhodl veškeré tyto informace ukládat do databáze, konkrétně do tabulky `user_preferences`, ve které jako primární klíč slouží uživatelské jméno, které je napříč ČVUT vždy unikátní. Do této tabulky lze ukládat vše, co si uživatel nastaví, např. oblíbené entity, jazyk či různá nastavení vzhledu aplikace. Při přihlášení uživatele do systému jsou následně veškerá tato data z databáze získána a vrácena frontedové části, která příslušná nastavení aplikuje.

4.2.7 Informace o kolejích

Ke splnění tohoto požadavku je nutné vytvořit novou entitu – kolej (dormitory). Poté je potřeba získat data o jednotlivých kolejích. K tomu lze využít scrapování (vizte 4.1.3), pomocí kterého lze získat seznam všech kolejích zmíněných na stránkách SUZ ČVUT a z jednotlivých stránek kolejí lze pak extrahovat ty

nejdůležitější informace (název, kontaktní informace, URL adresu studentského klubu, URL adresu koleje na stránkách SUZ ČVUT...). Veškerá tato data lze získávat, zpracovávat a ukládat v rámci běhu `fetch` příkazu, proto je nutné vytvořit také třídu `DormitoryFetcher` implementující rozhraní `ItemFetcher`.

4.2.8 Fotky učitelů/zaměstnanců

Ke splnění tohoto požadavku je potřeba rozšířit entitu `Employee` o novou členskou proměnnou, a to `photoUrl`. Systém neukládá fotky z webu FIT ČVUT přímo, ale ukládá si do databáze pouze URL adresu. K získání těchto adres je opět použito scrapování, ke kterému dochází vždy při spuštění příkazu `fetch` v rámci získávání dat v třídě `TeacherEmployeeFetcher`.

4.2.9 Události na fakultě

Ke splnění tohoto požadavku je potřeba vytvořit novou entitu – událost (event). Události na fakultě lze získávat z různých webů, např. z webu FIT ČVUT, z webu Studentské Unie ČVUT nebo z webu ČVUT. Poslední dva vyjmenované weby slouží pro celouniverzitní události. Autor se v rámci tohoto požadavku zaměří pouze na fakultní události, proto získává data o událostech z webu FIT ČVUT. Jelikož události jsou statická data, je jejich získávání zařazeno do příkazu `fetch`. Celkový proces integrace této nové entity je velmi podobný integraci menz, jejíž průběh je popsán v návodu 4.3.1. Jelikož se autor práce snaží cílit na jednoduchou rozšiřitelnost aplikace, rozhodl se pro tento požadavek implementovat nové rozhraní `EventFetcherInterface`, které slouží pro jednoduchou rozšiřitelnost při přidávání nových stránek s událostmi fakulty/školy. Pro přidání nové stránky/zdroje stačí pouze vytvořit novou třídu implementující výše zmíněné rozhraní, které v rámci metody `fetchEvents` vrací pole událostí. O vše ostatní už se postará konfigurace zmíněného rozhraní a třída `EventFetcher`, která ve své metodě `fetch` zavolá metodu `fetchEvents` na všech třídách implementující rozhraní `EventFetcherInterface`.

4.3 Integrace nových zdrojů dat

V rámci kapitoly 3.2 autor analyzoval a vyjmenoval nové zdroje dat, které je nutné integrovat do aplikace WhereIS. Integrace takovýchto zdrojů dat je důležitou součástí rozšiřování aplikace, proto se autor rozhodl sepsat návod, jak taková integrace nového zdroje dat a entity do systému WhereIS probíhá. Tento návod také pomůže dalším vývojářům k navázání na autorovu práci a k lepšímu pochopení fungování aplikace. Autor z vyjmenovaných systémů považuje za nejdůležitější webovou službu *Agáta*, proto popisuje návod kroky integrace právě tohoto systému.

4.3.1 Návod na integraci nového systému – Agáta

Návod na integraci nového systému se může lišit počtem kroků ve chvíli, kdy je potřeba data ze zdroje webové služby nejdříve získat, zpracovat a následně uložit do databáze. To je důležité hlavně z těchto důvodů:

- aby systém WhereIS nezahltil webové služby opakujícími se požadavky na stejná data (k tomu se dají využít i *cache*),
- aby data byla dostupná i v případě výpadku webových služeb,
- aby data byla připravena v požadované reprezentaci,
- aby se dala data indexovat a vyhledávat pomocí vyhledávacího nástroje Sphinx.

Webová služba Agáta obsahuje informace o menzách, které jsou ideálním příkladem dat pro uložení do databáze a dat, které by uživatel chtěl v rámci systému vyhledávat. Z toho důvodu autor popíše kompletní návod takové integrace, který obsahuje navíc vytvoření nové entity (menzy) a rozšíření příkazu `fetch`.

1. Vytvořte novou entitu `Canteen`. Pomocí konzolového příkazu `php bin/console make:entity` spustíte interaktivního průvodce vytváření entity společně s různými nápovědami. Jak je navíc uvedeno v dokumentaci [25], vygeneruje tento příkaz třídu samotné entity `Canteen` i `Repository` třídu `CanteenRepository`.

2. Vytvořte migraci pomocí konzolového příkazu `php bin/console make:migration`. Migrace jsou `PHP` soubory pro *Doctrine* obsahující `SQL` příkazy pro provedení nových databázových změn (a pro vrácení nově provedených změn).

3. Proveďte migrace pomocí konzolového příkazu `php bin/console doctrine:migrations:migrate`.

4. Vytvořte třídu `AgataResource` ve jmenném prostoru `App\Service\Resource` (složka *src/Service/Resource*). Třídy `Resource` obsahují metody pro komunikaci s webovou službou a získávání dat z těchto služeb. Pro třídu `AgataResource` to budou např. metody `fetchCanteens`, `getMenu`, `getOpeningHours`...

   - Tyto třídy využívají také `Parser` tříd pro parsování získaných dat a převedení dat do objektů. Je tedy vhodné vytvořit třídu `AgataParser` ve jmenném prostoru `App\Parser` (složka *src/Parser*).
   - Není vždy nutné vytvářet novou `Resource` třídu. Pokud se jedná o nový systém s API, je vhodné novou třídu vytvořit. Pokud se jedná o webovou stránku či přidání nového zdroje z již integrovaného systému, lze využít existujících tříd.

5. Vytvořte třídu `CanteenFetcher` implementující rozhraní `ItemFetcher` ve jmenném prostoru `App\Fetch\Fetcher` (složka *src/Fetch/Fetcher*). Třídy `Fetcher` slouží k získání, zpracování a uložení dat do databáze. Rozhraní `ItemFetcher` vyžaduje implementaci metody:

   `fetch()` získá data z webové služby a uloží je do databáze (využívá implementovaných `Resource` tříd).

   Je nutno uvést několik poznámek k těmto třídám:

- Jednotlivé `Fetcher` třídy primárně aktualizují[4] data v databázi. Pokud získaná data v databázi nejsou, jsou do ní uložená. Pokud se naopak některá data nezaktualizovala přes rok, jsou z databáze odebrána (proměnná `lastUpdate` u většiny entit). Mezi entity, které tuto proměnnou nepoužívají, patří zejména jídlo. Záznamy jídel jsou vždy při spuštění příkazu `fetch` smazány. Smažou se tím tak jídla, která již v týdnu byla vařena, a jelikož se týdenní jídelníčky mohou kdykoliv změnit, budou tak vždy všechna jídla v databázi aktuální.
- `Fetcher` třídy získávají i jiná, přidružená a potřebná data než jen ta z jejich názvu. Např. třída `CanteenFetcher` kromě menz získává také všechna jídla jednotlivých menz.
- Je nutné, aby `Fetcher` třída implementovala rozhraní `ItemFetcher`. Jen tak je automaticky zaregistrována do třídy `FetchManager`. Volání metody `fetch` u jednotlivých těchto třídách dochází v abecedním pořadí. Na to je potřeba myslet, pokud některá `Fetcher` třída závisí na datech získaných z jiné `Fetcher` třídy.

6. Vytvořte třídu `CanteenItem` implementující rozhraní `Item` ve jmenném prostoru `App\Service\Item` (složka *src/Service/Item*). Tato třída reprezentuje *service* třídu pro danou entitu. Obsahuje v sobě metody pro hledání a vracení dat. Nejdůležitější jsou metody rozhraní:

   `findPreview(QueryString $queryString)` reprezentuje metodu, která na základě zadaného vyhledávacího řetězce `queryString` najde[5] relevantní výsledky a tyto výsledky vrátí v *preview*[6] podobě – volá se během vyhledávání třídou `SearchService`,

   `getDetail(array $conditions)` vrátí všechny informace o dané entitě na základě podmínek `conditions`, tyto podmínky definují způsob nalezení konkrétní entity v databázi (např. `['id' => $id]` – volá se při kliknutí na *preview* výsledku.

7. Vytvořte třídu `CanteenView` dědící ze třídy `AbstractItemView` ve jmenném prostoru `App\View` (složka *src/View*), která implementuje rozhraní `ItemView`. Tato třída obsahuje pouze data pro zobrazení *preview* vráceného výsledku metodou `findPreview` v `CanteenItem`. Třída `CanteenView` musí implementovat metody:

   `getTitle()` vrátí titul (nadpis) daného výsledku, který je zobrazen u výsledku na frontendu,

   `getRank(QueryString $query)` vypočítá rank (váhu) výsledku na základě zadaného dotazu (často vrátí pouze rank získaný ze Sphinx).

   Jedná se o `readonly` třídy, což znamená, že hodnoty lze členským proměnným přiřazovat pouze v konstruktoru a poté je již nelze měnit. Není tedy potřeba implementovat gettery a settery a jednotlivé členské proměnné lze označit jako `public`.

[4] Tedy nesmažou všechna data a nevkládají nová. To dává smysl jen u některých entit jako události či jídla.

[5] Pomocí vyhledávacího nástroje Sphinx.

[6] Neobsahuje všechna data, nýbrž jen nutná data pro korektní identifikaci výsledku.

8. Vytvořte třídu `CanteenDetail` dědící ze třídy `ItemDetail` ve jmenném prostoru `App\Detail` (složka *src/Detail*), která reprezentuje obálku všech dat vrácených v detailu výsledku. Tato třída je vrácena metodou `getDetail()` v `CanteenItem`. Jedná se opět o `readonly` třídy.

9. Vytvořte REST endpoint pro získání detailu menzy ve třídě `RestDetailController` ve jmenném prostoru `App\Controller\REST` (složka *src/Controller/REST*).

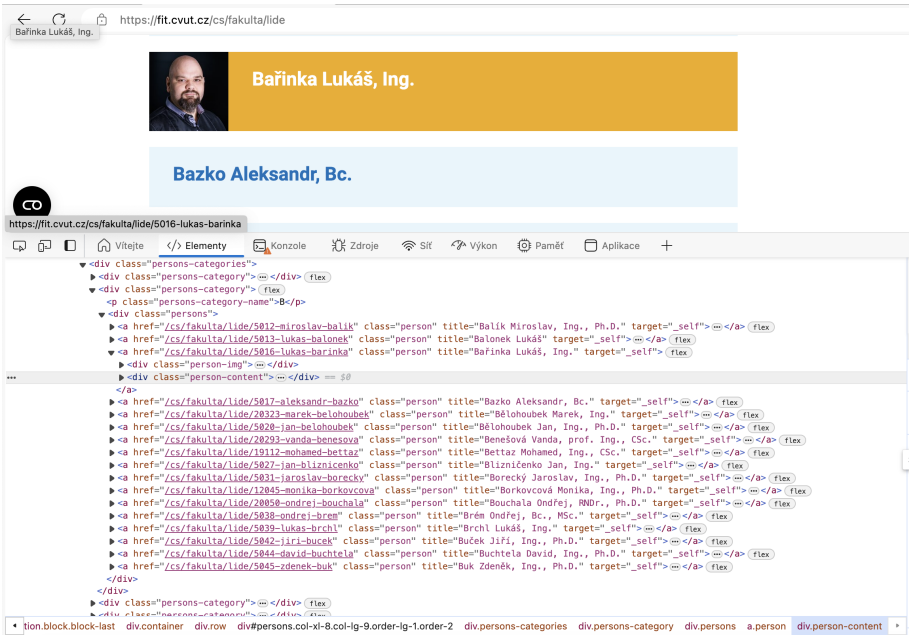
10. Jako poslední krok je nutné vytvořit index pro menzu ve vyhledávacím nástroji Sphinx. To je potřeba udělat v konfiguračním souboru `sphinx.conf` ve složce *sphinx* stejným způsobem jak pro ostatní entity. Podrobnější informace o konfiguraci Sphinx indexů vizte kapitolu 4.5.

4.3.2 Integrace webových stránek

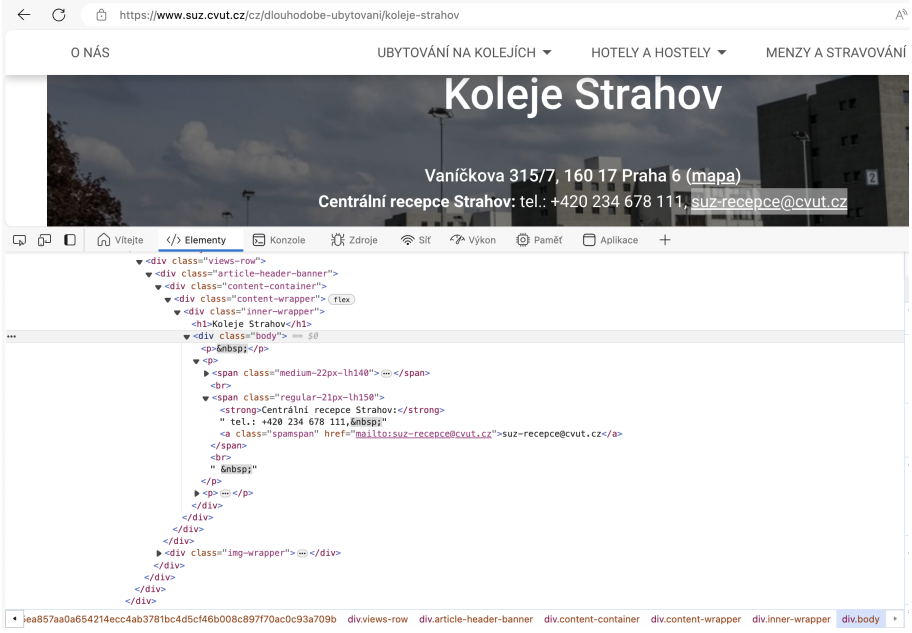
V kapitole 4.3.1 autor popsal kompletní návod na integraci nového zdroje dat do systému WhereIS a vytvoření nové entity. Jelikož v rámci této práce dojde k integraci nového typu zdroje dat, a to webových stránek, popíše krátce autor i způsob jejich integrace pomocí scrapování (vizte 4.1.3) a možné problémy, na které lze narazit. Mezi tyto problémy patří zejména struktura webových stránek, ze kterých jsou informace scrapovány. Od struktury stránek se odvíjí složitost samotné extrakce dat.

Web FIT ČVUT V případě webu FIT ČVUT je extrakce dat díky dobré struktuře stránek jednoduchá (vizte obrázek 4.1). Pro extrakci dat z webu a tedy i stránek FIT ČVUT vytvořil autor novou třídu `WebResource`. Dále byla vytvořena třída `WebParser`, která se stará o parsování scraperem získaných dat do požadovaných objektů. Jelikož se v době psaní této práce využívá tento web na získání informací o budovách, fotkách zaměstnanců a o událostech na fakultě, používají tuto třídu zejména `Fetcher` třídy, které během příkazu `fetch` vytvoří/zaktualizují entity budov a událostí a přiřadí URI fotek příslušným zaměstnancům.

Web SUZ ČVUT V případě webových stránek SUZ ČVUT už situace tak jednoduchá není, naopak by autor podotkl, že se jedná o těžký úkol v případě extrakce některých dat. Jak je vidět na obrázku 4.2, struktura webových stránek jednotlivých kolejí na webu SUZ ČVUT není zdaleka ideální. Pro odřádkování se používá element pro odstavec `<p>`, nepoužívají se jako hodnoty atributů `id` a `class` názvy bloků, ale názvy definující styl bloku (a těch je na stránce více) a v neposlední řadě je hlavní e-mailová adresa koleje obalena třídou `spam-span`. Podle webové stránky [26] je `SpamSpan` malý kus JavaScriptového kódu, který umožňuje zakrýt e-mailovou adresu a zabránit tak spambotům v jejím shromažďování. Nicméně autorovi práce se tento „filtr“ podařilo obejít a e-mailovou adresu získat. Stejně jako metody pro extrakci dat z webu FIT ČVUT i metody pro práci s webem SUZ ČVUT byly přidány do tříd `WebResource` a `WebParser`.



Obrázek 4.1: Struktura webové stránky „Lidé“ webu FIT ČVUT



Obrázek 4.2: Struktura webové stránky „Koleje Strahov“ webu SUZ ČVUT

4.4 Řešení problémů prototypu

V rámci kapitoly 2.2 autor vyjmenoval a analyzoval nalezené problémy a nedostatky původního prototypu systému WhereIS. V rámci této kapitoly autor navrhne řešení ke každému nalezenému problému či nedostatku.

4.4.1 Autorizace uživatelů a špatné oddělení scopes v URL

Jelikož jsou tyto problémy součástí většího problému, a to špatně implementovaného přihlašování uživatelů, pokryje autor tyto problémy naráz. Navíc, jak bylo řečeno v úvodu této práce, implementace tohoto řešení byla provedena již dříve v rámci týmové práce za pomoci kolegy Bc. Jakuba Jabůrka. Autor této práce tedy popíše fungování použitého řešení.

Jak bylo řečeno v analýze těchto problémů v kapitolách 2.2.1 a 2.2.2, v původním prototypu je přihlašování uživatele a vytváření session prováděno manuálně, kde dochází ke kontrole query parametru `state` pro kontrolu úspěšného přihlášení, a autorizační odkaz je generován z předem připravené řetězcové šablony. Pro řešení těchto problémů jsou použity balíčky *Symfony Security Bundle* a *OAuth2 Client Bundle* popsané v 4.1.2.

Nejprve je potřeba vytvořit několik tříd:

`User a UserProvider` třídy popsané v kapitole 4.1.2.1,

`Zuul` třída implementující *ZuulOAAS* klienta pro balíček popsaný v 4.1.2.2, stará se o vygenerování autorizačního odkazu s příslušnými scopes, přesměrování na konkrétní URL *ZuulOAAS* serveru a vrátí „uživatele“ (třída `ZuulUser`) pro obdržený přístupový token,

`ZuulUser` třída obsahující *resource owner* informace, v tomto případě pouze přihlašovací jméno ČVUT,

`ZuulAuthenticator` třída zodpovědná za extrakci údajů[7] z HTTP požadavku a za řízení přihlašování pomocí metod:

- `supports()` slouží k rozhodnutí[8], jestli má dojít k extrakci údajů z URL (`true` = ano), je volána přímo Symfony frameworkem před zpracováním každého požadavku,
- `authenticate()` volána, pokud `supports()` vrátí `true`, používá `Zuul` klienta k získání přístupového tokenu na základě získaného autorizačního kódu,
- `onAuthenticationSuccess()` určuje, co se má stát po úspěšné autentizaci, pokud vrátí `null`, požadavek je předán controlleru,
- `onAuthenticationFailure()` určuje, co se má stát po neúspěšné autentizaci (např. návrat na úvodní, přihlašovací stránku),
- `start()` slouží k nastartování autentizačního procesu v případě, kdy se neautorizovaný uživatel snaží přistoupit k obsahu vyžadující přihlášení.

[7] Autorizačního kódu z návratové URL po úspěšném přihlášení.
[8] Jestli je aktuální URL cesta určena pro Identity Providera.

```yaml
knpu_oauth2_client:
    clients:
        zuul:
            type: generic
            provider_class: App\Security\OAuth\Provider\Zuul
            provider_options:
                base_url: 'https://auth.fit.cvut.cz'
                scopes:
                    - 'cvut:umapi:read'
                    - 'cvut:sirius:personal:read'
                    - 'cvut:cpages:common:read'
                    - 'cvut:kosapi:read'
            client_id: '%oauth.client_id%'
            client_secret: '%oauth.client_secret%'
            redirect_route: auth_zuul_check
```

Výpis kódu 4.3: Konfigurace OAuth2 klienta v `knpu_oauth2_client.yaml`

```yaml
security:
    enable_authenticator_manager: true
    providers:
        user_provider:
            id: ...\UserProvider
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: user_provider
            custom_authenticators:
                - ...\ZuulAuthenticator
            entry_point: ...\ZuulAuthenticator
            logout:
                path: auth_logout
                target: index
```

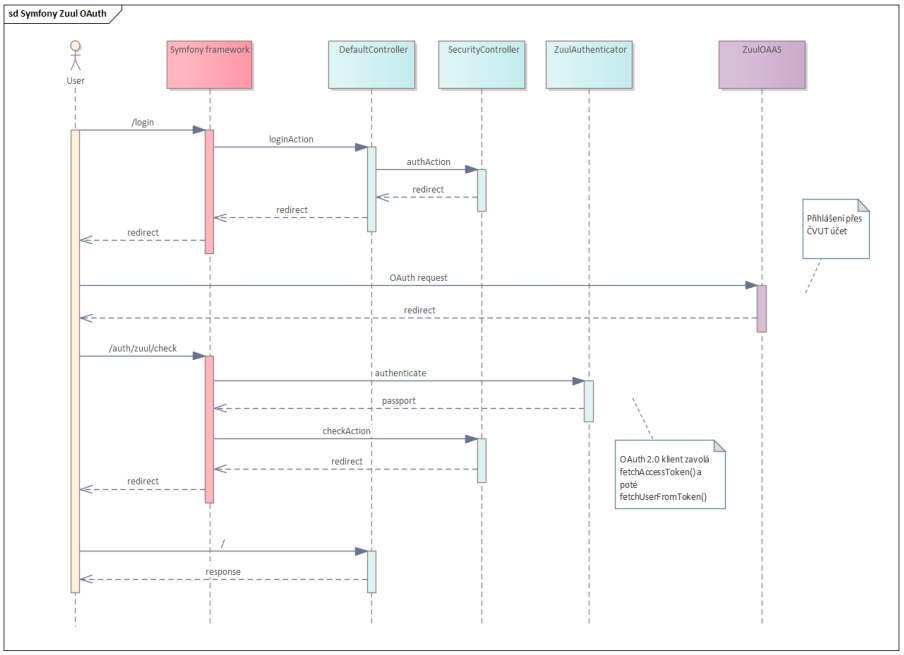
Výpis kódu 4.4: Konfigurace Symfony Security v `security.yaml`

Následně je potřeba provést konfiguraci balíčků. Balíček *OAuth2 Client Bundle* lze nakonfigurovat v konfiguračním souboru `knpu_oauth2_client.yaml` ve složce *config/packages*. Konfigurace pro aplikaci WhereIS s výše popsanými a vytvořenými třídami, požadovanými scopes a přidělenými hodnotami `CLIENT_ID` a `CLIENT_SECRET` je vidět ve výpisu 4.3.

Balíček *Symfony Security Bundle* je nutné nakonfigurovat v konfiguračním souboru `security.yaml` ve složce *config/packages*. Konfigurace tohoto balíčku společně s přiřazením třídy `UserProvider` a definicí firewallů je vidět ve výpisu 4.4.

Jelikož je samotný proces autentizace a autorizace uživatele použitím těchto

balíčků relativně komplikovaný a dochází ke spoustě automatických volání metod a přesměrování, rozhodl se autor práce za pomoci kolegy Bc. Jakuba Jabůrka znázornit tento proces pomocí sekvenčního diagramu, vizte obrázek 4.3.



Obrázek 4.3: Sekvenční diagram autorizace a autentizace uživatele

4.4.2 Chybějící indexy v databázi

Tuto neefektivitu ve vyhledávání lze vyřešit jednoduchým způsobem, a to přidá- ním databázových indexů k potřebným sloupcům. Pokud autor vezme v potaz již nově přidané entity, budou indexy přidány pro následující sloupce:

místnost, předmět – jméno a kód,

zaměstnanec – celé jméno a přihlašovací jméno,

menza, kolej, jídlo, událost – jméno,

uživatel – přihlašovací jméno,

katedra – české a anglické jméno,

budova – kód a adresa.

Indexy lze entitám přidat jednoduše pomocí anotace `ORM\Index` nad název třídy entity, např. pro vytvoření indexu nad jménem menzy stačí do třídy `Canteen` ve jmenném prostoru `App\Entit` (složka *src/Entity*) přidat anotaci `#[ORM\Index(columns: ["name"], name: "IDX_CANTEEN_NAME")]`, kde `co-`

`lumns` určuje sloupce, nad kterými se index vytvoří, a `name` definuje název indexu. Takovýmto způsobem je vytvořen tradiční B-Stromový index.

B-Stromový index, jak popisují Koruga a Bača [27], je hierarchická struktura, konkrétně vyvážený strom, který má malou hloubku, ale velký faktor větvení. Díky tomu trvá vyhledávání $\mathcal{O}(\log n)$ operací, kde n je počet dat v tabulce. B-Stromový index je vhodné vytvářet nad daty s vysokou kardinalitou, tedy velkým počtem různých hodnot, což je v případě WhereIS splněno.

Samotné vyhledávání entit pro uživatelské dotazy je řešeno vyhledávacím nástrojem Sphinx, nicméně indexy nad daty v databázi aplikace jsou i tak důležité kvůli rychlému vyhledávání entit, které je používáno zejména v `updateOrPersist()` metodách `Repository` tříd.

4.4.3 Počítání rank hodnoty výsledků a implementace vyhledávání relevantních výsledků

Ke zlepšení těchto schopností systému WhereIS je použit vyhledávací nástroj Sphinx. Informace a důkladný popis jeho integrace jsou napsány v kapitole 4.5.

4.4.4 Propojenost a celistvost dat

V rámci rešerše existujícího prototypu 2.2.5 byl nalezen problém v neúplnosti dat, tedy, že data jsou získávána pouze z jednoho zdroje, kvůli tomu mohou být neúplná, a v databázi nejsou nijak vzájemně propojená. Autor pro tyto nedostatky v následujících podsekcích navrhne řešení.

4.4.4.1 Propojenost

Základní problém tohoto nedostatku je považování určitého zdroje dat za „jeden zdroj pravdy“, tedy, že data jsou brána pouze z tohoto jediného zdroje a žádného jiného. To lze lehce vidět například u místností či zaměstnanců, které byly v prototypu získávány z KOSapi. Prototyp tak neměl v databázi všechny zaměstnance fakulty (pouze učitele) a místnosti na fakultě (pouze učebny) a vznikly tak „díry“ v datech. Tyto díry v datech se však mohou v aplikaci vyskytovat nadále. V době psaní této práce je to dáno zejména roztříštěností Fakulty Informačních Technologií do několika budov, například posluchárny v nové budově T9 patří oficiálně pod Fakultu Architektury. Dalším důvodem jsou samotné školní zdroje a systémy, které nemají vždy nejaktuálnější data.

K vyřešení tohoto problému je nutné získávat data z více zdrojů a existující entity v databázi aktualizovat těmito získanými daty, nebo v případě jejich absence v databázi vytvářet nové. Zaměstnance lze tak získávat jak z KOSapi (učitelé), tak i z webu FIT ČVUT, kde navíc krom neakademických zaměstnanců lze získat také fotografie zaměstnanců (i učitelů) na fakultě. Podobně lze postupovat pro místnosti, které lze získávat jak z KOSapi (učebny), tak i z kontaktních údajů zaměstnanců získaných z Umapi, kde jsou uvedené jejich kanceláře.

Aplikováním tohoto postupu na všechny entity dojde ke sloučení několika `Fetcher` tříd tak, že jedna `Fetcher` třída získává různá data z různých zdrojů a pomocí těchto dat aktualizuje i několik entit naráz. Z toho důvodu vznikly ve všech `Repository` třídách metody s názvem `updateOrPersist(Entity $e)`, které se pokusí najít entitu v databázi, a pokud ji najdou, entitu zaktualizují

```
public function updateOrPersist(Room $room): Room
{
    $roomFromDB = $this->findOneBy(...);

    if (empty($roomFromDB)) {
        $this->add($room);
        return $room;
    }

    $this->updateRoomData($roomFromDB, $room);
    return $roomFromDB;
}

private function updateRoomData(Room $roomFromDB, Room $room)
{
    $roomFromDB->setName($room->getName())
        ->setCode($room->getCode())
        ->setLocality($room->getLocality())
        ->setLastUpdate(new DateTimeImmutable())
        ->setIsClassroom($room->isClassroom())
        ->setFloorPlan($room->getFloorPlan());
}
```

Výpis kódu 4.5: Implementace metody `updateOrPersist` ve třídě `RoomRepository`

daty z obdrženého objektu, a pokud entita v databázi není, dojde k jejímu uložení. Jednotlivé `Fetcher` třídy tak pracují s více jak jednou `Repository` třídou naráz. Příklad implementace metody `updateOrPersist(Entity $e)` je vidět ve výpisu 4.5.

4.4.4.2 Celistvost

Jednotlivé entity v databázi je vhodné mezi sebou více provázat. K tomu je potřeba vytvořit vazby v databázi mezi souvisejícími entitami (cizí klíče). Lze tak vytvořit vazbu mezi zaměstnanci a učiteli a vědět tak, jaký zaměstnanec učí jaký předmět, nebo mezi místnostmi a zaměstnanci a vědět tak, jaká kancelář patří jakému zaměstnanci, nebo mezi budovami a místnostmi, katedrami, menzami a kolejemi a vědět tak, kde se jednotlivé entity fyzicky nachází.

Tyto vztahy lze definovat několika způsoby, avšak nejjednodušší způsob je využití konzolového příkazu `php bin/console make:entity`, který po zadání názvu již existující entity umožní entitě přidat další atributy. Při zadávání typu atributu stačí zadat `relation` a poté definovat, k jaké entitě a jakým typem vazby má být zvolená entita vázaná. Příkaz pak vše vygeneruje na základě zadaných údajů a jediné, co zbývá, je vytvoření a provedení migrace.

4.4.5 Problémy fungování příkazu `fetch`

Jak vychází najevo z 2.2.6, příkaz `fetch` má tři základní problémy. Problém související se spouštěním příkazu, kdy aplikace běží, lze vyřešit jednoduše za

```
#[AsSchedule('fetch_scheduler')]
class ScheduleProvider implements ScheduleProviderInterface
{
    public function getSchedule(): Schedule
    {
        return (new Schedule())
            ->add(RecurringMessage::every('1 day',
                new RunFetchCommand(), from: ...));
    }
}
```

Výpis kódu 4.6: Implementace `Scheduler` třídy

použití databázových transakcí popsaných v 4.1.5. Jelikož každá `Fetcher` třída v sobě obsahuje třídu `EntityManager` pro ukládání entit do databáze, lze tuto třídu využít i pro práci s transakcemi. K tomu implementuje třída `EntityManager` tři základní metody:

`beginTransaction()` – započne novou transakci,

`commit()` – všechny změny/příkazy v transakci provede a uloží do databáze,

`rollback()` – všechny změny/příkazy v transakci zahodí (pokud dojde během transakce k chybě/výjimce).

Problémy související s manuálním spouštěním příkazu a aktualizací dat lze vyřešit naráz pomocí job scheduleru (plánovače práce) popsaného v 4.1.6. Autor se rozhodl využít *Symfony Scheduleru*, který pro správné fungování potřebuje vytvořit následující třídy:

`Message` třída – jakákoliv třída s názvem zprávy, v tomto případě `RunFetchCommand`, třída může být prázdná, nemusí implementovat žádné metody nebo obsahovat žádné členské proměnné,

`Scheduler` třída – třída, která definuje rozvrh zasílaných zpráv (jaká zpráva se po jaké době má poslat) a podle něj zprávy posílá, třída se zaregistruje jako `Scheduler` s určitým názvem pomocí anotace `#[AsSchedule('název_scheduleru')]` nad definicí třídy a musí implementovat rozhraní `ScheduleProviderInterface`, příklad implementace této třídy je možné vidět ve výpisu 4.6,

`Handler` třída – třída, která provede daný kus kódu po obdržení příslušné zprávy, tento kód se provádí v metodě `__invoke(Message $m)`, třída se zaregistruje jako `Handler` pomocí anotace `#[AsMessageHandler]` nad definicí třídy, v tomto případě se jedná o `RunFetchCommandHandler` třídu, která zavolá metodu `fetch` ve třídě `FetchManager`.

4.4.6 Příprava databáze pro příkaz `fetch`

Jak je uvedeno v 2.2.7, implementace metody `prepareDB` ve `Fetcher` třídách není problematická pro většinu existujících entit. Problém však nastává v přípa-

dě perzistence oblíbených entit uživatele, kdy po smazaní dané entity z databáze dojde ke ztrátě vazby mezi uživatelem a entitou. Z toho důvodu a kvůli změnám fungování `Fetcher` tříd popsaných v 4.4.4 se autor rozhodl tuto metodu zcela odstranit. Databáze se pro příkaz `fetch` tedy nijak nepřipravuje. Data v databázi se aktualizují z nově získaných dat či se vytvoří nová a stará nepoužívaná data se po určité době smažou.

4.4.7 Způsob fungování vyhledávání

Jak bylo řečeno v kapitole 2.2.9, aktuální způsob fungování vyhledávání má 3 zásadní problémy. Přestože se tyto problémy týkají zejména frontendu, jejich řešení se dotýká i backendové části. V rámci změny logiky způsobu vyhledávání je potřeba:

- implementovat REST endpointy pro vyhledávání a získání detailu výsledku,
- implementovat 2 typy výsledků/pohledů:

  preview – slouží pro zobrazení nezbytných informací na stránce s výsledky k jednoznačné identifikaci vráceného výsledku,

  detail – slouží k zobrazení všech informací k danému výsledku (po kliknutí na preview výsledku).

Jak je vidět v kapitole 4.3.1, autor již s touto změnou fungování počítal, a proto jsou v návodu popsány metody `findPreview` a `getDetail`. Samozřejmě se může stát, že některé entity nechají těla některých z metod prázdné, protože je nebudou pro své fungování potřebovat, např. `FreeRoomItem` nepotřebuje implementovat metodu `getDetail`.

4.4.8 Citlivé údaje v kódu

K vyřešení tohoto problému se dá využít vestavěného řešení od *Symfony*, a to *Symfony's Secrets Management System*, popsané v 4.1.4. Pro usnadnění pochopení fungování a pracování s tímto systémem se autor rozhodl sepsat krátký návod.

1. Nejprve je potřeba vygenerovat dvojici asymetrických klíčů pro příslušné prostředí (pokud již nebyly vygenerovány). Dvojice klíčů pro vývoj se vygeneruje spuštěním konzolového příkazu `php bin/console secrets:generate-keys`. Pokud chcete vygenerovat dvojici klíčů pro produkci, je potřeba specifikovat prostředí přidáním `APP_RUNTIME_ENV=prod` před příkazem výše. Tento způsob změny prostředí se dá použít na všechny dále zmíněné příkazy, proto autor tuto skutečnost již znovu neuvádí.
2. Následně stačí přidat citlivou informaci do „trezoru“ pomocí konzolového příkazu `php bin/console secrets:set NÁZEV_SECRET_HODNOTY`. Po spuštění tohoto příkazu je uživatel vyzván k zadání hodnoty citlivé informace[9].

[9]Pozor, při zadávání hodnoty do konzole se pro větší bezpečí nic neukazuje. Neznamená to však, že by se nic nezadávalo.

```
parameters:
    api.agata_key: '%env(AGATA_API_KEY)%'
    ...

services:
    ...

    App\Service\Resource\AgataResource:
        arguments:
            $apiKey: '%api.agata_key%'
```

Výpis kódu 4.7: Příklad práce s citlivými hodnotami v `services.yaml`

- Pokud chcete zkontrolovat správně zadanou hodnotu, stačí použít konzolový příkaz `php bin/console secrets:list --reveal`, který vypíše doposud všechny uložené hodnoty v trezoru.
- Pokud jste při zadávání hodnoty udělali chybu, stačí spustit znovu příkaz pro uložení citlivé informace se stejným názvem `NÁZEV_SECRET_HODNOTY`.
- Pokud chcete citlivou informaci smazat úplně, stačí použít konzolový příkaz `php bin/console secrets:remove NÁZEV_SECRET_HODNOTY`.
- Pro všechny takto přidané hodnoty (včetně vygenerované dvojice klíčů) jsou vytvořeny zašifrované soubory ve složce *config/secrets/{env}/*.

3. Abyste mohli používat hodnotu citlivé informace v dané třídě, je potřeba upravit konfigurační soubor `services.yaml` ve složce *config*. Ihned na začátku souboru se nachází definice `parameters`, ve které lze definovat název citlivé hodnoty, která slouží jako reference v tomto souboru, a následně přístup k samotné citlivé informaci v „trezoru“. Pro přístup k citlivé informaci se používá syntaxe `%env(NÁZEV_SECRET_HODNOTY)%`, např. `api.agata_key: '%env(AGATA_API_KEY)%'`. Poté už stačí v definici `services` přidat cestu k samotnému objektu a pomocí `arguments` předat citlivou hodnotu do konstruktoru dané třídy.
   - Pokud chcete změnit prostředí (dev, prod), je potřeba v souboru `.env` v hlavní složce projektu změnit hodnotu proměnné `APP_ENV`.
   - Příklad definice a předání citlivé hodnoty v konfiguračním souboru `services.yaml` je vidět ve výpisu 4.7.
   - Příklad konstruktoru objektu, kterému jsou takto předávány citlivé informace, je vidět ve výpisu 4.8.

4.4.9 Konfigurace autowiringu tříd

Symfony nabízí velmi jednoduché a elegantní řešení tohoto problému, a to je klíčové slovo `_instanceof`, které se používá v konfiguračním souboru `services.yaml`. Pod tímto klíčovým slovem lze pro každý interface definovat

```
class AgataResource
{
    /**
     * @param string $apiKey
     ...
     */
    public function __construct(private string $apiKey, ...)
    {
    }

    ...
}
```

Výpis kódu 4.8: Příklad konstruktoru objektu přijímající citlivou hodnotu definovanou v `services.yaml`

```
    _instanceof:
        App\Service\Item\Item:
            tags: [ 'item_tag' ]
```

Výpis kódu 4.9: Použití `_instanceof` v souboru `services.yaml` pro třídy implementující `Item` rozhraní

pole tagů, které se budou používat pro následný autowiring. Symfony pak pro členskou proměnnou, u které se použije příslušný tag, provede autowire kolekce implementující rozhraní `Iterable`, ve které se nachází všechny třídy, které implementují rozhraní pod příslušným tagem. Příklad použití tohoto klíčového slova v souboru `services.yaml` je vidět ve výpisu kódu 4.9 a příklad použití tagu ve třídě `SearchService` je vidět ve výpisu 4.10. Je nutné upozornit, že třídy jsou v kolekci seřazené podle abecedního pořadí.

4.5 Integrace vyhledávacího nástroje Sphinx

Vyhledávání výsledků pro uživatelské dotazy je nově v aplikaci WhereIS řešeno vyhledávacím nástrojem Sphinx verze 3.X. Pro správné fungování vyhledávání je potřeba nakonfigurovat (batchové) indexy nad jednotlivými entitami, které se vyhledávají. K této konfiguraci dochází v konfiguračním souboru s názvem `sphinx.conf`. Kromě indexů se v tomto souboru definují také zdroje, odkud má Sphinx data tahat, a konfigurace vyhledávacího démona `searchd`, který se stará o samotné vyhledávání. Indexy jsou následně vytvářeny a aktualizovány programem `indexer`. Autor popíše tyto tři důležité části v samostatných sekcích.

4.5.1 Konfigurační soubor `sphinx.conf`

Jedná se o nejdůležitější soubor v rámci celé konfigurace Sphinx. V tomto souboru se konfigurují zdroje dat, batchové indexy a samotný vyhledávací démon `searchd`. Tento soubor se nachází se složce *sphinx*.

```
readonly class SearchService
{
    public function __construct(
    #[TaggedIterator('item_tag')] private iterable $items, ...)
    {
    }

    ...

    public function findPreview(QueryString $value): array
    {
        ...
        foreach ($this->items as $item) {
            $results[] = $item->findPreview($value);
        }
        ...
    }
}
```

Výpis kódu 4.10: Příklad použití `_instanceof` tagu ve třídě `SearchService` pro `Item` třídy

Zdroje dat Zdroje dat se v souboru `sphinx.conf` definují pomocí klíčového slova `source`. Za tímto klíčovým slovem následuje název zdroje a tělo s dalšími parametry konfigurující tento konkrétní zdroj. Nejdůležitější z těchto parametrů jsou:

`type` – typ připojení (např. `pgsql` – PostgreSQL, `mysql` – MySQL, `ODBC` – ODBC připojení...),

`sql_host` – typ zdroje (nejčastěji `database`),

`sql_user`, `sql_pass`, `sql_db`, `sql_port` – přihlašovací údaje do databáze (uživatel, heslo, název databáze, port),

`sql_query` – SQL SELECT dotaz vybírající z určité tabulky sloupce, se kterými index pracuje (např. `SELECT id, name, sphinx_types FROM canteen`).

Batchové indexy Batchové indexy se definují pomocí klíčového slova `index`. Za tímto klíčovým slovem následuje název indexu a tělo s dalšími parametry konfigurující tento konkrétní index. Nejdůležitější z těchto parametrů jsou:

`source` – název zdroje dat, který je použit pro vytvoření indexu (vizte 4.5.1),

`field_string` – definuje řetězcová pole indexu (tedy názvy sloupců vybrané SELECT dotazem ve zdroji, které budou indexovány a zároveň i vraceny jako součást nalezeného výsledku),

`morphology` – definuje stemmer, který je použit pro stematizaci řetězcových polí a dotazů nad indexem (např. `stem_en` či `stem_cz`),

`charset_table` – definuje tabulku znaků, které nemají být brány jako oddělovače, a transformační pravidla (např. A..Z –>a..z transformuje všechna velká písmena na malá),

`stopwords` – definuje soubory stop slov, které budou z řetězcových polí a dotazů odstraněny (soubory musí být textové, slova v nich oddělená mezerou a musí se nacházet ve složce {*datadir*}*/extra*, díky Dockeru stačí soubory vložit do složky *sphinx/extra*).

Pro správné ukládání batchových indexů je také nutné definovat složku na disku, do které se tyto indexy budou ukládat. To se dělá pomocí parametru `datadir` v těle klíčového slova `common`.

`Searchd` Vyhledávací démon `searchd` se konfiguruje pomocí klíčového slova `searchd` a dalších parametrů v těle za tímto slovem. Nejdůležitější z těchto parametrů jsou:

`listen` – definuje IP adresu s portem pro SphinxAPI či SphinxQL, na kterých démon poslouchá,

`read_timeout` – počet sekund, po kterém dojde k timeoutu vyhledávání,

`max_children` – maximální počet paralelních pracujících vláken, které vyhledávají.

4.5.2 Program `indexer`

`Indexer` pomocí konzolového příkazu `indexer --all --config cesta/k/sphinx.conf` vytvoří na základě konfiguračního souboru `sphinx.conf` batchové indexy. Tyto indexy jsou ukládané do složky definované v `sphinx.conf` jako soubory. Batchové indexy je také nutné pomocí tohoto programu aktualizovat, pokud se data v databázi mění. K aktualizaci indexů dojde pomocí konzolového příkazu `indexer --all --rotate --config cesta/k/sphinx.conf` a po aktualizaci zašle `indexer` signál vyhledávacímu démonu `searchd`, že došlo k jejich aktualizaci. Tento příkaz se spouští automaticky pomocí plánovače práce Cron (vizte 4.1.6) po 5 minutách kvůli běžící `Scheduler` třídě, která spouští jak příkaz `fetch` (popsáno v 4.4.5), tak i aktualizaci aktuálních jídelníčků menz.

4.5.3 Vyhledávací démon `searchd`

Vyhledávací démon `searchd` zpracovává uživatelské dotazy a provádí vyhledávání nad jednotlivými indexy vytvořenými `indexerem`[10]. Získané výsledky z indexů následně seřadí podle jejich váhy a vrátí je uživateli. Tento démon lze také chápat jako vyhledávací server, který poslouchá na IP adrese a portu definovaných v souboru `sphinx.conf`. Aplikace WhereIS komunikuje s tímto démonem ve třídě `SphinxSearch` (složka *src/Sphinx*), kde se kromě nastavení připojení definují také možnosti vyhledávání (rankovací funkce, maximum vrácených výsledků...). Samotný démon se spouští pomocí příkazu `searchd --config cesta/k/sphinx.conf`.

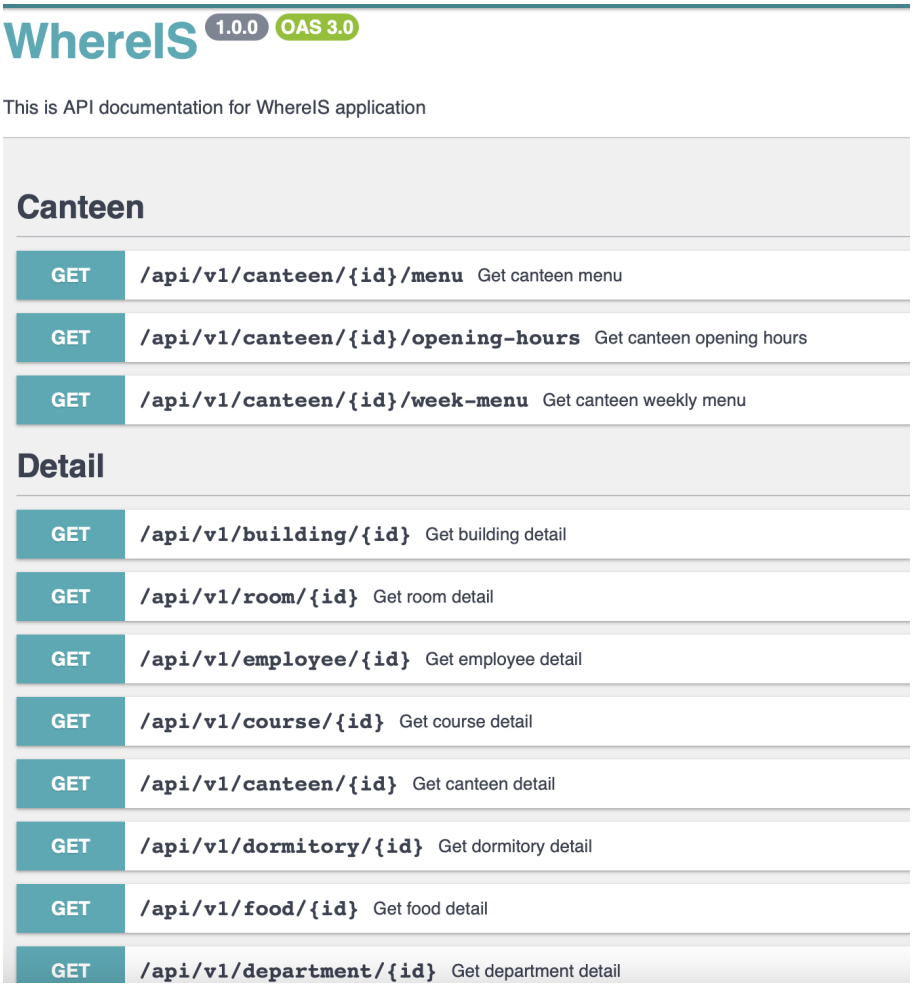
[10] Pokud se však nejedná o real time indexy, které nejsou tvořeny `indexerem`, ale samotným Sphinx enginem.

4.5.4 Pomocné sloupce pro indexování

Jak již autor popsal v kapitolách výše, indexy se konfigurují přes konfigurační soubor `sphinx.conf`. Pro každý index lze definovat velké množství full-textových polí, které jsou indexovány a používány při vyhledávání. Pro různé entity je tak vhodné udržovat si v databázi další, pomocné sloupce či data. Předměty lze tak například vyhledávat pomocí klíčových slov či osnov přednášek (vše bráno z KOSapi). Uživateli tedy stačí zadat dotaz typu „pumping lemma“ a zobrazí se mu předměty, které v klíčových slovech či v osnovách přednášek mají tento řetězec. Pro některé entity se také vyplatí definovat umělý sloupec popisující, o jakou entitu se jedná. Autor tedy pro entity typu událost, kolej, menza, budova a další vytvořil sloupec `sphinx_types` v příslušné tabulce entity, ve kterém definuje různá klíčová slova v českém a anglickém jazyce, pomocí kterých lze entity vyhledat. Díky tomu se uživateli po zadání dotazu typu „menza“ či „událost“ vrátí všechny menzy nebo události v databázi. Tento sloupec není definován pro všechny entity (např. předměty či učitelé), protože záznamů k těmto entitám je v databázi mnoho a nedávalo by smysl vracet všechny tyto záznamy.

4.6 REST API

V rámci této práce autor také vytvořil REST API, které využívá frontendová část aplikace vytvořená kolegou Bc. Illiou Brylovem. REST API obsahuje všechny endpointy potřebné pro vyhledávání, přístup k detailům jednotlivých entit a práci s uživatelskými preferencemi (oblíbené entity, zvolený jazyk). `Controller` třídy a tedy i tyto zdroje se nachází ve jmenném prostoru `App\Controller\REST` (složka *src/Controller/Rest*). Dokumentace API je dostupná pod endpointem `HOST_NAME/api/doc` (vizte ukázkový obrázek 4.4).



Obrázek 4.4: Ukázka dokumentace REST API pod endpointem `HOST_NAME/api/doc`

Kapitola 5

Nasazení a testování

5.1 Statická analýza kódu

Jak uvádí Louridas [28], statické analyzátory kódu jsou programy, které hledají chyby v kódu bez jejich spuštění. Tyto chyby hledají na základě známých a typických vzorů, kterých se programátoři opakovaně dopouští. Přestože statických analyzátorů v době psaní této práce existuje pro všechny populární programovací jazyky velké množství, všechny na pozadí fungují velmi podobně. Kód přečtou a sestaví vnitřní model (abstraktní reprezentace kódu), který následně používají pro hledání typických chybných vzorů. Krom toho provádí také analýzu průtoku dat v kódu. Ta je užitečná zejména u dynamicky typovaných programovacích jazyků, ve kterých není nutné uvádět datové typy proměnných. Tato analýza se snaží určit možné datové typy, kterých proměnné v kódu mohou nabývat. To je důležité zejména pro identifikaci bezpečnostních zranitelností.

Jelikož je backendová část aplikace WhereIS psaná v PHP, je potřeba použít statický analyzátor kódu právě pro tento jazyk. Nejčastěji využívaným statickým analyzátorem PHP je podle [29] PHPStan. Autor se tedy rozhodl také využít PHPStan, protože podle oficiálního webu analyzátoru [30] je možné jej přidat jako rozšíření do populárních PHP frameworků, jako je právě Symfony, a lze jej tak velmi jednoduše nainstalovat pomocí balíčkovacího nástroje `composer`. PHPStan se pak spustí z hlavní složky projektu pomocí konzolového příkazu `vendor/bin/phpstan analyse src`, kde `src` je složka souborů s kódem. Po dokončení příkazu dojde k vypsání nalezených chyb pro jednotlivé třídy/soubory. PHPStan umožňuje definovat úroveň striktnosti. Pokud je tedy nalezených chyb velké množství, je možné snížit tuto úroveň pomocí přepínače `-l` nebo `--level` a chyb se bude zobrazovat méně. Celkem je těchto úrovní v době psaní této práce 10, kdy nultá úroveň je nejzákladnější a devátá nejstriktnější. Výchozí zvolená úroveň je šestá, kterou používá i autor této práce.

5.2 Testování

Ve své předešlé práci [1] provedl autor heuristickou analýzu a uživatelské testování. Aplikace se za tu dobu velmi změnila a tak by dávalo smysl tato vyhodnocení provést znovu, nicméně pro testování backendové části není ani jedna z výše zmíněných možností vhodná. Heuristická analýza se zabývá použitelností

aplikace, což je spojené zejména s frontendovou částí. Uživatelské testování se dá provést, nicméně zdaleka neodhalí většinu chyb, testuje aplikaci spíše jako celek, než po jednotlivých funkcionalitách, a obecně je velmi drahé testovat něco uživatelem, když se nabízí možnost automatického testování. Uživatelské testování aplikace tedy vyhodnocuje ve své práci kolega Bc. Illia Brylov. Tato práce se zabývá automatizovaným testováním pomocí unit testů.

Jak uvádí Khorikov [31], unit test je automatizovaný test, který rychle a izolovaně otestuje malou část kódu (unit, jednotka). Izolovanost v tomto smyslu znamená, že pokud je testovaný kus kódu závislý na jiných třídách (využívá je), je nutné tyto třídy nahradit za *mocky*. Cílem mocku je tedy izolovat testovaný kus kódu a simulovat operace kódu, na kterých je testovaný kód závislý. Mock třídám lze tedy definovat, co se má při volání jakých metod s danými parametry vracet a lze tak jednoduše testovat velké množství situací. Unit testy také nesmí být na sobě závislé. Je tedy dobrým zvykem pouštět unit testy v náhodném pořadí pří každém spuštění. Frameworků pro psaní unit testů existuje v době psaní této práce velké množství. Pro psaní unit testů v jazyce PHP je hojně používaný *PHPUnit*.

5.2.1 PHPUnit

Jak uvádí sám autor frameworku Bergmann [32], PHPUnit je programátorsky orientovaný testovací framework pro PHP. Jedná se o instanci architektury *xUnit* pro frameworky pro unit testování. Lze jej velmi jednoduše instalovat do existujících projektů pomocí balíčkovacího nástroje `composer`. V případě frameworku Symfony je PHPUnit součástí testovacího balíčku Symfony s názvem *Test Pack*. Instalací tohoto balíčků dojde k automatickému vytvoření složky *tests* v projektu, ze které PHPUnit spouští unit testy, a souboru *.env.test* s proměnnými prostředí pro testování. Unit testy se spustí z kořenové adresáře projektu pomocí příkazu `php bin/phpunit`.

Pro psaní unit testů třídy je nutné vytvořit testovací třídu ve složce *tests*. Je doporučováno použít stejnou složkovou strukturu pro testy jako v projektu. Testovací třída by se měla jmenovat stejně jako testovaná třída se slovem „Test“ na konci. Testovací třídy musí dědit ze třídy `TestCase` z jmenného prostoru `PHPUnit`. Díky tomu je v rámci testovací třídy dostupné velké množství pomocných metod, jmenovitě stojí za zmínku:

`setUp()` – metoda, která je volána před každým unit testem,

`tearDown()` – metoda, která je volána po každém unit testu,

`createMock()` – metoda, která vytvoří mock instanci zvolené třídy (např. `$mock = $this->createMock(Food::class)`),

`expects()` – metoda očekávající počet volání určité metody mock třídy,

`method()` – definování metody mock třídy, pro kterou je očekáván počet volání a pro kterou lze definovat návratovou hodnotu,

`willReturn()` – metoda definující, co určitá metoda bude vracet za hodnotu.

Jednotlivé unit testy se pak v testovací třídě píší se slovem „test“ na začátku názvu. Metoda `fetch()` by se tedy testovala pomocí metody `testFetch()`.

Pokud chybí slovo „test“ na začátku názvu testovací metody, není metoda považována za testovací (unit test) a nebude spuštěna. Testovacích metod testujících jednu metodu bývá samozřejmě více než 1. Většina metod totiž kvůli `if else` či datům, se kterými pracují, nemá pouze jednu cestu průběhu a unit testy by měly otestovat všechny tyto průběhy (i vyhazování výjimek). Ve výsledku tedy mohou vzniknout testovací metody typu `testFetchSuccessful()`, `testFetchFail()` či `testFetchException()`.

Jelikož autor práce využívá `readonly` tříd, vzniká u PHPUnit problém s vytvářením jejich mocků. Metoda `createMock()` totiž funguje tak, že vytváří třídu, která dědí z původní třídy. V PHP lze dědit z `readonly` tříd, jenže dědící třída musí být také označena jako `readonly`, k čemuž v případě mockovaných tříd v PHPUnitu nedochází. Z toho důvodu musel autor využít balíčku *Bypass readonly*[11], který umožňuje vytváření mock tříd z `readonly` tříd tak, že mockovanou třídu převede z `readonly` na klasickou. Pro zapnutí tohoto převodu je nutné v každém unit testu, kde dochází k vytváření mock třídy z `readonly` třídy, zavolat metodu `BypassReadonly::enable()`.

5.2.2 Unit testy vyžadující databázi

Testování tříd, které interagují s databází, se od testování ostatních tříd relativně liší. V aplikaci WhereIS se jedná zejména o `Repository` třídy. U takovýchto tříd je nutné testovat, jestli se provedené operace opravdu propsaly do databáze a jestli všechny změny jsou v databázi. Pro tyto účely je podle [33] potřeba vytvořit novou databázi pro testy. Typ databáze použitý pro testování se může lišit od typu databáze používaného pro normální fungování aplikace. Lze tedy zvolit základní databázový systém SQLite. Je potřeba nastavit proměnnou `DATABASE_URL` v souboru proměnných prostředí `.env.test`. Poté stačí vytvořit databázi pomocí konzolového příkazu `php bin/console --env=test doctrine:database:create` a vytvořit v ní schéma pomocí příkazu `php bin/console --env=test doctrine:schema:create`. Pro jednodušší práci s databází během testování používá autor balíček *Doctrine test bundle*[12], který automaticky vyčistí databázi před a po každém unit testu.

Testovací třídy `Repository` tříd se také liší od ostatních testovacích tříd. Je potřeba dědit ze třídy `KernelTestCase` z jmenného prostoru `Symfony\Bundle\FrameworkBundle\Test`. Následně je potřeba získat instanci třídy `EntityManager` pro práci s entitami a databází, toho lze docílit např. v `setUp()` zavoláním `$kernel->getContai- ner()->get('doctrine')->getManager()`. Následně lze již s instancí třídy `EntityManager` běžně pracovat. Příklad unit testu `Repository` třídy společně s metodou `setUp()` je vidět ve výpisu 5.1.

5.3 CI/CD

CI/CD, neboli Continuous Integration / Continuous Delivery, má za cíl podle [34] zefektivnit a urychlit životní cyklus vývoje softwaru. Continuous Integration označuje praxi automatické a časté integrace změn kódu do sdíleného úložiště (repozitář – nejčastěji Git). Continuous Delivery / Deployment je dvoudílný proces, který se týká integrace, testování a nasazení/dodávání změn

[11] `https://github.com/zoltanka/bypass-readonly`
[12] `https://github.com/dmaicher/doctrine-test-bundle`

```
protected function setUp(): void
{
    $kernel = self::bootKernel();
    $this->entityManager = $kernel
        ->getContainer()
        ->get('doctrine')
        ->getManager();
    $this->buildingRepository = $this
        ->entityManager
        ->getRepository(Building::class);
}

public function testAdd(): void
{
    $building = new Building('TK', 'Adresa 1', ...);
    $this->buildingRepository->add($building);

    $this->assertNotNull($building->getId());
    $this->assertEquals('TK', $building->getCode());
    $this->assertEquals('Adresa 1', $building->getAddress());
}
```

Výpis kódu 5.1: Příklad unit testu `BuildingRepository` třídy společně s metodou `setUp()`

kódu. Continuous Delivery se zastaví před automatickým nasazením do produkce, zatímco Continuous Deployment automaticky uvolňuje aktualizace do produkčního prostředí.

Jelikož je aplikace WhereIS verzována na školním GitLab serveru, rozhodl se autor přidat Continuous Integration do repozitáře aplikace. Continuous Delivery / Deployment není v rámci této práce realizováno. Continuous Integration je v GitLab repozitáři realizováno pomocí vestavěné podpory GitLab CI/CD. Pro zprovoznění CI stačí v kořenové složce projektu vytvořit soubor *.gitlab-ci.yml*. V tomto souboru se definují jednotlivé *stage* (lze chápat jako logický kus práce). Každá stage obsahuje několik prací, které vykonává. Stage lze na sebe navazovat. Pokud předchozí stage neuspěla, další nebude ani spuštěna. Autor do CI zahrnul spouštění unit testů (stage „test“) a statickou analýzu kódu (stage „phpstan“). Jelikož je potřeba nainstalovat nástroje a balíčky pro tyto operace a připravit databázi pro unit testy, přidal autor do `.gitlab-ci.yml` souboru klíčové slovo `before_script`, které provede vypsané operace před začátkem práce.

5.4 Nasazení na server

Z důvodu usnadnění práce při instalování všech potřebných technologií a závislostí pro nasazení aplikace na server se autor rozhodl použít virtualizační nástroj Docker, který autor využíval již v předešlé práci [1]. Díky Dockeru nebude potřeba na hostitelském zařízení instalovat a konfigurovat všechny potřebné technologie a závislosti. Vše potřebné se provede v samotných kontej-

nerech jednotlivých procesů. Vzhledem k množství změn a novým technologií, které aplikace WhereIS využívá, je nutné upravit Docker compose soubory `compose.yaml` a `compose.override.yml` a přidat nové kontejnery pro nově používané technologie a procesy. Hlavní myšlenka Docker kontejnerů totiž je, že jeden kontejner rovná se jeden proces. V případě aplikace WhereIS se bude jednat o následující kontejnery:

`php` – kontejner s PHP, ve kterém běží server,

`database` – kontejner s PostgreSQL databází,

`scheduler` – kontejner s PHP, ve kterém běží `Scheduler` proces posílající zprávy pro aktualizaci jídelníčků a spouštění `fetch` příkazu (vizte 4.4.5),

`sphinx` – kontejner s vyhledávacím nástrojem Sphinx, ve kterém běží démon `searchd`,

`node` – kontejner s NodeJS spouštějící `yarn build` pro kompilaci souborů frontendu.

Všechny tyto kontejnery jsou již nakonfigurované a pro spuštění aplikace tedy stačí použít konzolový příkaz `docker compose up` (více ke spouštění aplikace vizte přílohu B). Poslední věc, na kterou se nesmí zapomenout, je automatická aktualizace batchových indexů ve `sphinx` kontejneru. Jelikož `indexer` posílá démonu `searchd` signál o provedené aktualizaci indexů, je nutné tuto aktualizaci provádět přímo uvnitř kontejneru `sphinx`. Nelze tedy vytvořit další kontejner, který by prováděl tuto aktualizaci. Proto se autor rozhodl využít plánovače práce Cron (vizte sekci 4.1.6), pomocí kterého lze každých X minut vlézt do kontejneru `sphinx` a spustit aktualizaci indexů. Do cron tabulky UNIXového systému stačí přidat pro automatickou aktualizaci indexů po pěti minutách tento řádek: `*/5 * * * * docker exec sphinx indexer --all --config /opt/sphinx/conf/sphinx.conf --rotate`.

Závěr

Na závěr práce by autor rád připomněl cíle práce a zhodnotil, jestli došlo ke splnění všech cílů dle očekávání. Následně autor rozebere možnosti budoucího možného vývoje a rozšíření aplikace.

Zhodnocení práce

Hlavním cílem práce bylo zlepšit a dokončit backendovou část aplikace WhereIS tak, aby byla provozu schopná a dala se nasadit. V rámci zlepšení bylo hlavním cílem vylepšit vyhledávací schopnosti aplikace a opravit problémy a vyřešit nedostatky v původním prototypu. Hlavní cíl práce považuje autor za splněný, jelikož všechny nalezené problémy v rešerši prototypu byly v praktické části adresovány, a vyhledávací schopnosti aplikace se mnohem zlepšily díky vyhledávacímu nástroji Sphinx, který byl do aplikace integrován. V rámci praktické části práce došlo také k integraci většiny uživatelských požadavků, které byly získány z vypracovaného dotazníku, a s nimi došlo také k integraci nových zdrojů dat jako Agáta, Courses či webových stránek FIT a SUZ ČVUT a velkého množství nově vyhledatelných entit jako jsou menzy, jídla, katedry, koleje, události a budovy. Posledními cíli praktické části byly implementace REST API, které je zdokumentováno v rámci aplikace na endpointu `HOST_NAME/api/doc`, a aplikaci otestovat a nasadit. Poslední cíle tedy autor považuje také za splněné, protože aplikace byla otestována jak pomocí unit testů, tak i pomocí statického analyzátoru kódu, které autor navíc přidal do GitLab CI, a aplikace byla nasazena na server, a je tak dostupná k používání a uživatelskému testování.

V rámci teoretické části se autor zaměřil především na rešerši existujících řešení a vyhledávacích nástrojů. Dále také analyzoval problémy a nedostatky existujícího prototypu. V rámci analýzy autor společně s kolegou Bc. Illiou Brylovem vypracoval dotazník pro získání požadavků od uživatelů aplikace a tyto požadavky následně společně s nově potřebnými zdroji dat zanalyzoval. Došlo také ke zhodnocení, které požadavky budou do systému integrovány a které nikoliv. Ke konci teoretické části popsal autor nový doménový model aplikace.

V rámci nesplněných cílů by autor rád zmínil přetrvávající problém se školními právy, které zamezily implementaci nejbližších událostí zaměstnanců. Autor je tedy nucen nezahrnout tuto funkcionalitu do výsledné aplikace. Dále

se dá za částečně nesplněný úkol považovat integrace uživatelských požadavků. Autor se rozhodl nezaintegrovat do systému 2 požadavky. První z nich, který se týkal přidání různých specifických míst (toalety, respiria, kuchyňky...), nepřidal autor z toho důvodu, že v době psaní této práce neexistuje žádná webová služba či systém, ze kterého by se tyto informace daly jednoduše získat. Druhý z těchto požadavků se týkal integrace izolovaných školních systémů (ProgTest, Marast, DBS-Portál...). Autor by velmi rád tyto systémy integroval do aplikace WhereIS, nicméně žádný z těchto systémů nenabízí API či jakýkoliv jiný přístup ke svým datům (krom klasického přihlášení přes ČVUT účet). Z toho důvodu není v době psaní této práce možné tyto systémy do WhereIS integrovat.

Budoucí možný vývoj

Existuje nespočet dalších možných funkcionalit, které by se do systému daly přidat. Neustále se nabízí možnost rozšiřovat aplikaci o nové entity a zdroje dat. Lze tak například integrovat webovou stránku ÚTVS ČVUT[13], ze které lze získávat všechny dostupné tělocviky na univerzitě a případně také i jejich paralelky společně s informacemi, kdy a kde se paralelka koná a jak je zaplněná. Dále se do aplikace dají přidat již několikrát zmíněná specifická místa jako toalety, respiria či kuchyňky.

Dalším velkým milníkem by pro aplikaci WhereIS bylo rozšíření na celouniverzitní úroveň. Aktuálně se aplikace soustředí zejména na FIT ČVUT, nicméně studenti studují předměty i jiných fakult a navštěvují tak nové, neznámé budovy, ve kterých se neorientují a jednoduše ztrácejí. Proto by přidání budov všech fakult ČVUT společně s jejich místnostmi, předměty a učiteli ulehčilo život nejen studentům na FIT ČVUT, ale také všem ostatním studentům na fakultách ČVUT.

Kromě přidávání nových entit a zdrojů dat by se také dala přidat funkcionalita týkající se sdílení rozvrhů uživatelů. Aktuálně není možné pomocí žádné školní aplikace zobrazit si rozvrh jiného studenta (což vzhledem k ochraně osobních údajů a GDPR dává smysl). Nicméně pokud by daný uživatel udělil souhlas, aby jiný uživatel s daným přihlašovacím jménem ČVUT měl přístup k jeho rozvrhu, bylo by tak vše v pořádku a uživatelé by si takto jednoduše mohli sdílet rozvrhy. V rámci aplikace WhereIS by se takový způsob sdílení rozvrhu dal implementovat a díky tomu by si kamarádi mohli jednoduše sdílet rozvrhy mezi sebou, než aby si posílali PDF export rozvrhu ze systému KOS.

[13] `https://www.utvs.cvut.cz/vyuka/povinna-volitelna`

Bibliografie

1. HEGER, Tomáš. *Aplikace WhereIS*. Praha, 2022. Dostupné také z: https://dspace.cvut.cz/bitstream/handle/10467/102082/F8-BP-2022-Heger-Tomas-thesis.pdf?sequence=-1%5C&isAllowed=y. Bakalářská práce. FIT ČVUT.
2. CERI, Stefano; BOZZON, Alessandro; BRAMBILLA, Marco; VALLE, Emanuele Della; FRATERNALI, Piero; QUARTERONI, Silvia. *Web Information Retrieval* [online]. První. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013 [cit. 2024-02-19]. ISBN 978-3-642-39314-3. Dostupné z DOI: 10.1007/978-3-642-39314-3.
3. BUTTCHER, Stefan; CLARKE, Charles L. A.; CORMACK, Gordon V. *Information Retrieval: Implementing and Evaluating Search Engines*. Reprint. MIT Press, 2016. ISBN 978-0262528870.
4. MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. *Stemming and lemmatization* [online]. 2008. [cit. 2024-02-19]. Dostupné z: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.
5. PIBIRI, Giulio Ermanno; VENTURINI, Rossano. Techniques for Inverted Index Compression. *ACM Computing Surveys* [online]. 2021-11-30, roč. 53, č. 6, s. 1–36 [cit. 2024-02-19]. ISSN 0360-0300. Dostupné z DOI: 10.1145/3415148.
6. TURNBULL, Doug. *BM25 The Next Generation of Lucene Relevance* [online]. [cit. 2024-02-25]. Dostupné z: https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/.
7. FOGLIA, Charlotte. *Understanding Search-Based Applications* [online]. 2023. [cit. 2024-02-22]. Dostupné z: https://www.sinequa.com/resources/blog/understanding-search-based-applications/.
8. BIAŁECKI, Andrzej; MUIR, Robert; INGERSOLL, Grant. Apache Lucene 4. In: *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval* [online]. První. Dunedin, New Zealand: Department of Computer Science, University of Otago, 2012, s. 17–24 [cit. 2024-02-21]. ISBN 978-0-473-22026-6. Dostupné z: https://www.researchgate.net/profile/Andrzej-Bialecki/publication/260282732_Apache_

Lucene_4/links/5ede6a2545851516e65f1fa6/Apache-Lucene-4.pdf%5C#page=22.

9. SHARMA, Atri. *Practical Apache Lucene 8* [online]. První. Apress Media LLC, 2020 [cit. 2024-02-22]. ISBN 978-1-4842-6345-7. Dostupné z DOI: 10.1007/978-1-4842-6345-7.
10. STROHMAN, Trevor; METZLER, Donald; TURTLE, Howard; CROFT, W. Bruce. *Indri: A language-model based search engine for complex queries* [online]. Amherst, USA, 2005 [cit. 2024-02-22]. Dostupné z: https://ciir.cs.umass.edu/pubfiles/ir-407.pdf. vědecký článek. University of Massachusetts Amherst.
11. *The Lemur Project* [online]. 2000. [cit. 2024-05-08]. Dostupné z: https://www.lemurproject.org.
12. AKSYONOFF, Andrew. *Sphinx overview* [online]. 2001. [cit. 2024-02-23]. Dostupné z: http://sphinxsearch.com/about/sphinx/.
13. NUGRAHA, Arie. Indexing Bibliographic Database Content Using MariaDB and Sphinx Search Server. *Code4Lib Journal* [online]. 2014, roč. 2014, č. 25, s. 10 [cit. 2024-02-23]. ISSN 1940-5758. Dostupné z: https://journal.code4lib.org/articles/9793.
14. YANTSAN, Ellie. *Sphinx – the Beginner's Guide* [online]. 2008. [cit. 2024-02-23]. Dostupné z: https://www.mageworx.com/blog/sphinx-the-beginners-guide.
15. AKSYONOFF, Andrew. *Sphinx3 Documentation* [online]. 2001. [cit. 2024-02-23]. Dostupné z: http://sphinxsearch.com/docs/sphinx3.html.
16. HARTINGER, David. *Lekce 4 - UML - Doménový model* [online]. 2017. [cit. 2024-03-12]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram.
17. *Security* [online]. 2009. [cit. 2024-02-05]. Dostupné z: https://symfony.com/doc/current/security.html.
18. THÉR, Bc. Jan. *Extrakce dat z webu pomocí web scrapingu*. Hradec Králové, 2022. Dostupné také z: https://theses.cz/id/gkn516/STAG96371.pdf. Diplomová práce. Univerzita Hradec Králové.
19. KHDER, Moaiad. Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. *International Journal of Advances in Soft Computing and its Applications* [online]. 2021-12-30, roč. 13, č. 3, s. 145–168 [cit. 2024-02-05]. ISSN 27101274. Dostupné z DOI: 10.15849/IJASCA.211128.11.
20. *How to Keep Sensitive Information Secret* [online]. 2009. [cit. 2024-02-05]. Dostupné z: https://symfony.com/doc/current/configuration/secrets.html.
21. ZANINI, Antonello. *Database Transactions 101: The Essential Guide* [online]. Stockholm, 2003 [cit. 2024-03-11]. Dostupné z: https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/.
22. AWATI, Rahul. *Job Scheduler* [online]. 2000. [cit. 2024-03-11]. Dostupné z: https://www.techtarget.com/searchdatacenter/definition/job-scheduler.

23. HIRA, Zaira. *How to Automate Tasks with cron Jobs in Linux* [online]. 2014. [cit. 2024-03-11]. Dostupné z: https://www.freecodecamp.org/news/cron-jobs-in-linux/.
24. *Scheduler* [online]. 2009. [cit. 2024-02-05]. Dostupné z: https://symfony.com/doc/current/scheduler.html.
25. *Databases and the Doctrine ORM* [online]. 2009. [cit. 2024-02-05]. Dostupné z: https://symfony.com/doc/current/doctrine.html.
26. *SpamSpan — An unobtrusive anti-spam solution for accessible email obfuscation* [online]. 2006. [cit. 2024-02-05]. Dostupné z: https://www.spamspan.com.
27. KORUGA, Petra; BAČA, Miroslav. Analysis of B-tree data structure and its usage incomputer forensics [online]. 2010, s. 6 [cit. 2024-02-05]. Dostupné z: https://www.researchgate.net/publication/210381551_Analysis_of_B-tree_data_structure_and_its_usage_in_computer_forensics%5C#:~:text=B%5C%2Dtree%5C%20is%5C%20a%5C%20fast,to%5C%20fit%5C%20in%5C%20main%5C%20memory..
28. LOURIDAS, Panos. Static code analysis. In: *IEEE Software* [online]. 4. vyd. IEEE, 2006, sv. 23, s. 58–61 [cit. 2024-04-18]. Č. 4. ISSN 0740-7459. Dostupné z DOI: 10.1109/MS.2006.114.
29. SPAETZEL, John. *The top PHP static code analysis tools of 2024* [online]. 2010. [cit. 2024-04-18]. Dostupné z: https://meh.dev/php-static-analysis-tools.
30. MIRTES, Ondřej. *PHPStan* [online]. 2016. [cit. 2024-04-18]. Dostupné z: https://phpstan.org.
31. KHORIKOV, Vladimir. *Unit Testing: Principles, Practices, and Patterns*. První. New York: Manning Publications Co., 2020. ISBN 9781617296277.
32. BERGMANN, Sebastian. *PHPUnit* [online]. [cit. 2024-04-18]. Dostupné z: https://phpunit.de/index.html.
33. *Testing* [online]. 2009. [cit. 2024-04-18]. Dostupné z: https://symfony.com/doc/6.4/testing.html.
34. *What is CI/CD?* [online]. [cit. 2024-04-18]. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

Příloha **A**

Seznam použitých zkratek

ADT Abstraktní Datový Typ

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BM Best Match

CI Continuous Integration

CD Continuous Delivery

CSS Cascading Style Sheets

IDF Inverse Document Frequency

IR Information Retrieval

GDPR General Data Protection Regulation

HTTP HyperText Transfer Protocol

HTML HyperText Markup Language

JSON JavaScript Object Notation

ODBC Open Database Connectivity

PDF Portable Document Format

RAM Random-Access Memory

REST Representational state transfer

RTF Rich Text Format

SQL Structured Query Language

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTF Unicode Transformation Format

TF Term Frequency

XML Extensible markup language

Příloha **B**

Návod na spuštění aplikace pomocí Dockeru

Pro úspěšné spuštění aplikace lokálně je potřeba vlastnit soubor s dešifrovacím klíčem pro prostředí *dev*. Tyto soubory **nejsou** verzovány v GitLab repozitáři projektu. Pro obdržení souboru kontaktujte autora práce.

1. Nainstalujte si nejnovější verzi Dockeru[14,15].
2. Spusťte Docker (respektive Docker démona).
3. Otevřete příkazovou řádku ve složce projektu.
4. V příkazové řádce spusťte příkaz `docker compose build --no-cache --pull`.
5. V příkazové řádce spusťte příkaz `docker compose run node yarn install`.
6. V příkazové řádce spusťte příkaz `docker compose up`. Toto spuštění proveďte bez přepínače `-d`, abyste jednoduše zjistili, kdy se aplikace spustí (doběhne příkaz `fetch`).
7. Po několika minutách (cca 10-15) dojde ke spuštění aplikace (a doběhnutí příkazu `fetch`), je však potřeba vytvořit Sphinx indexy v kontejneru `sphinx`. Toho lze docílit příkazem `docker compose run sphinx indexer --all --config /opt/sphinx/conf/sphinx.conf`.
8. Po vytvoření indexů v kontejneru Sphinx lze aplikaci znovu spustit pomocí příkazu `docker compose up [-d]`. Tento příkaz je již možné spustit s přepínačem `-d` pro běh na pozadí. Pokud chcete aplikaci spustit rychleji bez opakovaného běhu příkazu `fetch` (který není znovu nutný), je potřeba ve složce *frankenphp* v souboru `docker-entrypoint.sh` zakomentovat řádek 54 (`php bin/console fetch`) pomocí znaku # a spustit aplikaci pomocí příkazu `docker compose up --build -d`.

[14] https://www.docker.com/products/docker-desktop/
[15] Testováno na Docker Desktop 4.29.0

9. Po chvilce bude aplikace dostupná na adrese localhost[16].
10. Aplikace sama stahuje a aktualizuje data v databázi (kontejner `scheduler`). Z toho důvodu je nutné čas od času spustit příkaz `docker exec sphinx indexer --all --config /opt/sphinx/conf/sphinx.conf --rotate`, který zaktualizuje indexy v kontejneru `sphinx`. Autor doporučuje přidat tento příkaz do cronové tabulky (UNIX), aby se spouštěl automaticky každých 5 minut.
    a) V Linuxu (testováno na Ubuntu) lze použít příkaz `crontab -e` pro otevření cronové tabulky.
    b) Na poslední řádek v cronové tabulce stačí vložit tento řádek: `*/5 * * * * docker exec sphinx indexer --all --config /opt/sphinx/conf/sphinx.conf --rotate`.
11. Pro vypnutí aplikace stačí použít zkratku CTRL + C (pokud neběží na pozadí), nebo příkaz `docker compose down` (pokud běží na pozadí).
12. Po těchto krocích stačí aplikaci spouštět čistě pomocí příkazu `docker compose up [-d]`.

[16] https://localhost/

Příloha C

Obsah přiloženého ZIPu

/
- `readme.txt` ..... stručný popis obsahu média
- `app` ..... adresář se soubory aplikace
  - `config` ..... adresář s konfiguračními soubory frameworku a balíčků
  - `frankenphp` . adresář se skripty a konfigurací frankenphp pro Docker
  - `front` ..... adresář se zdrojovými kódy frontendu
  - `migrations` ..... adresář s databázovými migracemi
  - `public`
    - `room_plans` ..... adresář s plánky místností
  - `sphinx` ..... adresář s potřebnými soubory pro Docker Sphinx
  - `src` ..... adresář se zdrojovými kódy backendu
  - `templates` ..... adresář s index šablonou
  - `tests` ..... adresář s unit testy
  - `README.md` ..... návod na spuštění a používání aplikace
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF