ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

PlagCheckr

Detekce plagiátorství s integrací do Moodle FEL

Miroslav Jarý
Softwarové inženýrství a technologie

Květen 2023
Vedoucí práce: Ing. Miroslav Balík, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jarý** Jméno: **Miroslav** Osobní číslo: **499074**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Studijní obor: **bez oborů**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

PlagCheckr - Detekce plagiátorství s integrací do Moodle FEL

Název bakalářské práce anglicky:

PlagCheckr - Plagiarism detection with Moodle FEE integration

Pokyny pro vypracování:

1. Seznamte se s obecnou problematikou plagiátorství nejen na vysokých školách, a problematikou porovnání podobností prostých textů.
2. Seznamte se s možnostmi integrace detekce plagiátorství do aplikace Moodle.
3. Navrhněte řešení detekce plagiátorství do systému Moodle FEL, s využitím současné infrastruktury. Berte v potaz limity jazyka PHP, ve kterém je systém napsán.
4. Implementujte navrhnuté řešení.
5. Otestujte navrhnuté řešení na sadě testovacích dat, poskytnutých v rámci analýzy [1].

Seznam doporučené literatury:

[1] Ondřej Bureš. RPAPS 2021 - Rozšíření Moodle FEL - 2021 .https://campuscvut.sharepoint.com/:w:/s/FEL-13393/CZM/PHPdev/EcntkJxLY_tJpwCAZLDmqNQB_LU3GpjAe=9DfyZq. 2021.
[2] VMWare Inc. Spring Boot. 2023.https://spring.io/projects/spring-boot.
[3] Moodle Pty Ltd. Plagiarism API — Moodle Developer Resources. 2023.https://moodledev.io/docs/apis/subsystems/plagiarism.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Miroslav Balík, Ph.D. katedra teoretické informatiky FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Miroslav Balík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

V první řadě bych rád poděkoval svému otci za jeho neustálou podporu po celou dobu mého studia. Dále bych rád poděkoval *Centru znalostního managementu FEL ČVUT* za možnost posunout se ve svém oboru, a bez kterého bych neměl možnost skutečně rozvinout své dovednosti. Závěrem bych rád poděkoval nejen svým přátelům, ale i Ing. Miroslavu Balíkovi, Ph.D. za jeho trpělivost a ochotu při vedení této práce.

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.

V Praze dne 24. 05. 2023

.....

Abstrakt / Abstract

Tato bakalářská práce je zaměřená na problém detekce plagiátorství v systému Moodle FEL ČVUT, v podobě externí služby v jazyce Java, běžící na infrastruktuře systému, a zásuvného modulu do aplikace Moodle, který se službou komunikuje.

V první, teoretické části je popsána problematika z hlediska aplikace Moodle, možná místa a způsoby, kde a jak se plagiátorství bránit, včetně dnes již dostupných řešení. Poté je nastíněna teorie ohledně textové analýzy a způsobů vyhledávání plagiátorství.

Následně se práce zabývá návrhem řešení pro použití na Moodle FEL ČVUT, s ohledem na současnou infrastrukturu a možnosti systému. Zároveň jsou zde popsány již existující komerční řešení, jejich výhody a nevýhody.

Ve třetí části práce je popsána implementace služby v podobě API a zásuvného modulu do aplikace Moodle. Zároveň je zde zahrnut popis jejich částí, a objevené problémy.

V poslední části je poté provedeno shrnutí celé práce, včetně výsledků testování a ladění služby.

Klíčová slova: detekce plagiátorství, Moodle, Moodle FEL ČVUT, PHP, Java, Spring Boot, REST

This bachelor thesis is focused on the problem of plagiarism detection in CTU FEE Moodle, in the form of an external Java service running on the system infrastructure and a Moodle plugin that communicates with the service.

The first, theoretical part describes the issue from the perspective of a Moodle application, possible places and ways where and how to prevent plagiarism, including solutions available today. Subsequently, the theory regarding text analysis and ways of searching for plagiarism is outlined.

Afterwards, the thesis discusses the design of a solution for use at CTU FEE Moodle, taking into account the current infrastructure and capabilities of the system. At the same time, existing commercial solutions, their advantages and disadvantages are described.

The third part of the thesis describes the implementation of the service in the form of an external API and a Moodle plugin. At the same time, a description of their parts and encountered problems are included.

Then, in the last part, a summary of the whole thesis is made, including the results of testing and tuning of the service.

Keywords: plagiarism detection, Moodle, CTU FEE Moodle, PHP, Java, Spring Boot, REST

Title translation: PlagCheckr (Plagiarism detection with CTU FEE Moodle integration)

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Tato bakalářská práce se zabývá analýzou, návrhem a řešením problému detekce plagiátorství, konkrétně určenému pro použití v aplikaci *Moodle*[1]. Výsledné řešení, nazvané *PlagCheckr*, je rozděleno na dvě části. První, v podobě externí mikroslužby v jazyce Java (využívající platformu *Spring Boot*[2]), nasazené v infrastruktuře systému *Moodle FEL ČVUT*[3] (instance aplikace *Moodle*, používané na *FEL ČVUT*), je určena pro samotnou výpočetně náročnou analýzu a porovnání textů. Druhou je pak zásuvný modul do již zmíněné aplikace, který s mikroslužbou komunikuje a předává jí práce ke kontrole. Při návrhu a implementaci řešení byl brán zřetel na možnost jeho provozu nejen na *FEL ČVUT*, ale i na jiných fakultách i školách v České republice, které již v současné době aplikaci *Moodle* využívají.

Práce staví na analýze Ondřeje Bureše[4], provedené v rámci *Centra znalostního managementu*, a uskutečněné v rámci projektů *RPAPS*[1] na *FEL ČVUT* v roce 2021. Analýza byla vyhotovena na základě podkladů a diskuze s Ing. Markem Brothánkem, Ph.D.

Je nutné zdůraznit, že výsledné řešení není univerzálním automatem, který dokáže odhalit všechny zcizené, okopírované, či jinak přisvojené práce, a jejich autory automaticky penalizovat. *PlagCheckr* je pouze nástrojem pro vyučující, který by jim měl usnadnit tyto plagiáty rychle odhalit. Konečné rozhodnutí, a případné potrestání, je vždy přenecháno vyučujícímu.

1.1 Předmluva

S nástupem distanční výuky v průběhu roku 2020 způsobeným pandemií *COVID-19* se výrazným způsobem zvýšil zájem o výuku v online prostoru, a s ní související automatizaci hodnocení studentů. Současně se však do online prostoru přesunuly i pokusy studentů hodnocení ovlivnit ve svůj prospěch.

Problémem podvádění v online prostoru se již od začátku pandemie zabývala řada odborníků i firem. Jedním z příkladů je společnost *SCIO*, pořádající Národní srovnávací zkoušky[6], které některé vysoké školy uznávají jako ekvivalent přijímacích zkoušek do jejich studií. Řešení spočívalo v aktivním monitorování všech zkoušených pomocí aplikace *ProctorTrack*[7], běžící na počítači zkoušeného studenta – této aplikaci musel být udělen přístup nejen k webkameře, ale celému počítači, což způsobilo řadu kontroverzí[8], a shromážděná data byla i terčem útoků[9]. Takovýto přístup k osobním údajům je proti zásadám a nařízením Českého vysokého učení technického v Praze[10], a je takřka nepoužitelný v běžné výuce.

Alternativním přístupem k řešení tohoto problému může být kontrola pouze samotných řešení, odevzdaných studentem – ta sice může být méně účinná, zato však nevyžaduje dodatečné monitorování autorů, ani nijak nenarušuje jejich soukromí. Při této

[1] *Rozvojové projekty akademických pracovníků a studentů* je každoroční vnitřní soutěž v rámci *Institucionálního programu ČVUT*[5].

kontrole se například můžeme soustředit na vyhledávání tzv. *plagiátů* – úmyslně zkopírovaného cizího textu. Plagiátem může být myšleno jak opsání závěrečné práce, tak i znovupoužití vypracovaného protokolu laboratorní práce jiného studenta z předchozích let.

Tímto přístupem se již řídí ochrana, implementována do systému *BRUTE*[11] (Balík pro Rezervace, Upload, Testování a Evaluaci studentských úloh) – jde o jeden ze dvou systémů pro podporu výuky, oficiálně používaných na *FEL ČVUT*. Tato ochrana byla předmětem bakalářské práce Tomáše Votroubka[12].

V druhém systému, *Moodle FEL ČVUT*[3], který je založen na open-source aplikaci *Moodle*[1], však dosud podobná ochrana chybí. Důvodů lze nalézt několik.

- **Architektura aplikace** – oproti systému *BRUTE* je *Moodle* aplikace třetí strany – znalost kódu jejími správci je tak výrazně nižší. Moodle je zároveň mnohem rozsáhlejší aplikace – ve verzi 3.11 její kód čítá téměř 4 miliony řádků[13].
- **Rychlost** – aplikace je napsána v interpretovaném programovacím jazyce *PHP*, který byl vyvinut pro snadnou tvorbu dynamických webových stránek, a nikoliv pro složité výpočty. To se projevuje na jeho rychlosti, a oproti jiným, kompilovaným jazykům výrazně zaostává[14].
- **Stáří** – první verze aplikace byla vydána v roce 2002[15], a většina skriptů, které jsou součástí jádra, nebyly dosud rozšířeny o nová paradigmata či funkce jazyka PHP. Z dnešního hlediska se tak mohou některá řešení ohledně rozšíření systému zdát jako příliš složitá.

Na základě nejen zvýšeného používání systému, ale i požadavků vyučujících kateder fakulty, bylo však rozhodnuto, že zavedení ochrany do *Moodle FEL ČVUT* je jednou z hlavních priorit jeho dalšího rozvoje. Tato práce si tak klade za cíl navrhnout alespoň částečné řešení problému plagiátorství, následně ho zrealizovat a uvést do provozu.

1.2 Struktura práce

První část této práce se v kapitole 2 zabývá teorií ohledně problému plagiátorství. Je zde uveden nejen úvod do této problematiky, ale i současný stav jejího řešení v online i offline prostoru na *FEL ČVUT*. Dále je v této kapitole podrobně rozebrána teorie okolo analýzy a porovnání podobností dvou textů; jeho možnosti, výhody i nevýhody.

Ve druhé části, tj. v kapitole 3, se následně práce zaobírá samotným návrhem řešení tohoto problému pro aplikaci *Moodle* – výběrem vhodných typů aktivit pro kontrolu, a následně i architekturou řešení. Závěrem kapitoly jsou pak zmíněny příklady aplikací třetích stran rovněž se zabývající tímto problémem, spolu s jejich stručným popisem a charakteristikou.

Třetí část v kapitole 4 podrobně rozebírá implementaci řešení systému pro detekci plagiátorství. Nejprve jsou zde rozebrány možnosti použití technologií; od datových formátů až po samotný jazyk implementace služby. Následující dvě sekce se zabývají popisem implementace externí mikroslužby, a zásuvného modulu do aplikace Moodle, který s touto službou komunikuje. V poslední sekci jsou poté detailně rozebrány problémy, které byly v průběhu implementace odhaleny.

Práci pak uzavírá poslední část, řešící následné nasazení a ladění řešení na základě dat poskytnutých vyučujícími (dále rozebráno v kapitole 5), a poté i celkové shrnutí práce (v kapitole 6).

Kapitola 2
Teorie

2.1 Plagiátorství

Jak již bylo řečeno, tato práce se zabývá částečným řešením problému tzv. *plagiátorství*. Na internetových stránkách Masarykovy univerzity v Brně se nachází definice[16] plagiátorství, kterou se bude řídit tato práce:

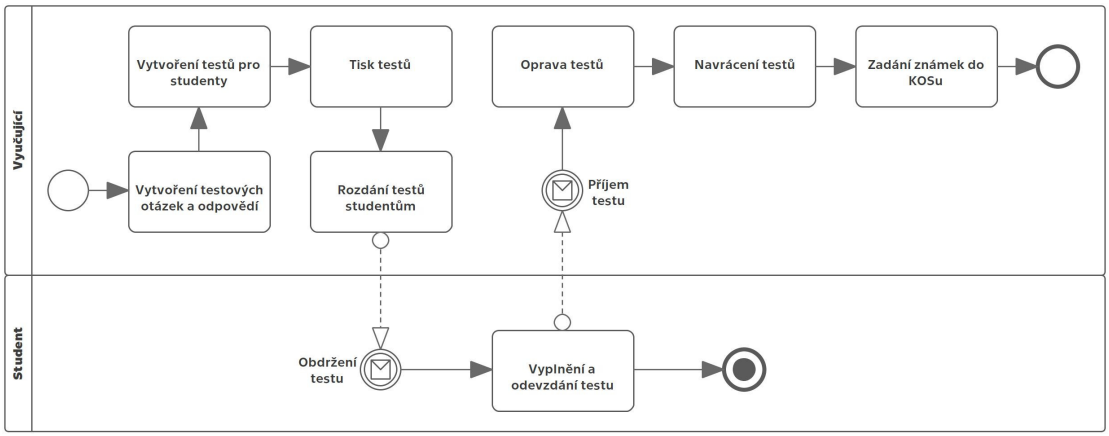
> *„Za plagiátorství lze považovat úmyslné kopírování cizího textu a jeho vydávání za vlastní, nedbalé nebo nepřesné citování použité literatury, opomenutí citace (byť neúmyslné) některého využitého zdroje."*

S plagiátorstvím je možné se setkat v nejrůznějších podobách – od opisování v písemných testech, přes kopírování závěrečných prací, až po znovupoužívání zdrojových kódů programů z internetu či od jiných studentů. Plagiátorství však není pouze problém dnešní doby – řada politiků či známých osobností čelí obvinění z opsání svých závěrečných prací[17] nebo dokonce i uměleckých děl[18], či byli z plagiátorství již v minulosti usvědčeni[19].

Nástup moderních informačních a komunikačních technologií však tuto činnost výrazně zjednodušil. Technologie nám však zároveň mohou pomoci tyto prohřešky lépe zachytávat a trestat.

2.1.1 Prevence na FEL ČVUT

Proces offline zkoušení je vymodelován na obrázku 2.1 – z něj je patrné, že velká část úkonů přímo závisí na samotném vyučujícím, od přípravy testu, po kontrolu průběhu, až po samotné vyhodnocení.



Obrázek 2.1. Procesní diagram offline zkoušení studentů.

Nejčastějším způsobem, jak vyučující předcházejí opisování či plagiátorství, je aktivní monitorování studentů – např. při psaní zkouškových testů. Studenti jsou tak pod neustálým dohledem; v některých případech to však má pouze psychologický efekt. Pokud

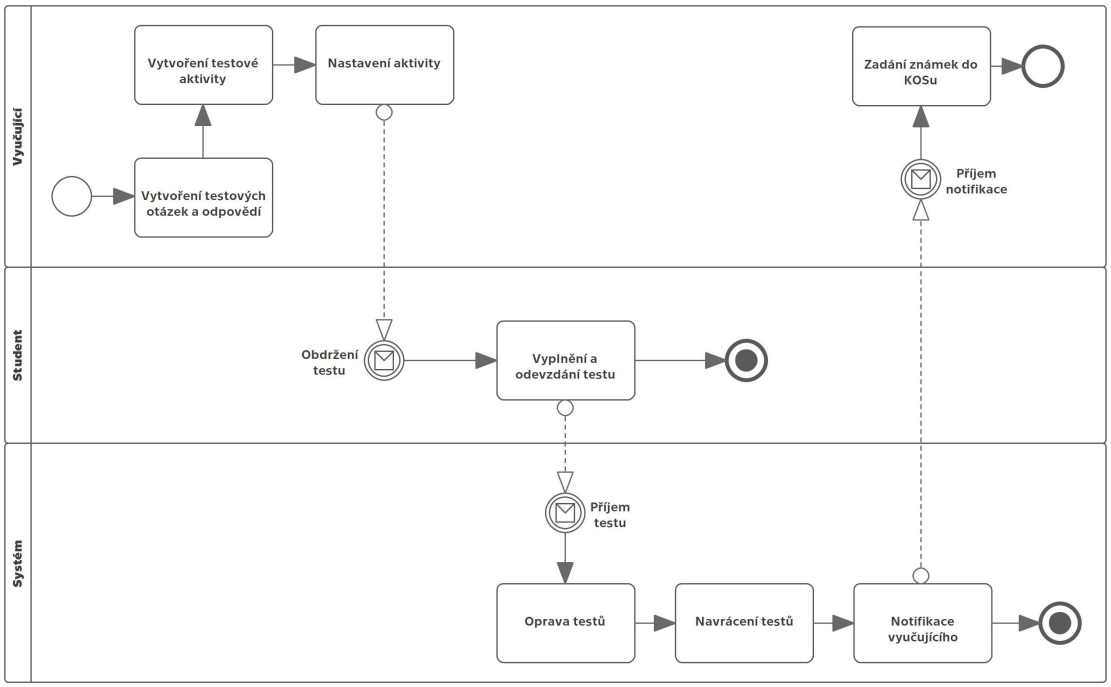
se zkouška koná ve velké učebně, či je na termín přihlášeno mnoho studentů, zkoušející s největší pravděpodobností nedokáže ohlídat všechny zkoušené.

V případě praktických zkoušek, které se většinou skládají v počítačových laboratořích, se přístupy liší. Někteří zkoušející odpojují počítače od internetové sítě, jiní naopak používání internetu povolují, avšak studentům zakazují vzájemnou komunikaci. Většinou pak studenty dále zásadně nehlídají.

Co se týče domácí přípravy, zde je v praktických úlohách spojených s programováním používán převážně již zmíněný systém *BRUTE*[11], který dokáže kopírovaná řešení do jisté míry odhalit. Naopak v písemných testech, esejích či slohových cvičeních stále převažuje manuální kontrola vyučujícím.

2.1.2 Prevence v aplikaci Moodle

Proces online zkoušení je zachycen na obrázku 2.2 – ten se do jisté míry podobá procesu na obrázku 2.1, avšak již na první pohled je vidět, že značná část práce, převážně kontrolní části, je přesunuta na systém, ve kterém se test koná.



Obrázek 2.2. Procesní diagram online zkoušení studentů.

Na příkladu aplikace *Moodle* jsou studentům úkoly zadávány pomocí tzv. *aktivit* (které jsou následně hodnoceny). Ty mohou být nejrůznějšího typu, od kvízů a dotazníků, přes nahrávání souborů, až po jednoduché zaslání prostého textu. Systém nabízí ve svém základu několik způsobů, kterými se snaží zabránit v podvádění a opisování při tomto odevzdávání, a to jak aktivně, tak i pasivně.

- **Safe Exam Browser**[20] – jedná se o speciální webový prohlížeč s integrací do aplikace Moodle, který brání používání jiných stránek či aplikací v průběhu testu. Vyučující mohou v aktivitě nastavit, že její splnění může být provedeno pouze v tomto prohlížeči. Funkce se týká pouze aktivity *Test*.

- **Náhodné generování testů** – obsah testů (použité otázky a jejich pořadí) je vytvořen až po spuštění pokusu studentem, a je tak zcela náhodný[1], čímž je opisování značně ztíženo.
- **Omezení odevzdávání** – vyučujícímu umožňuje nastavit, aby bylo odevzdávání omezeno podle různých parametrů:
  - **podle IP adresy**; odevzdávání tak může být např. podle rozdělení IP rozsahů na *FEL ČVUT* umožněno pouze z počítačů v laboratořích katedry daného předmětu,
  - **podle data a času odevzdání**; odevzdávání tak může být umožněno pouze v určitý čas a jen po určitou dobu.

Dále aplikace nabízí tzv. *Plagiarism API*[22], které umožňuje napojit odevzdaná řešení vybraných aktivit na externí služby určené přímo pro detekci plagiátorství, jako jsou *Ouriginal*[23] či *Turnitin*[24]. Tato komerční řešení jsou podrobněji rozebrána v sekci 3.3.

2.2 Analýza a porovnání textů

Algoritmus vyjádření porovnání dvou textů lze rozdělit na dva kroky.

- **Předzpracování** – kdy je text upraven do lépe analyzovatelné podoby. Příkladem může být tzv. *lemmatizace*, kde jsou slova osekána na své kořenové základy, a jsou tak zbavena např. skloňování či časování; např. ze slov *hrneček* a *hrnečkách* vznikne shodný tvar *hrneč*. Bez předzpracování textu by se mohlo stát, že totožný text, jednou psán v *ich-formě* (tj. v první osobě) a podruhé v *er-formě* (tj. v třetí osobě), by analýza nezachytla jako plagiát, i když jím jistě je.
- **Porovnání** – kdy je podobnost dvou textů datově vyjádřena. Může jít o jednoduché procentuální vyjádření, ale i o složité mapování jednoho textu na druhý.

Jednotlivé kroky jsou dále rozebrány samostatně v podsekcích 2.2.1, resp. 2.2.2 a 2.2.3.

2.2.1 Předzpracování textu

První způsob předzpracování textu je převod na čistý text; tj. zbavení se nejen jakéhokoliv formátování a stylů dokumentu, ale i obrázků, tabulek apod. Tímto krokem bohužel ztrácíme část informace, např. u laboratorních protokolů, kde tabulky tvoří podstatnou část informace. Zároveň jsou však tabulky poměrně jednoduché v odhalení plagiátorství – vyučující pohledem zjistí opsané hodnoty. Problematické může být také případné špatné kódování diakritiky v PDF souborech (dále rozebráno v sekci 4.4).

Dalším způsobem je tzv. *tokenizace*. V tomto kontextu jde o rozdělení textového celku na menší části, zde na jednotlivá slova. Rovněž zde dochází k odstranění interpunkce či převodu velkých písmen na malá. Součástí tokenizace může být i výše zmíněná *lemmatizace* – ta je taktéž důležitá, obzvláště v českém, či jiném flektivním jazyku (obsahujícím skloňování, časování či jiné podobné modifikace)[25].

V případě potřeby urychlení analýzy lze použít odstranění tzv. *stopslov*[26] – slov, která nepřidávají textu téměř žádnou informaci. Příkladem mohou být v českém jazyce předložky a spojky, či v angličtině určité a neurčité členy. Tato metoda se používá i kvůli zabránění falešným shodám.

[1] Přesněji pseudonáhodný, jelikož procesy v počítačích jsou deterministické[21]. Vskutku náhodná generace by vyžadovala pozorování např. termálních dějů či radioaktivního rozpadu.

Pro vyšší přesnost analýzy, a tím pádem i odhalení více propracovaných plagiátů, lze využít rozpoznávání synonym. Pro účely lingvistických zpracování textů vznikla na základě lexikální databáze *WordNet*[27] univerzity v Princetonu její česká verze s názvem *Czech WordNet*[28], obsahující okolo 48 000 literálů. V roce 2011 byla tato česká verze v rámci diplomové práce Marka Blahuše[29] neoficiálně rozšířena o literály přeložené z anglického WordNetu. Tato metoda je však vhodná převážně pro delší texty, jakými mohou být například vysokoškolské závěrečné práce.

2.2.2 Porovnání textu vyjádřené hodnotou

Vyjádření porovnání textu hodnotou slouží převážně k rychlému úsudku, s jakou pravděpodobností se jedná o plagiát. To může být užitečné u dlouhých dokumentů, kde můžeme snadno přehlédnout jednu či dvě okopírované stránky. Zároveň, pokud si dal autor práci, a např. vmísil části cizího textu do vlastního, nemusí být na první pohled plagiát rozpoznatelný.

Způsobů výpočtu číselného vyjádření podobností lze nalézt mnoho. Pro většinu z nich jsou pro dva porovnávané texty A a B vytvořeny pomocné vektory $\vec{A}$ a $\vec{B}$, obsahující informaci o četnosti výskytu jednotlivých slov (nebo pouze informaci, jestli se dané slovo v textu nachází). Pokud si tedy např. zvolíme jako první text *nové auto a nové kalhoty*, a jako druhý text *nové modré kalhoty*, pomocné vektory by vypadaly následovně:

$$\vec{A} = [2, 1, 1, 1, 0];$$
$$\vec{B} = [1, 0, 0, 1, 1];$$

kde pozice odpovídají slovům [*nové, auto, a, kalhoty, modré*]. S těmito vektory poté můžeme dále počítat pomocí vícero metrik.

Jednou z nejpoužívanějších je *Euklidovská vzdálenost*. V matematice se pod pojmem vzdálenost nejčastěji myslí tato metrika. Pro n-rozměrné vektory ji lze vyjádřit pomocí vzorce (1):

$$dist(\vec{A}, \vec{B}) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}. \tag{1}$$

Další často používanou vzdáleností je *Manhattanská vzdálenost*, nebo také *Taxíkářská vzdálenost*. Tyto názvy vznikly díky rozložení ulic v Newyorské čtvrti Manhattan, které spolu až na výjimky vždy svírají pravý úhel. Taxikáři při výpočtu taxy, kterou musí zákazník zaplatit, tak neberou v úvahu nejkratší vzdálenost (i tak si můžeme představit *Euklidovskou vzdálenost*), ale přesnou trasu, kterou ve voze ujedou. Tuto vzdálenost lze obecně pro n-rozměrné vektory vyjádřit vzorcem (2):

$$dist(\vec{A}, \vec{B}) = \sum_{i=1}^{n}|A_i - B_i|. \tag{2}$$

Porovnání obou výše zmíněných vzdáleností v dvourozměrném prostoru můžeme vidět na obrázku 2.3; u Euklidovské vzdálenosti je možná vždy jen jedna cesta – ta nejkratší. Naopak cest určujících Manhattanskou vzdálenost je vícero.

Obrázek 2.3. Grafické porovnání *Manhattanské vzdálenosti* (zeleně) a *Euklidovské vzdálenosti* (červeně). Mapa převzata z `https://snazzymaps.com`.

Oba tyto výpočty jsou ve skutečnosti konkrétní příklady *Minkowského vzdálenosti*, pojmenované po německém matematikovi Hermannu Minkowském. Tyto vzdálenosti mají určený řád p, který následně užíváme ve vzorci (3) pro n-rozměrné vektory:

$$dist(\vec{A}, \vec{B}) = (\sum_{i=1}^{n} |A_i - B_i|^p)^{\frac{1}{p}}. \tag{3}$$

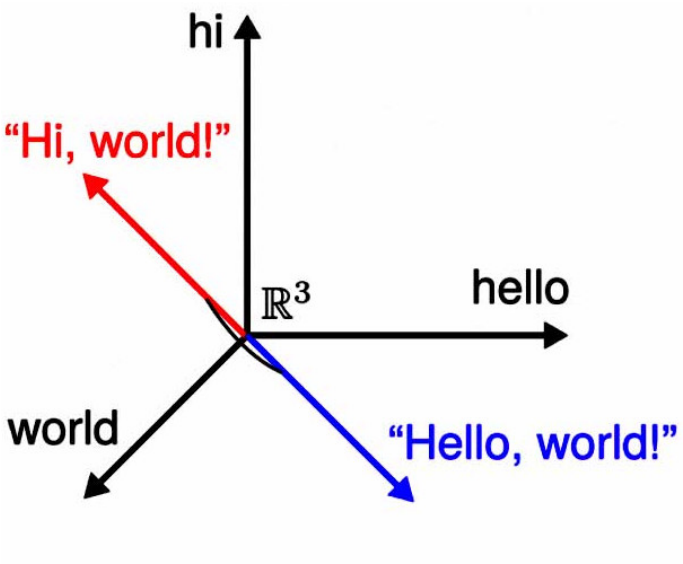
Z této rovnice můžeme snadno odvodit, že *Manhattanská vzdálenost* je ve skutečnosti *Minkowského vzdálenost* řádu $p = 1$ a *Euklidovská vzdálenost* je ve skutečnosti *Minkowského vzdálenost* řádu $p = 2$.

Vzhledem k tomu, že se však jedná o vzdálenosti, je výsledná hodnota vždy závislá na velikosti vektorů (počtu použitých slov), což je pro účely této práce nevhodné – zajímá nás pouze informace o vzájemném výskytu těchto slov, kterou lze matematicky vyjádřit jako *směr* vektoru. Tomuto problému lze předejít tzv. *normalizací* vektorů – všechny složky vektoru vydělíme jeho současnou velikostí.

Důležité je zmínit, že existují i tzv. *editační vzdálenosti* – příkladem může být známá *Levenštejnova vzdálenost*[30]. Ty se však zabývají lehce odlišným problémem – díky nim lze spočítat opravdovou vzdálenost dvou textů (nikoliv vzdálenost množin vyskytujících se slov), neboli jak těžké je přetvořit jeden text na druhý záměnou písmen; např. slova *oko* a *pokoj* mají *Levenštejnovu vzdálenost* rovnou 2 – stačí ze slova *pokoj* odstranit první a poslední písmeno. Tato metodika se však pro účel analýzy nehodí – nebere totiž v úvahu ani kontext textu, ani logickou podobnost, která je v analyzovaných textech důležitá.

Kromě výpočtu vzdálenosti můžeme číselně vyjádřit podobnost dvou textů A a B i dalšími metrikami.

- **Kosinova podobnost** – výsledkem je kosinus mezi dvěma dříve zmíněnými vektory $\vec{A}$ a $\vec{B}$. Její grafické vyjádření můžeme spatřit na obrázku 2.4.
- **Symetrická metrika podobnosti** – definována jako podíl počtu unikátních slov obsažených v obou dokumentech zároveň a počtu unikátních slov obsažených alespoň v jednom ze dvou dokumentů.
- **Asymetrická metrika podobnosti** – definována jako podíl počtu unikátních slov obsažených v obou dokumentech zároveň a počtu unikátních slov prvního dokumentu; na rozdíl od předchozích metrik tak zde záleží na pořadí dokumentů. Jde tedy o vyjádření, jak je jeden dokument obsažen v druhém.
- **Symetricky asymetrická metrika podobnosti** – jedná se o pouhou abstrakci; pomocí asymetrické metriky spočteme obsažení jednoho textu v druhém a druhého textu v prvním, následně vrátíme maximum těchto dvou hodnot.



Obrázek 2.4. Grafická vizualizace *kosinové podobnosti.*

U všech těchto metrik pak výsledek dosahuje hodnot v rozmezí 0 až 1, kde 0 je naprostá neshoda (texty tedy nemají ani jedno shodné slovo), a 1 je naprostá shoda (texty používají naprosto stejná slova).

2.2.3 Porovnání textu vyjádřené mapováním

Tento způsob mapování může být užitečný převážně pro pozdější manuální kontrolu řešení. Cílem je mapování jednoho či více úseků prvního textu A na části druhého textu B, na základě určité podobnosti. Jednoduchým praktickým příkladem může být Unixový příkaz `diff`, který dokáže zobrazit rozdíl mezi dvěma soubory. Zde se však musí jednat o naprostou[1] shodu. Příklad použití tohoto příkazu lze vidět na obrázku 2.5.

```
root@Aspire-5 ~$ diff --color=auto first.txt second.txt
2c2
< Tento text je v prvním souboru.
---
> A tento text je v druhém souboru.
\ No newline at end of file
```

[1] Výjimkou mohou být mezery, řádkování či jiné znaky, které textu nepřidávají téměř žádnou hodnotu.

Obrázek 2.5. Ukázka použití příkazu `diff`.

V případě analýzy však tento jednoduchý příkaz nestačí – je zapotřebí vlastní řešení. V první řadě musí umět vytvářet vazby mezi těmito texty – to příkaz `diff` neumí. Následně je důležité, aby byla možná reference na původní text; koncovému uživateli nebude zobrazen zpracovaný text, nýbrž původní PDF. Je tak důležité, aby aplikace byla schopná převést analyzovaný text zpět na původní, alespoň co se týče jeho pozice v souboru.

K tomuto účelu lze použit převod textů na tzv. n-gramy; zjednodušeně řečeno jde o množinu po sobě jdoucích slov o n prvcích, většinou se za těmito účely používají hodnoty $n = 2$ nebo $n = 3$. Pokud tedy máme např. text *staré šaty a nové boty*, a z něj uděláme n-gram pro hodnotu $n = 2$, výsledné prvky budou vypadat takto:

$$\{staré, šaty\}, \{šaty, a\}, \{a, nové\}, \{nové, boty\}$$

Tato technika jde ruku v ruce s výše zmíněnou *tokenizací* - n-gramy jsou pouze vícero tokenů dohromady. Zároveň, pokud při přípravě textu budeme počítat s potřebou zpětného mapování na původní text, je možné tuto techniku bez větších komplikací použít.

Jedním možným problémem může být určení počtu či velikosti n-gramů, aby označený text byl opravdu plagiátorský, a aby řešení neoznačovalo text, který je shodný pouze náhodou. Tomuto problému se více věnuje kapitola 5, která se zabývá testovacím provozem řešení.

Kapitola 3
Návrh řešení

3.1 Výběr aktivit k analýze

Jak již bylo zmíněno v podsekci 2.1.2, studenti v aplikaci Moodle plní úkoly odevzdáváním tzv. aktivit. Využití jednotlivých druhů (do kterých mohou studenti odevzdávat řešení či jinak do nich přispívat) dostupných v systému *Moodle FEL ČVUT* lze vidět v tabulce 3.1.

Název aktivity	Počet instancí
Úkol	13587
Fórum	7672
Test	6501
Balíček SCORM	874
Přednáška	636
Anketa	634
Docházka	361
Dotazník	220
Workshop	41
Databáze	22
LiveVoting	15
Offline test	9
H5P	2

Tabulka 3.1. Využití jednotlivých aktivit ve všech 5893 kurzech systému *Moodle FEL ČVUT*, ke dni 26. 12. 2022. Data převzata z administračního přehledu stránky

Z dat lze vyčíst, že nejčastěji jsou využívány tři druhy aktivit: *Úkol*[31] (angl. *Assignment*), *Fórum*[32] (angl. *Forum*) a *Test*[33] (angl. *Quiz*). Další druhy aktivit se vyskytují řádově méně, či se pro účely ochrany proti plagiátorství nehodí.

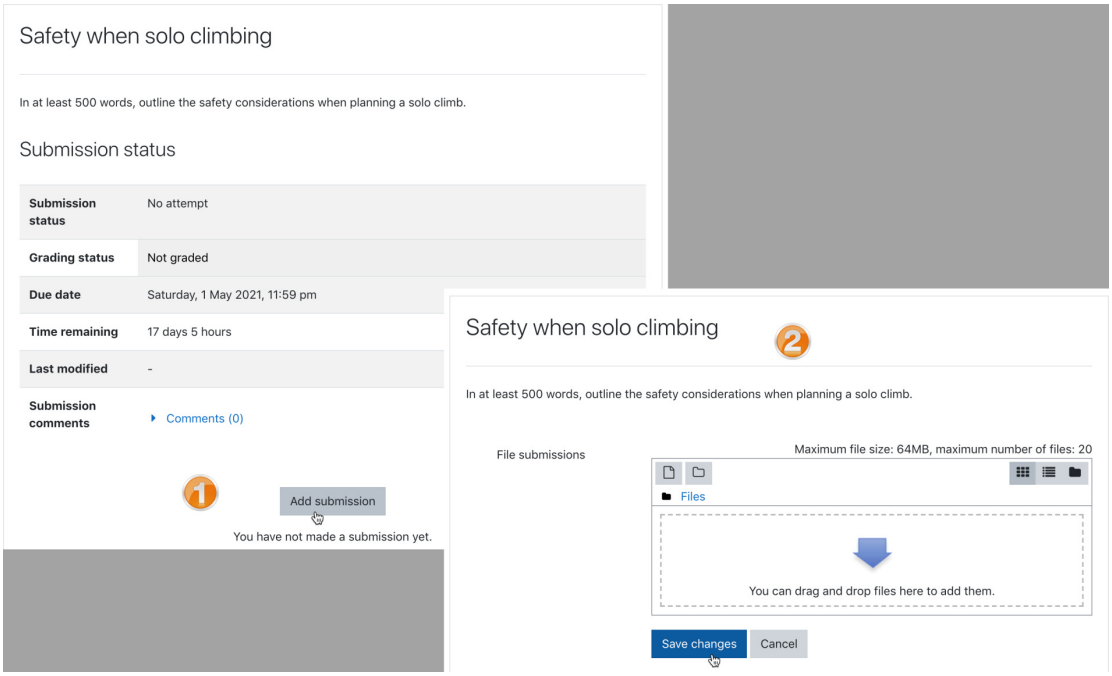
I z tohoto důvodu byla z dalšího zkoumání vynechána aktivita *Fórum*. Ta totiž slouží spíše k diskuzi mezi studenty a/nebo vyučujícími, a odevzdávání samotných prací studentů zde neprobíhá. Dalším důvodem pro vynechání této aktivity je její nízká míra používání v kurzech – dle statistik v tabulce 3.1 je to sice aktivita s druhým nejvyšším počtem instancí v kurzech, avšak toto vysoké číslo je způsobeno jejím automatickým přidáním do každého nově vytvořeného kurzu; reálně aktivitu využívá spíše nižší procento vyučujících.

Nejvhodnější aktivity pro uplatnění ochrany proti plagiátorství by tak dle dat byly *Úkol* a *Test*. Jejich detailní charakteristika, případy použití a konečný úsudek jsou předmětem podsekcí 3.1.1 a 3.1.2.

3.1.1 Aktivita Úkol

Úkol[31] (angl. *Assignment*) umožňuje studentům zapsaným v daném kurzu odevzdávat vypracovaná řešení, ať již v podobě prostého textu (za použití integrovaného textového

editoru), PDF souboru či jiného média. Řešení je následně vyučujícím ohodnoceno, a případně je studentovi předán i slovní komentář s poznámkami. Studenti mohou dle daného nastavení odevzdávat řešení jak samostatně, tak i ve skupinách. Příklad této aktivity je zachycen na obrázku 3.1; na první obrazovce se nachází přehled úkolu s informacemi o zadání, na druhé pak formulář, pomocí kterého student odesílá své řešení.



Obrázek 3.1. Příklad aktivity *Úkol* z pohledu studenta. Převzato z [31].

Z hlediska další analýzy se jako nejvhodnější typ souborů jeví PDF soubory. Ty mohou obsahovat nejrůznější obsah:

- **Prostý text** – jedná se často o hlavní obsah souboru; příkladem mohou být eseje, slohová cvičení apod.
- **Obrázky** – ty nejčastěji slouží jako pouhá ilustrace či doplnění k prostému textu[1]. Obrázky tak nejsou pro analýzu nijak zvlášť přínosné.
- **Ofocený text** – zde se také jedná o obrázek, jehož obsahem je však např. ručně psaný text. Ten má obsahově stejnou hodnotu jako text prostý.
- **Vzorce** – stejně jako obrázky slouží spíše jako doplnění textu samotného. Zároveň často bývají neměnné a stejné pro všechny studenty[2]. Jejich zpracování by tak analýze přinášelo pouze minimální hodnotu.

Z výčtu možného obsahu je k analýze nejvhodnější prostý i ofocený text, jelikož představují stejnou informaci – liší se pouze v její formě. Zároveň text v těchto souborech bývá dostatečně dlouhý na průkaznost celé analýzy.

Problém s ofoceným textem může být jeho převedení do prostého textu – analyzované informace jsou totiž skryty v použitých slovech, a nikoliv písmu autora. Převod psaného textu do prosté formy (tzv. *OCR – Optical Character Recognition*) je však téma hodno

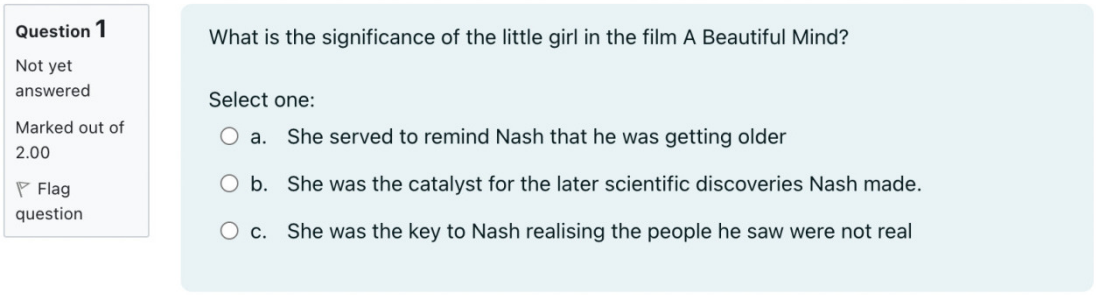
[1] Výjimkou mohou být např. přímo zadané grafické úlohy, jako např. *nakreslete graf funkce...* Jejich výskyt v úlohách je však výrazně nižší, a případná implementace by přesahovala rozsah této práce.

[2] Mohou se však vyskytnout případy, kdy například dva či více studentů udělají v postupu nebo výsledném vzorci stejnou chybu – takovéto chyby jsou však mnohem snáž spatřitelné pouhým okem, oproti např. porovnání prostého textu.

samostatné práci. V případě budoucího rozšíření výsledného řešení je však tato funkce pravděpodobným kandidátem.

3.1.2 Aktivita Test

Aktivita *Test*[33] (angl. *Quiz*) více připomíná např. zkoušku či zápočtový test – studentům jsou kladeny otázky, na které musí odpovědět výběrem jedné či více předem připravených odpovědí (tzv. *uzavřené otázky*), nebo sepsáním odpovědi vlastními slovy (tzv. *otevřené otázky*). Příklad takové otázky (konkrétně *výběru z možných odpovědí*) lze vidět na obrázku 3.2.



Obrázek 3.2. Příklad testové otázky z pohledu studenta. Převzato z [33].

Jednotlivé otázky v kurzu vyučující vytváří v tzv. *Bance úloh* – díky ní mohou být použity ve vícero testech v rámci kurzu. Statistika použití jednotlivých druhů otázek v *Moodle FEL ČVUT* (a jejich rozlišení na otevřené a uzavřené) je uvedena v tabulce 3.2.

Typ otázky	otevřená/uzavřená	Počet instancí
Výběr z možných odpovědí	uzavřená	98616
Numerická úloha	otevřená	80272
Krátká tvořená odpověď	otevřená	18382
Dlouhá tvořená odpověď	otevřená	9746
Pravda/Nepravda	uzavřená	1347
Vypočítávaná úloha	otevřená	1264
Výběr chybějících slov	uzavřená	1179
Přiřazování	uzavřená	749
Přetahování do textu	uzavřená	472
Přetahování ukazatelů umístění	uzavřená	146
Vypočítávaná úloha s více možnostmi	uzavřená	78
Jednoduchá vypočítávaná úloha	otevřená	71
Přetahování do obrázku	uzavřená	68

Tabulka 3.2. Využití jednotlivých typů otázek ve všech 5893 kurzech systému *Moodle FEL ČVUT*, ke dni 26. 12. 2022. Data převzata z administračního přehledu stránky.

Uzavřené otázky jsou vzhledem k nedostatku informací obsažených v odpovědi (pouze vybraná možnost) krajně nevhodné pro zpětnou analýzu – pokud student odpoví správně, nemůžeme rozlišit, zda-li tak učinil na základě vlastních znalostí, nebo po debatě se spolužákem. Avšak pokud např. tři studeni vyberou dvě stejné, špatné odpovědi z pěti možných, můžeme je podezřívat, že se na odpovědích dohodli. Tyto

případy jsou však velice vzácné, a dá se jim snadněji předejít pomocí prevence, která již byla dříve rozebrána.

Otevřené otázky jsou k analýze mnohem vhodnější, jelikož obsahují mnohem více informací. Jak bylo zmíněno v tabulce 3.2, patří mezi ně:

- **numerická úloha** – student odesílá číselnou hodnotu, např. výsledek výrazu;
- **krátká tvořená odpověď** – prostý text o maximálně jedné či dvou větách; vyučující při vytváření otázky specifikuje klíčová slova, která jsou považována za správná, systém je pak v textu hledá;
- **dlouhá tvořená odpověď** – prostý text libovolné délky; zde neprobíhá žádná automatická kontrola, a vyučující musí provést manuální hodnocení po odevzdání;
- **vypočítávaná úloha** – odpovědí je číselná hodnota jako u numerické úlohy, zde se však správný výsledek počítá při generování, a úloha může být parametrizována.

Jako v případě uzavřených otázek se zde setkáváme s problémem nedostatku informací pro správný úsudek, zda-li se jedná o plagiát či ne. Jedinou vyjimkou je dlouhá otevřená odpověď, kde je užití podobné jako v aktivitě *Úkol* z podsekce 3.1.1, odpovědi však zde i tak bývají výrazně kratší.

Po celkovém prozkoumání aktivity typu *Test* bylo dospěno k názoru, že jediný typ otázek, alespoň částečně vhodný k zpětné analýze odevzdaných řešení, je typ *dlouhá tvořená odpověď*. Vzhledem k její míře použití (viz tabulka 3.2), a celkové neefektivitě implementace ochrany pouze pro jednu část aktivity bylo rozhodnuto nadále se zabývat řešením pouze pro dříve zmíněnou aktivitu *Úkol*. K podobnému závěru dospěla i analýza Ondřeje Bureše[4], na které tato práce staví.

3.2 Architektura

Vzhledem k výpočetní náročnosti celého procesu analýzy je důležitá řádná příprava architektury řešení. Jak již bylo řečeno v úvodu, jazyk PHP, ve kterém je aplikace Moodle napsána, patří mezi jedny z pomalejších. Implementovat tak celý proces analýzy přímo jako modul aplikace by bylo nejen zdlouhavé a neefektivní, ale celá analýza by mohla výkonnostně ovlivnit i zbytek systému, který je již nyní z dříve uvedených důvodů značně pomalý.

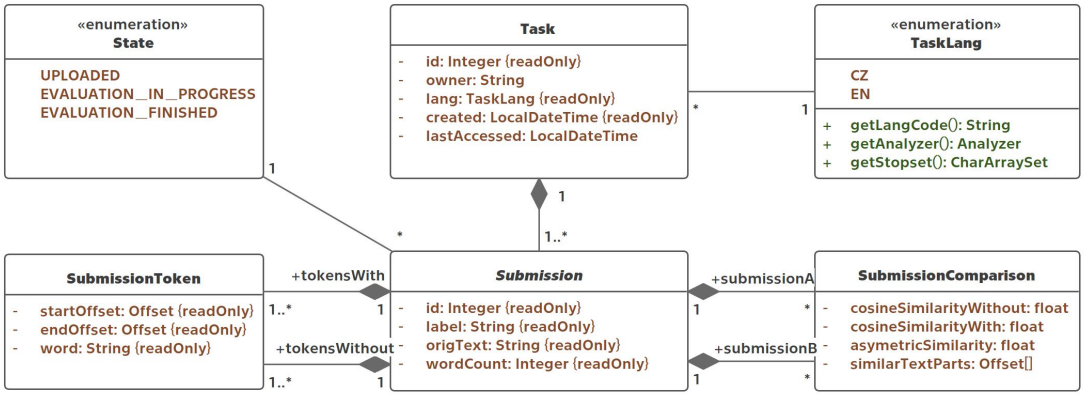
Ideálním řešením by tak bylo rozdělení na dvě samostatné části – první, externí služba, zajišťující samotnou analýzu textů, a druhá, ve formě zásuvného modulu do aplikace Moodle, která by s externí službou pouze komunikovala, a reprezentovala její výsledky.

3.2.1 Datový model externí služby

V sekci 2.2 byl vysvětlen průběh celé analýzy. Tomu musí být přizpůsoben i datový model služby. Její součástí nebude relační či jiná databáze pro ukládání dat – ta se tak budou uchovávat pouze v paměti. Analýza by totiž měla být dostatečně rychlá, a vyhotovení výsledků by nemělo zabrat více než několik vteřin. Při návrhu služby tak bude důležité dbát na efektivní práci s pamětí, a postupné mazání již zpracovaných požadavků po určité době; případně pak dovolit uživatelům, aby sami mohli požadavky mazat poté, co obdrží data o nahraných souborech.

Návrh datového modelu je zachycen na obrázku 3.3. Uživatel služby bude zakládat tzv. *Tasky* – tj. požadavky na vzájemné porovnání sady PDF souborů. Obsah každého souboru je uložen v třídě *Submission*, včetně informace, zda-li se jedná o soubor přímo od studenta (který je třeba analyzovat), nebo jestli jde o *kontrolní vzorek*. Těmito vzorky

má uživatel možnost obohatit porovnání – může jít např. o řešení studentů z minulých let. Tyto vzorky nebudou porovnávány mezi sebou, ale pouze se soubory přímo od studentů.

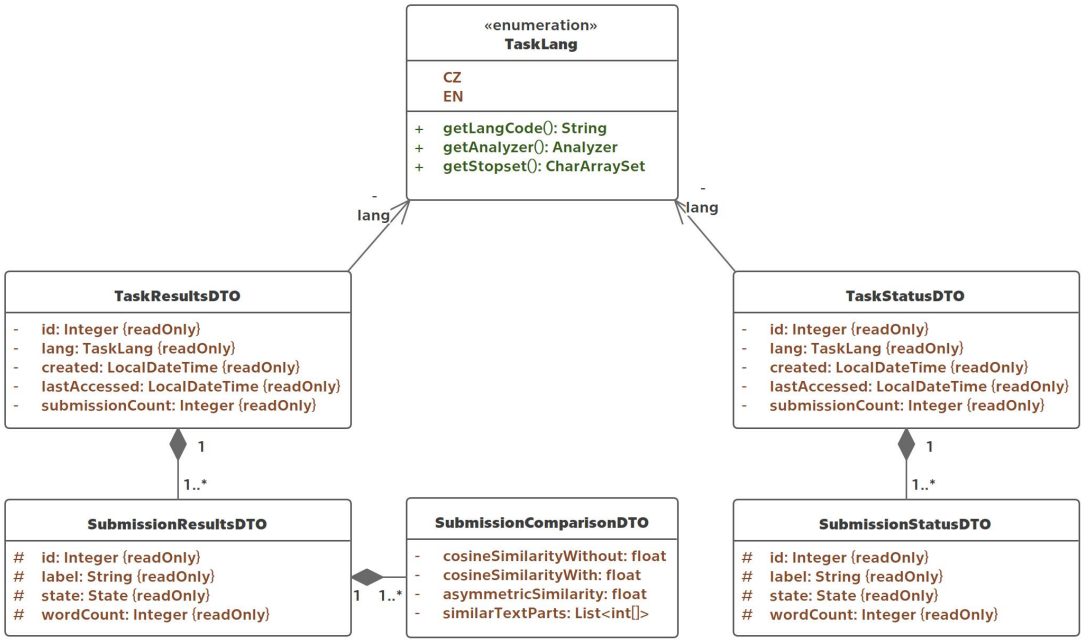


Obrázek 3.3. Navrhovaný interní datový model služby.

Samotný obsah souborů bude tokenizován, tj. rozdělen na jednotlivá slova, a ta budou spolu s počátečním a koncovým posunem od začátku původního textu, který přijde vhod při zpětném mapování na původní text (např. při vizualizaci podezřelých částí textu), vložen do třídy *SubmissionToken*. Následně bude spuštěn proces samotného vzájemného porovnání, jehož výsledky budou uloženy do třídy *SubmissionComparison*.

Všechny nahrané soubory budou mít i tzv. *State*, který přehledně vyjadřuje, v jaké fázi se daný soubor nachází. Pokud je do daného *Tasku* nahrán další soubor, všechny dosud nahrané soubory s ním musí být po jeho zpracování porovnány, aby byla data kompletní.

Uživatel služby však všechna tato data nepotřebuje (nebo mu je vůbec nechceme předávat) – proto je třeba vytvořit i externí datový model obsahující *DTO*[1], jehož prostřednictvím budou uživatelům služby potřebná data vystavena (viz obrázek 3.4).



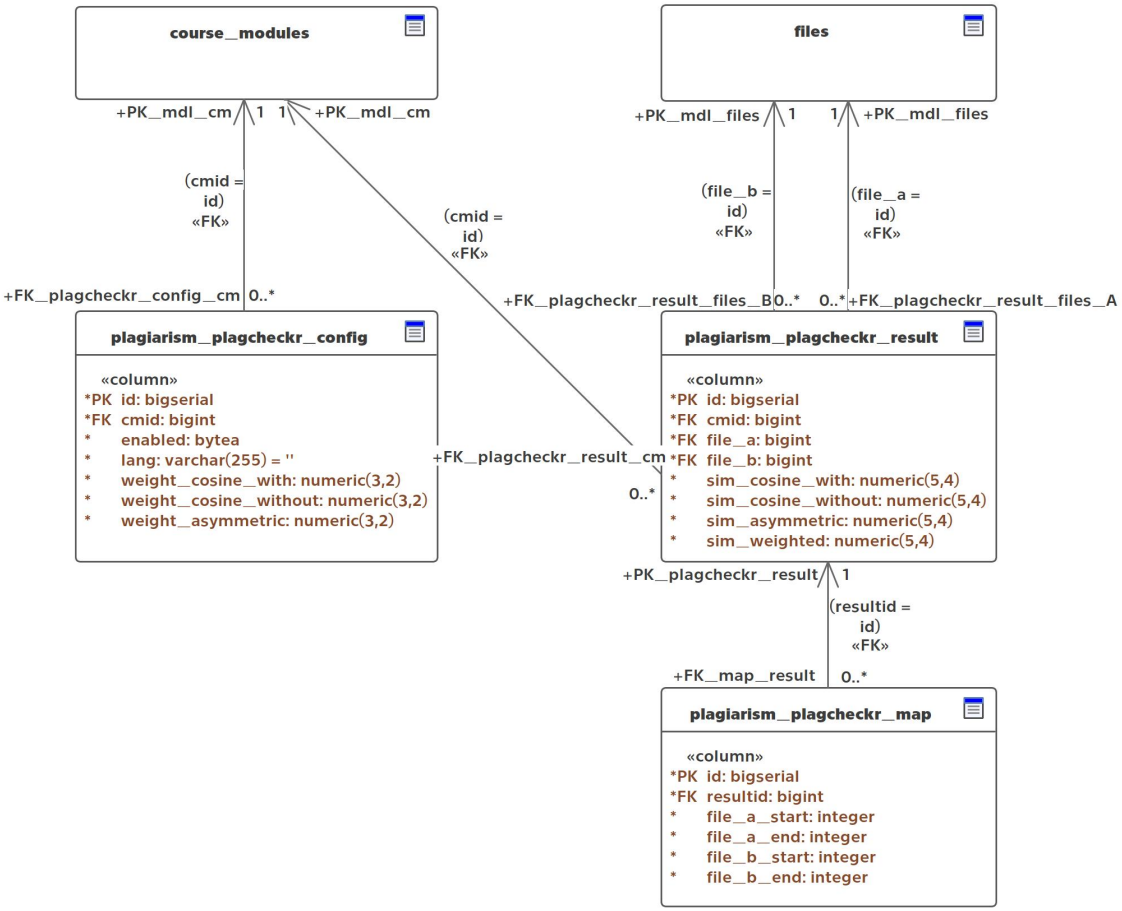
[1] Zkráceně *Data Transfer Object*, jde o objekt, který je přímo určen pro předávání informací mezi procesy či aplikacemi.

Obrázek 3.4. Navrhovaný externí datový model služby.

3.2.2 Datový model zásuvného modulu

Obdobně jako v podsekci 3.2.1, bude i na straně aplikace a jejího zásuvného modulu třeba připravit řádný datový model, do kterého budou data analýzy uložena. V podsekci 2.1.2 byla zmíněna existence tzv. *Plagiarism API*[22], které v Moodle umožňuje napojení na aktivity typu *Úkol*, *Fórum* a *Workshop*, naslouchání odevzdáváním studentů, a jejich následné předání externím službám. Podrobný popis, jak takovýto plugin vytvořit, jaké funkce by měl implementovat, a jaké možnosti napojení na aktivity Moodle umožňuje, lze nalézt v dokumentaci Moodle[34].

Data analýzy (získána od služby) budou uložena do databáze aplikace, do vlastních tabulek. Jejich návrh je možné vidět na obrázku 3.5. Tabulky *course_modules* a *files* jsou součástí jádra Moodle, a proto jsou vyobrazeny jako tzv. *blackbox* – vnitřní struktura těchto tabulek je pro nás nezajímavá, a pouze by celý diagram znepřehlednila. Naopak tabulky začínající prefixem *plagiarism_plagcheckr_* patří přímo zásuvnému modulu.



Obrázek 3.5. Navrhovaný databázový model Moodle modulu.

Tabulka *plagiarism_plagcheckr_config* obsahuje konfiguraci modulu, která se váže k dané aktivitě v kurzu. Uživatel bude mít možnost nastavit jak jazyk analýzy, tak i váhy jednotlivých metrik – ty budou použity při výpočtu průměru, aby tak koncový uživatel měl možnost jednoduše porovnat jednu hodnotu mezi aktivitami, místo třech. Grafické rozhraní této konfigurace je detailněji popsáno v podsekci 4.3.4.

Zbývající dvě tabulky se věnují samotnému ukládání dat převzatých od externí služby – *plagiarism_plagcheckr_result* ukládá numerické metriky dvou souborů, zatímco

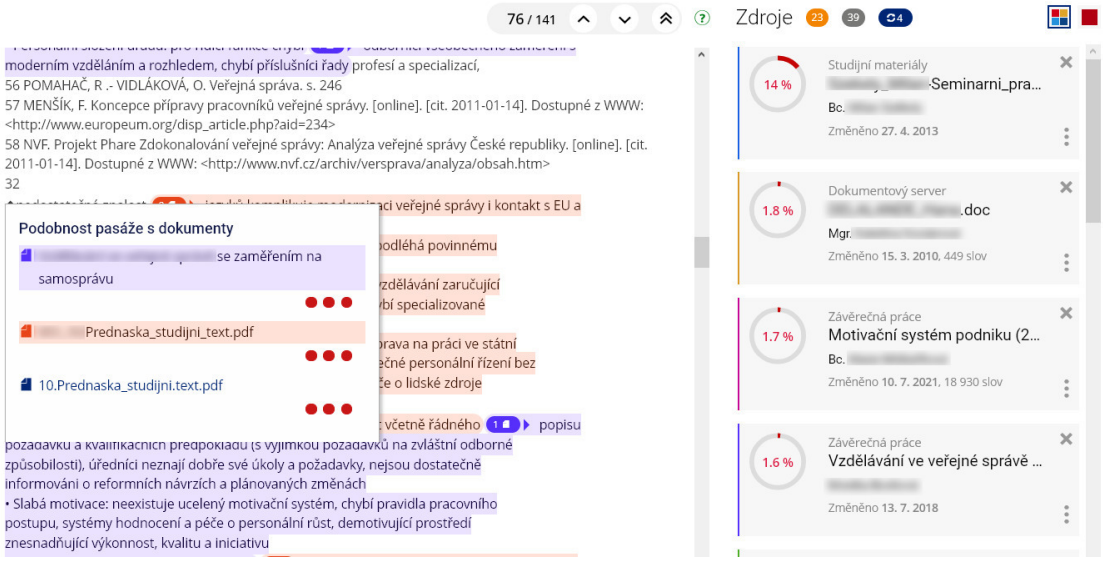
plagiarism_plagcheckr_map obsahuje informace o pozicích podobných částí textů, určené pro grafické vyobrazení uživateli.

3.3 Příklady aplikací třetích stran

Tato práce zdaleka není prvním pokusem o řešení problému plagiátorství v online prostoru. Tato sekce je tedy věnována nejrelevantnějším řešením, ať již komerčním, či akademickým.

3.3.1 Odevzdej.cz

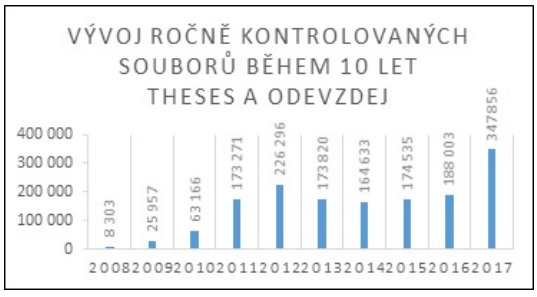
Jedním z nejpoužívanějších systémů v České republice, zabývající se problémem plagiátorství, je internetová aplikace *Odevzdej.cz*[35] (viz obr. 3.6). Ta vznikla již v roce 2009[36] na fakultě informatiky Masarykovy univerzity v Brně, na základě podnětů vyučujících nejen z vysokých škol v naší zemi. Jejím předchůdcem byl v roce 2008 systém *Theses.cz*[37], který dosud operuje.



Obrázek 3.6. Uživatelské rozhraní aplikace *Odevzdej.cz*. Převzato z [35].

Na rozdíl od systému *Theses.cz*, který je oblíbený převážně mezi vysokými školami, je *Odevzdej.cz* dostupný široké veřejnosti. Mohou ho tak využít nejen jednotlivci, ale i školy či firmy. Zároveň je možné napojení na školní e-learningové systémy, včetně Moodle (bohužel však bez veřejně dostupného zásuvného modulu). Tak učinila např. v roce 2022 Univerzita Pardubice[38].

Jak systém *Odevzdej.cz*, tak i jeho předchůdci, se těší neustálému nárůstu obliby. Tomu nasvědčuje i statistika (viz obr. 3.7), která ukazuje, že systémy v roce 2017 zaznamenaly rekordní počet více než 340 000 zkontrolovaných souborů.



Obrázek 3.7. Přírůstky kontrolovaných souborů v systémech *Odevzdej.cz* a *Theses.cz* v letech 2008 až 2017. Převzato z [36].

3.3.2 Řešení s podporou aplikace Moodle

První systém, který přímo podporuje integraci do aplikace Moodle, a zároveň veřejně nabízející zásuvný modul, je *Turnitin*[24]. Projekt vznikl v roce 1998 na základě iniciativy čtyř studentů Kalifornské univerzity v Berkeley – Johna Barrie, Emmanuela Brianda, Melissy Lipscombové a Christiana Storma. Ti v tomto roce vytvořili jednoduchý systém na nahrávání řešení studentů, založený na zpětné vazbě od jejich spolužáků. V roce 2000 je však představena funkce kontroly podobnosti prací, s využitím technologie porovnávání vzorů uložených v databázi, vyvinuté studenty z Berkeley[39].

Dalším systémem s obdobnou integrací je *Ouriginal*[23] (dříve *Urkund*). Ten byl v roce 2021 zakoupen výše zmíněnou společností *Turnitin*, avšak dosud operuje jako samostatný systém[40].

V České republice se odhalováním plagiátorství (a jeho případnou integrací do aplikace Moodle) zabývala diplomová práce Michala Sklenára[41], pro již neexistující systém *Moodle FIT*. Práce byla vypracována ve spolupráci s *Českou zemědělskou univerzitou*, a výsledný zásuvný modul spoléhal na komunikaci s výše zmíněným systémem *Odevzdej.cz*[35]. Velkými nedostatky tohoto modulu byla vysoká cena licence externí služby a nízká přesnost.

V současné době není známo, jestli je modul nadále používán; vzhledem k zániku Moodle FIT, a absenci modulu na *Moodle-vyuka*[42] (systému Moodle pro ostatní fakulty Českého vysokého učení technického v Praze) je to však vysoce nepravděpodobné.

Kapitola 4
Implementace

Následující kapitola je věnována popisu implementace řešení, navrženého na základě podkladů v kapitolách 2 a 3. Součástí jsou i ukázky ze zdrojového kódu či uživatelské obrazovky.

4.1 Výběr technologií

V předchozích kapitolách této práce byly všechny postupy rozebrány obecně, bez návaznosti na konkrétní technologie. Jejich výběrem, včetně řádných odůvodnění, se tak zabývá právě tato sekce.

4.1.1 Jazyk externí služby

Největší volnost ve výběru technologií byla na straně externí služby. Zde bylo v první řadě třeba vybrat jazyk, ve kterém bude služba napsána. Jak bylo zmíněno v kapitole 3.2, cesta externí služby byla zvolena z výkonnostních důvodů. Kromě rychlosti bylo třeba brát i v úvahu následnou implementaci komunikace s aplikací Moodle – jazyk C je bezpochyby jedním z nejrychlejších programovacích jazyků[14], ale jeho použití v externí službě je sice možné, ale vysoce nepraktické[43]. Vzhledem k těmto požadavkům přicházejí v úvahu následující jazyky:

- **Java** – dle vícero zdrojů[44–45] se jedná o nejčastěji používaný jazyk pro mikroservisní architekturu. Má tak širokou podporu komunity, a projektový manažer *Maven* usnadňuje používání knihoven třetích stran – jednou z nich je framework *Spring*[46], který je často používán pro vývoj externích API. Jednou z jeho nevýhod mohou být vyšší nároky na zdroje systému, který aplikaci hostuje – ta je totiž spuštěna ve vlastním virtuálním stroji, tzv. *JVM – Java Virtual Machine*, díky které je však zajištěna téměř dokonalá kompatibilita mezi různými operačními systémy a procesorovými architekturami.
- **Kotlin** – jazyk, který vznikl v roce 2011, a jehož autorem je česká společnost *JetBrains*. Programy napsané v tomto jazyce jsou interoperabilní[1] s knihovnami či jiným kódem psaným v *Javě*. Z tohoto důvodu je však jeho rychlost téměř totožná[47]; hlavní výhodou *Kotlinu* je přehlednější syntaxe.
- **Rust** – jedná se o poměrně nový, ale populární jazyk. Jeho vznik se datuje do roku 2007, jako vedlejší projekt Graydona Hoare. Má vlastního správce balíčků, *Cargo*, který snadno umožňuje instalaci nových knihoven. Mezi ně již patří knihovny pro zpracování textů či vytváření webových služeb[48] – jedná se tak o plnohodnotnou a validní možnost. Oproti *Javě* je značně rychlejší, implementace služby by však byla mírně náročnější, např. kvůli potřebě spravování paměti přímo v kódu (oproti konceptu *Garbage Collectoru*, používaného v *JVM*).

Při konečném výběru se rozhodovalo mezi jazyky *Java* a *Rust* – v obou byl naimplementován vzorový program, který měl za úkol z přiloženého PDF souboru extrahovat

[1] umožňující pracovat i s kódem jiných jazyků či systémů

prostý text. V případě jazyka *Java* byla použita knihovna *Apache PDFBox*[49], v případě *Rustu* knihovna *pdf-extract*[50]. Oba programy byly testovány na rychlost pomocí vzorového PDF souboru o 50 stránkách (a celkem 45 000 slovech), s vygenerovaným *Lorem Ipsum*[1] textem. Výsledky měření programů (viz příloha A) je možné vidět v tabulce 4.1.

Jazyk	1. běh	2. běh	3. běh	4. běh	5. běh
Rust	3,749s	3,750s	3,811s	3,789s	3,801s
Java	0,856s	0,863s	0,865s	0,854s	0,906s

Tabulka 4.1. Čas vykonání vzorových programů v *Javě* (openjdk 17.0.6) a *Rustu* (rustc 1.68.2). Změřeno pomocí programu *GNU Time* (user time).

Překvapivě zde vidíme, že vzorový program v jazyce *Java* je značně rychlejší; což je v rozporu s dříve uváděnými statistikami[14]. Vysvětlení je však jednoduché – při pozorování bylo zjištěno, že program v jazyce *Rust* používal po celý běh pouze jedno vlákno, oproti osmi dostupným na testovacím stroji – použitá knihovna *pdf-extract* totiž využití vícero vláken nepodporuje. I když je tak výsledek testování neprůkazný, poukazuje na jeden důležitý fakt – vzhledem ke svému stáří a rozšíření jsou knihovny jazyku *Java* nejen lépe optimalizované, ale i mnohem více otestované a stabilnější, než ty v jazyce *Rust*. I z tohoto důvodu tak bylo rozhodnuto, že externí služba bude napsána v jazyce *Java*.

4.1.2 Metodika komunikace

Dalším důležitým aspektem výsledného řešení je výběr metody komunikace mezi externí službou a zásuvným modulem. V úvahu můžeme vzít např. možnosti, které nabízí aplikace Moodle ve svých webových službách[51]:

- **REST** – *REpresentational State Transfer*, je architektonický styl, definovaný Royem Fieldingem v jeho disertační práci z roku 2000[52], a ze všech nabízených možností je tato nejvíce používaná. Je založen na *HTTP* protokolu, používá *HTTP status* kódy k vyznačení stavu, a je nezávislý na jazyku, ve kterém je implementován. Jako jednu z nevýhod může být považováno, jak již bylo řečeno, že se jedná o pouhý styl, a nikoliv standard – některé aplikace tak nemusí dodržovat všechny jeho rysy (příkladem je např. i Moodle[2]). Ty služby, které však tak učiní, nazýváme *RESTful*.
- **XML-RPC** – *XML Remote Procedure Call*, je na rozdíl od *RESTu* standardizovaný protokol, který ale ke komunikaci taktéž využívá *HTTP* protokol, avšak jako datový formát striktně používá *XML* (existuje však alternativa používající datový formát *JSON*, nazývající se *JSON-RPC*). Vznikl v roce 1998 za podpory společnosti *Microsoft*. V současné době je od něj již upouštěno[3] na úkor jeho nástupce *SOAP* (viz níže).
- **SOAP** – *Simple Object Access Protocol* vznikl vývojem z *XML-RPC*, a propůjčuje si od něj mnoho vlastností. Na rozdíl od něj však může používat libovolný komunikační protokol, nejen tedy *HTTP* (i tak je však používán nejčastěji). Přes něj jsou posílány

[1] Lorem Ipsum je text, který by měl svým vzhledem a rozložením připomínat opravdový jazyk. V dnešní době se využívá převážně v typografii.

[2] Webové služby Moodle například nesprávně používají HTTP stavové kódy – nehledě na výsledek požadavku vždy vrací hodnotu 200 – OK. Tento fakt je vývojářům známý, avšak v současné době neplánují změnu[53].

[3] Například jazyk PHP ve verzi 8 již rozšíření, které se o komunikaci pomocí protokolu stará, nabízí pouze jako externí balíček, který již není udržovaný[54].

tzv. *obálky*, skládající se z hlavičky (obsahující např. informace o autentizaci) a samotného těla zprávy. Volitelně může *obálka* obsahovat i chybovou zprávu v případě, že došlo na straně serveru k chybě – to značí i příslušný *HTTP status* kód 500.

Pro externí službu byl nakonec vybrán styl *REST* – vzhledem ke své oblíbenosti mezi vývojáři, stáří protokolu *SOAP*, ale i například menší náročnosti na přenos dat (jelikož používá klasické *HTTP hlavičky* oproti hlavičkám *SOAP*, které jsou v těle *HTTP* odpovědi). Zároveň nám umožňuje použít libovolný datový formát, nejen *XML*.

4.1.3 Datový formát

Vzhledem k výběru metodiky v předchozí podsekci bylo potřeba vybrat i formát, ve kterém bude externí služba poskytovat odpovědi. Mezi nejčastěji používané formáty patří:

- **JSON** – *JavaScript Object Notation*, je dnes jedním z nejoblíbenějších datových formátů, leč byl uveřejněn již v roce 2001 – už tehdy se však stal oblíbeným formátem pro komunikaci mezi serverem a prohlížečem. Byl odvozen z části jazyka *JavaScript*, lze ho však použít v libovolném jiném programovacím jazyku. Data jsou organizovaná do asociativního objektu klíč – hodnota, a zanoření je zde značeno složitými závorkami. Výhodami je poměrně dobrá čitelnost, spolu s jednoduchou syntaxí. Nevýhodou může být např. nemožnost vyjádřit obousměrné vazby mezi objekty – musí tomu tak být přizpůsoben datový model. Zároveň zde chybí možnost provazovat části souboru pomocí odkazů či přidávat komentáře (ty však dovolují nadmnožiny jazyka, jako např. *HJSON* či *JSON5*).
- **XML** – *eXtensible Markup Language*, potomek *SGML – Standard Generalized Markup Language* a příbuzný jazyka *HTML*, je zároveň značkovací jazyk a datový formát. Zveřejněním v roce 1998 jde o nejstarší ze zmíněných formátů. Jeho velkou výhodou je snadná validace – každá data musí být obsažena v příslušné *značce*, která má jasně daný začátek a konec. Na rozdíl od formátu *JSON* nativně podporuje komentáře, nebo například provázání více částí dokumentů či zcela oddělené dokumenty. Jeho nevýhodou je však jeho poměrně zdlouhavá syntaxe, která navyšuje objem poslaných dat a redundanci (duplikaci dat, která neobsahují přidanou informaci).
- **YAML** – *YAML Ain't Markup Language*[1]. Oficiálně byl uveden v roce 2001, a jeho hlavní výhodou je přehlednost a čitelnost. Vnoření je zde značeno příslušným odsazením podobně, jako tomu je např. v jazyce *Python*. Nativně podporuje základní datové typy, včetně polí a map. Zmíněná metoda vnořování pomocí odsazení však může působit ve větších souborech problémy, jelikož lze poměrně snadno v zápisu nebo čtení udělat chybu.

Srovnání syntaxe uvedených datových formátů lze vidět na obrázku 4.1. Po zvážení všech výhod a nevýhod jednotlivých formátů bylo rozhodnuto o použití formátu *JSON* – hlavní bylo jeho rozšíření mezi webovými službami a jednoduchá syntaxe.

[1] Původní znění *Yet Another Markup Language*[55] bylo posléze změněno na výše zmíněný rekurzivní akronym, z důvodu zdůraznění, že jde o datový formát, a nikoliv značkovací.

```
{
    "email": {
        "from": "petr@novotny.cz",
        "to": "jarymiro@fel.cvut.cz",
        "subject": "Nedoplatek za plyn",
        "body": "Lorem ipsum dolor sit amet, consectetur adipiscing elit..."
    }
}
```

```
<email>
    <from>petr@novotny.cz</from>
    <to>jarymiro@fel.cvut.cz</to>
    <subject>Nedoplatek za plyn</subject>
    <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</body>
</email>
```

```
email:
  from: petr@novotny.cz
  to: jarymiro@fel.cvut.cz
  subject: Nedoplatek za plyn
  body: Lorem ipsum dolor sit amet, consectetur adipiscing elit...
```

Obrázek 4.1. Srovnání syntaxe datových formátů. Shora dolů: *JSON*, *XML*, *YAML*.

4.1.4 Použité knihovny

Pro usnadnění implementace výsledného řešení byly použity knihovny třetí strany – a to jak v externí službě, tak i zásuvném modulu aplikace *Moodle*.

Na straně externí služby byly vybrány celkem tři knihovny, které usnadňují celý proces analýzy souborů. První z nich se nazývá *Apache Lucene*[56], a je ze všech ta nejzajímavější, a zároveň nejdůležitější – slouží k samotné lingvistické analýze textů a jejich tokenizaci. Knihovna má podporu téměř čtyřiceti světových jazyků[1], včetně češtiny a angličtiny, které jsou pro tuto práci nejdůležitější.

Dalšími použitými Java knihovnami jsou *Apache PDFBox*[49] a *Mapstruct*[58]. První byla již zmíněna v podsekci 4.1.1 – slouží k zpracování PDF souborů, a v práci je použita k extrahování textu. *Mapstruct* následně pomáhá s mapováním mezi interním datovým modelem (ve kterém jsou data uložena a následně zpracována) a externím (který je dostupný přes koncové body služby).

Ze strany Moodle je následně použita pouze jedna komunitní, open-source knihovna PDF.js[59], která je podporována společností *Mozilla*, a která ji používá ve svém prohlížeči *Firefox*. Umožňuje vykreslovat a zpracovávat PDF soubory za pomocí skriptů v jazyce *JavaScript*, a s její pomocí jsou vyučujícím graficky zobrazovány výsledky analýzy.

Všechny použité knihovny jsou zveřejněny pod licencí *Apache License 2.0*[60].

4.2 Implementace služby

Vzhledem k tomu, že je celá služba primárně vyvíjena pro fakultní instanci Moodle, která je spravována *Centrem znalostního managementu*[61], pro kostru celé služby byl použit mikroservisní *archetype*[62] – předdefinovaná šablona, ze které lze dědit knihovny či konfiguraci. Zároveň se zde nachází vzorový kód, kterým je možné se při implementaci řídit. Ten je po seznámení se se strukturou služby určen ke smazání.

Kód celé služby se nachází v balíčku (angl. *package*) `cz.cvut.fel.czm.plagcheckr`, a pro přehlednost je dále organizován do balíčků dle jeho účelu:

[1] Více informací lze nalézt v *Javadoc* dokumentaci knihovny[57].

- **api** – zde jsou definovány tzv. *ovladače* (angl. *controller*), které definují a obsluhují koncové body služby, její vstupní parametry a formát výstupu;
- **service**, ve které je definována business logika služby – ta je volána z výše zmíněných *ovladačů*. Zároveň má na starost uchovávání dat nahraných souborů.
- **model**, který reprezentuje datové modely použité službou (a které jsou popsány v podsekci 3.2.1) – jak bylo již popsáno dříve, je model rozdělen na interní (používaný uvnitř služby) a externí (který je vystaven uživatelům služby);
- **mapper**, kde jsou za použití knihovny *Mapstruct* (zmíněné v podsekci 4.1.4) předepsána mapování interního datového modelu na externí;
- **config**, obsahující konfiguraci služby – zabezpečení, detaily práce s *HTTP* protokolem apod.;
- **exception**, kde jsou definovány výjimky, které může služba v průběhu svého běhu vyvolat, a se kterými se může uživatel setkat – každá výjimka je opatřena příslušným *HTTP kódem* a chybovou hláškou obsahující základní informace.

Veškerý kód také rozsáhle využívá tzv. *anotace* (angl. *Java Annotations*). Jde o metadata kódu, jako např. komentáře, které jsou ale určeny kompilátoru, a mohou tak změnit či rozšířit funkcionalitu dané třídy či funkce. Jednou z nejznámějších anotací je `@Override` (dostupná od verze Java 1.5), která se používá, pokud potomek třídy přepisuje funkci svého rodiče.

4.2.1 Vytvoření požadavku

Pro zahájení analýzy musí uživatel nejprve založit nový požadavek (dále *Task*) na porovnání sady souborů. Může tak učinit zavoláním koncového bodu `/task`, jenž je obsluhován třídou *TaskController*. Jeho implementace je zachycena na obrázku 4.2.

```
@RestController
@RequestMapping(value = "/task")
@RequiredArgsConstructor
public class TaskController {

    private final TaskMapper taskMapper;

    private final TaskService taskService;

    /**
     * Create new task for current user.
     *
     * @param taskLang (optional, default cz) language of this task
     * @return response entity containing the task status
     */
    @PostMapping(value = "")
    public ResponseEntity<TaskStatusDTO> createTask(
            @RequestParam(required = false, defaultValue = "cz") TaskLang taskLang) {

        Task task = taskService.createTask(taskLang);
        return new ResponseEntity<>(taskMapper.createTaskStatusDTO(task), HttpStatus.CREATED);
    }
...
}
```

Obrázek 4.2. Implementace ovladače *TaskController*. Některé části kódu jsou z důvodu délky schovány.

Anotace na prvních třech řádcích jsou poměrně samovysvětlující – `RestController` informuje kompilátor, že se jedná o *controller* služby postavené na *REST* architektuře. `@RequestMapping` určuje *URI* mapování celé třídy, které zde slouží jako prefix – tedy

například pokud bude mít třída nastaveno mapování na `/trida` a funkce na `/funkce`, výsledné *URI* bude `/trida/funkce`. Třetí anotace pak pouze určuje, aby kompilátor sám vytvořil konstruktor s použitými proměnnými třídy – zde `taskMapper` a `taskService`. Životnost těchto proměnných tak řídí přímo *Spring*, a lze mu tak např. sdělit, že celý kód by měl používat jednu totožnou instanci třídy *TaskService*.

Následuje samotná definice funkce – zde lze nalézt anotaci `@PostMapping`. Ta rozšiřuje anotaci `@RequestMapping` o informaci, že se jedná o HTTP metodu *POST*, která formálně slouží k vytváření či modifikování požadavků. Hlavička funkce uvádí, že je zde přijímán pouze jeden argument *taskLang*, který je taktéž anotován – kompilátoru sděluje, že se jedná o nepovinný parametr požadavku a jehož výchozí hodnota je text *cz*. Třída *TaskLang* je sice enumerátor (který jak bylo zmíněno v podsekci 3.2.1 určuje jazyk analýzy), a uživatel ho samozřejmě nemá jak zaslat[1] – *Spring* však umí automaticky z hodnoty textu vytvořit tento enumerátor, a proto může být přímo uveden jako parametr funkce.

V těle funkce je již pouze zavolána business logika, a uživateli je vrácen externí model nově vytvořeného tasku. Opět, *Spring* se umí automaticky postarat o to, aby daná třída byla převedena do datového formátu *JSON* (vybraného v kapitole 4.1.3), což nejen zjednoduší práci, ale i zlepšuje čitelnost celého kódu, a výrazně tak usnadní jeho udržování v budoucím vývoji. Data jsou poté zabalena do třídy *ResponseEntity*, ve které můžeme specifikovat i např. status kód (zde 201 – *CREATED*). Příklad vrácených dat můžeme vidět na obrázku 4.3.


```
{
  "id": 0,
  "lang": "EN",
  "created": "2023-04-16T10:54:25.795914486",
  "lastAccessed": "2023-04-16T10:54:25.796499501",
  "submissionCount": 0,
  "submissions": []
}
```


Obrázek 4.3. Odpověď koncového bodu vytvoření nového tasku (`/task`). Vytvořeno za pomocí aplikace *Insomnia*[63].

4.2.2 Nahrání dat

Vzhledem k tomu, že komunikace pomocí *HTTP* (stejně tak jako *REST*) jsou bezstavové[2], je na uživateli, aby si pamatoval např. identifikátor tasku, který mu byl přidělen (viz odpověď na obrázku 4.3 pod klíčem `id`), a který bude použit v další komunikaci – bude totiž součástí *URI* např. při nahrávání souborů k analýze; každý požadavek musí mít vlastní, unikátní *URI*.

Definice koncových bodů zabývajících se prací se soubory je uložena ve třídě *SubmissionController*. Část jeho implementace je znázorněna na obrázku 4.4.

[1] Parametry volání koncových bodů jsou výlučně primitivních typů jako číslo nebo prostý text.

[2] Bezstavová komunikace neukládá informace o předchozí komunikaci či změnách, a vždy vrací stejnou odpověď na stejný požadavek. Na stejném principu fungují např. matematické funkce; $\log 100$ vždy vrátí hodnotu 2, ať se ptáme v Praze či na Měsíci, dnes či za 100 let…

```java
@RestController
@RequestMapping(value = "/task/{taskId}/submission")
@RequiredArgsConstructor
public class SubmissionController {

    private final TaskService taskService;

    private final SubmissionMapper submissionMapper;

    /**
     * Create (upload) new submission for the specified task.
     *
     * @param taskId  id of the associated task
     * @param file    uploaded file
     * @param label   (optional, default empty) user-defined label, used in DTOs
     * @param control (optional, default false) is this a control submission?
     * @return response entity containing the submission status
     * @throws AbstractExceptionResponse
     * @throws IOException
     */
    @PostMapping(value = "")
    public ResponseEntity<SubmissionStatusDTO> createSubmission(
            @PathVariable Integer taskId,
            @RequestParam MultipartFile file,
            @RequestParam(required = false, defaultValue = "") String label,
            @RequestParam(required = false, defaultValue = "false") boolean control)
            throws AbstractExceptionResponse, IOException {

        String fileType = file.getContentType();
        if (fileType == null) {
            throw new InvalidFileException();
        }
        if (!fileType.equals("application/pdf")) {
            throw new UnsupportedFileTypeException(fileType);
        }

        Task task = taskService.getTask(taskId);
        InputStream pdfStream = file.getInputStream();
        Submission submission = task.createSubmission(label, pdfStream, control);

        return new ResponseEntity<>(submissionMapper.createSubmissionStatusDTO(submission), HttpStatus.CREATED);
    }
...
}
```

Obrázek 4.4. Implementace ovladače *SubmissionController*. Některé části kódu jsou z důvodu délky schovány.

Prvotní tři anotace jsou stejné jako v podsekci 4.2.1, pouze u anotace `@RequestMapping` se změnila *URI* adresa – ve složených závorkách nyní obsahuje zmíněnou proměnnou *taskId*, která je následně v parametrech funkce referována jako `@PathVariable`.

Funkce obsluhující nahrávání souborů, opět za pomocí HTTP metody *POST*, však akceptuje vícero parametrů. Kromě již objasněného *taskId* se zde nachází parametr *file*, obsahující samotný nahraný soubor (reprezentovaný třídou `MultipartFile`), a další dva nepovinné parametry – *label*, který umožňuje uživateli označit daný soubor vlastním štítkem (který je následně používán ve výsledcích analýzy), a přepínač *control* určující, zda-li se jedná o kontrolní řešení (viz podsekce 3.2.1).

Tělo funkce následně obsahuje kontrolu, zda-li je nahraný soubor validní a ve formátu *PDF*, a poté se již o soubor stará opět business logika, která ho uloží do datového modelu, a přiřadí ho danému tasku. Nakonec je uživateli opět vráceno *DTO* s HTTP kódem 201 – *CREATED*. Tělo odpovědi lze vidět na obrázku 4.5.

```
{
  "id": 0,
  "label": "Podezřelý soubor",
  "state": "EVALUATION_FINISHED",
  "tokensWith": 13215,
  "tokensWithout": 9450
}
```


Obrázek 4.5. Odpověď koncového bodu nahrání souboru (`/task/0/submission`). Vytvořeno za pomocí aplikace *Insomnia*[63].

4.2.3 Předzpracování textu

Ve vrácených datech nahrání souboru je vidět, že se nově poslaný soubor již po pouhém nahrání nachází ve stavu `EVALUATION_FINISHED` – v průběhu implementace bylo zjištěno, že předpoklady provedené v podsekci 4.1.1 byly správné. Doba analýzy souborů se pohybuje v rozsahu desítek milisekund až nižších jednotek vteřin, v závislosti na počtu již nahraných souborů a délce jejich textů[1]. Soubory jsou tak zpracovány a porovnány hned po nahrání, a nikoliv asynchronně, jak bylo původně zamýšleno. Případná úprava kódu by však neměla být problematická – na rozdíl od např. jazyka *C* můžeme i zde použít anotaci knihovny *Spring*, přesněji `@Async`, která se o vytváření a správu vláken postará. Tato možnost je dále rozebrána v podsekci 5.2.2.

Veškeré zpracování dat je definováno přímo v třídách *Submission* (kde probíhá extrahování a lemmatizace textu) a *SubmissionComparison* (kde jsou jednotlivé soubory porovnány). Text je extrahován přímo v konstruktoru třídy (viz obrázek 4.6).

```
protected Submission(Integer id, String label, Task task, InputStream inputStream) throws IOException {
    this.id = id;
    this.label = label;
    this.task = task;
    this.state = State.UPLOADED;

    // Parse the inputstream into string
    PDDocument doc;
    try {
        doc = PDDocument.load(inputStream);
    } finally {
        inputStream.close();
    }

    // Get the text and remove all non-alphabetical characters
    try {
        this.origText = (new PDFTextStripper()).getText(doc)
            .replaceAll("[^aábcčddeéěfghiíjklmnňoópqrřsštťuúůvwxyýzžAÁBCČDĎEÉĚFGHIÍJKLMNŇOÓPQRŘSŠTŤUÚŮVWXYÝZŽ\\s]", " ")
            .replaceAll("\\s+", " ").trim();
    } finally {
        doc.close();
    }

    // Create lists of tokens
    this.tokensWith = this.lemmatizateText();
    this.tokensWithout = this.tokensWith.stream()
            .filter(token -> !this.task.getLang().getStopset().contains(token.getWord()))
            .collect(Collectors.toCollection(SubmissionTokenList::new));
}
```

Obrázek 4.6. Konstruktor třídy *Submission*, obsahující extrahování textu z PDF souboru.

Konstruktoru je předán PDF soubor v podobě třídy *InputStream*, která soubor reprezentuje jako bytový proud. Ten je následně převeden do třídy *PDDocument*, která je už součástí knihovny *Apache PDFBox*, a která slouží přímo jako reprezentace PDF souboru, uchovaného v paměti procesu. Následně je text extrahován, a pomocí dvou

[1] Každý n-tý nově nahraný soubor musí být totiž zpětně porovnán s $n - 1$ již zpracovanými soubory.

regulárních výrazů upraven – první z textu odstraní všechny nealfabetické znaky, a nahradí je mezerami; druhý pak všechny bílé znaky (mezery, znaky nového řádků apod.) nahradí jednoduchou mezerou. Platí tak i pokud je za sebou několik stejných či různých bílých znaků – všechny jsou převedeny na jednu mezeru. Je tak učiněno ze dvou důvodů:

- **úspora paměti** – ostatní znaky, jako jsou interpunkce, nové řádky či vzorce jsou pro analýzu insignifikantní, a není tak potřeba je ukládat;
- **jednotná interpretace znaků** – i když je formát PDF definován standardem, může se jeho interpretace lišit – blíže je tento problém popsán v sekci 4.4.

Po extrahování textu je následně zavolána metoda `lemmatizeText`, která upravený text lemmatizuje do seznamu všech slov – tokenů. Ty jsou následně dodatečně vyfiltrovany o stopslova[26] pro použití při výpočtu kosinové podobnosti (bez stopslov). Význam stopslov byl přiblížen v podsekci 2.2.1.

Při implementaci bylo myšleno na snadné rozšíření funkcionality – pokud by například v budoucnu mělo být umožněno nahrání čistého textu přímo do koncového bodu služby, jediná nutná úprava logiky by byla v podobě vytvoření nového konstruktoru, který by místo souboru přijímal jako parametr prostý text; zbytek logiky a funkcí může být zachován v současném stavu.

Již zmíněná funkce `lemmatizeText` je k vidění na obrázku 4.7.

```
/**
 * Method to process submitted text - split it into a list of lemmatized tokens.
 * @return list of lemmatized tokens
 * @throws IOException
 */
private SubmissionTokenList lemmatizateText() throws IOException {
    SubmissionTokenList tokens = new SubmissionTokenList();
    Analyzer analyzer = this.task.getLang().getAnalyzer();

    // Create token stream
    TokenStream tokenStream = analyzer.tokenStream("", this.origText);

    // Create attributes where data will be stored (same principle as pointers)
    CharTermAttribute charAtt = tokenStream.addAttribute(CharTermAttribute.class);
    OffsetAttribute offsetAtt = tokenStream.addAttribute(OffsetAttribute.class);

    // Reset this stream to the beginning
    try {
        tokenStream.reset();
        while (tokenStream.incrementToken()) {
            // Calc offset difference without whitespace chars (as it cannot be retrieved from the token)
            String subString = this.origText.substring(0, offsetAtt.startOffset());
            int difference = subString.length() - subString.replace(" ", "").length();
            // Store lemmatized token
            Offset offset = new Offset(offsetAtt.startOffset() - difference, offsetAtt.endOffset() - difference);
            SubmissionToken aToken = new SubmissionToken(offset, charAtt.toString());

            tokens.add(aToken);
        }
        // Perform end of stream operations
        tokenStream.end();
    } finally {
        // Correctly terminate token stream
        tokenStream.close();
    }

    // Correctly terminate analyzer
    analyzer.close();

    return tokens;
}
```

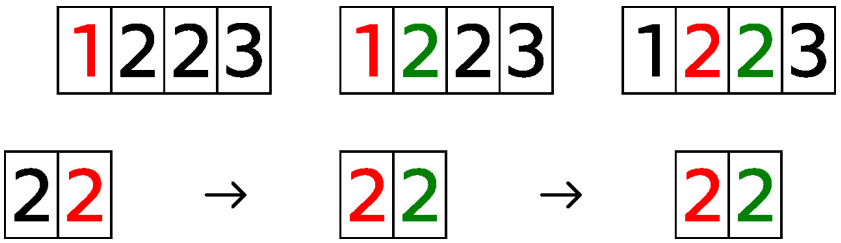
Obrázek 4.7. Implementace lemmatizace textu ve třídě *Submission*.

Zde je na základě vybraného jazyka *Tasku*, ke kterému soubor patří, vytvořen patřičný analyzátor z knihovny *Apache Lucene*, který nevynechává stopslova. Z něj je vytvořen proud tokenů, reprezentující zpracovaná slova, která v následné iteraci ukládáme do interního datového modelu, včetně informace o pozici v původním textu. Ta je však počítána dodatečně, bez ohledu na netisknutelné znaky; odůvodnění je blíže popsáno v sekci 4.4. Po ukončení celé iterace je proud řádně uzavřen, a funkce vrací *List* zpracovaných tokenů.

4.2.4 Porovnání textů

Samotné porovnání textů poté probíhá v třídě *SubmissionComparison*. Jsou zde implementovány tři metriky vyjádřené hodnotou, a jedna vyjádřená mapováním textů. Vybranými metrikami, vyjadřující podobnost hodnotou, jsou asymetrická a kosinova podobnost (za použití textu bez nebo se stopslovy), a způsob jejich výpočtu byl popsán v podsekci 2.2.2.

Poněkud zajímavější je však algoritmus použitý pro podobnost vyjádřenou mapováním – zde byl zvolen poněkud jiný přístup. Oba seznamy tokenů jsou iterovány tak, že posuneme oba seznamy, abychom porovnávali poslední prvek prvního seznamu s prvním prvkem druhého. Po skončení každé iterace vždy s jedním ze seznamů posuneme, a opět porovnáváme jejich prvky. Grafickou ukázku třech takovýchto iterací lze vidět na obr. 4.8. Iteracemi je zajištěno, že pokud se shodný text nachází v prvním souboru na začátku textu, a ve druhém na jeho konci (či naopak), algoritmus je schopný ho detekovat.



Obrázek 4.8. Grafická ukázka iterace seznamu tokenů, prováděná při hledání mapování textů.

V každé této iteraci poté hledáme nejdelší shodný úsek textu. Pokud tedy vezmeme v potaz zjednodušený příklad z obrázku 4.8, pak v první iteraci není žádný shodný úsek, v druhé se nachází jeden úsek délky 1, a ve třetí jeden úsek délky 2. Při hledání těchto úseků jsou povoleny dva druhy chyb – *krátkodobé* a *dlouhodobé*. Krátkodobé vyjadřují, kolikrát v řadě jsme narazili na neshodu při aktuálním porovnání; pokud je překročen stanovený práh, je úsek ukončen a zapamatován. Rovněž se tak stane u dlouhodobé chyby, kdy poměr celkového počtu neshod a celkového počtu slov taktéž překročí stanovený práh. Hodnotami těchto prahů se nadále zabývá testování v podsekci 5.2.3; tam je taktéž rozebrán další faktor – délka ukládaného úseku. Je nežádoucí, aby se ukládali např. úseky délky 1, tedy samostatná slova – s nejvyšší pravděpodobností se totiž nebude jednat o plagiátorství, pokud jsou ve dvou dokumentech použita stejná slova[1].

4.2.5 Vystavení výsledků

Pro uživatele jsou k dispozici koncové body `/task/TID` (pro zobrazení stavu daného tasku) a `/task/TID/results` (pro zobrazení výsledků tasku, včetně výsledných porov-

[1] Obzvlášť s přihlédnutím, že porovnávaná slova jsou lemmatizovaná, a osekána tak na svůj kořenový základ.

nání). Implementace je velice podobná té vyobrazené na obr. 4.2, pouze je zde pouze vracen *Task* uložený v třídě *TaskService*, a je zde použito notace `@GetMapping`, která určuje, že se jedná o HTTP metodu *GET*, sloužící k pouhému získávání dat ze serveru. Zároveň je možné zobrazit stav a výsledek jednotlivých souborů pomocí koncových bodů `/task/TID/submission/SID`, resp. `/task/TID/submission/SID/results` (kde `TID` je identifikátor příslušného tasku a `SID` identifikátor příslušného nahraného souboru). Příklad dat vrácených koncovým bodem pro jeden soubor je znázorněn na obrázku 4.9.


```
{
  "id": 0,
  "label": "Novák, Petr",
  "state": "EVALUATION_FINISHED",
  "tokensWith": 13215,
  "tokensWithout": 9450,
  "comparisons": {
    "Černý, Jan": {
      "cosineSimilarityWithout": 0.056926724,
      "cosineSimilarityWith": 0.060573623,
      "asymmetricSimilarity": 0.061904762,
      "similarTextParts": []
    }
  }
}
```


Obrázek 4.9. Odpověď koncového bodu, který poskytuje výsledek analýzy souboru daného tasku (`/task/0/submission/0/results`). Vytvořeno za pomocí aplikace *Insomnia*[63].

V datech je možné vidět, že soubor s identifikátorem 0 byl porovnán s jedním dalším souborem. V porovnáních jsou použity uživatelem definované štítky pro snazší orientaci – pokud je tedy např. uživatel použije pro zaznamenání autorů daných souborů, může tak snadno vidět, kteří autoři jsou podezřelí a kteří nikoliv (což je tento případ – nalezená shoda se u všech metrik pohybuje okolo 6%, a v textech nebyly nalezeny žádné podobné části).

V případě, že by služba nalezla v textech podobné části, obsahoval by klíč `similarTextParts` pole polí celočíselných hodnot, reprezentující počet znaků od začátku dokumentu ve formátu [`začátek_a`, `konec_a`, `začátek_b`, `konec_b`]. Příkladem mohou být data:

$$[[0, 10, 10, 20], [30, 50, 20, 40]],$$

která uživateli říkají, že se texty podobají ve dvou místech – první je umístěn v prvním souboru (který je identifikován v požadavku pomocí přiřazeného ID) mezi 1. a 11. znakem textu[1], a odpovídá mu 11. až 21. znak v druhém souboru (který je identifikován pomocí štítku daného souboru). Druhé místo se nachází v prvním textu mezi 31. a 51. znakem, a v druhém textu mezi 21. a 41. znakem.

4.2.6 Čištění dat

V rámci implementace bylo taktéž třeba myslet na jednu z nevýhod jazyka *Java* – poměrně vysokou náročnost na operační paměť systému. Data o analyzovaných souborech si tak služba nemůže uchovávat nekonečně dlouho; i proto byly do služby implementovány dva způsoby čištění dat.

Prvním je automatické čištění – *Spring* nabízí anotaci `@Scheduled`, která umožňuje nastavit ve službě procedury, které se budou spouštět automaticky v daném intervalu. Příklad použití této anotace je zachycen na obrázku 4.10.

[1] Datová reprezentace je, jakožto téměř vše v programování, číslovaná od nuly; v přirozeném jazyce ale počítáme od jedničky – nikdo nám neřekne, abychom přečetli nulté slovo v nulté větě.

```java
/**
 * Scheduled task deletion
 */
@Scheduled(fixedDelay = 900-000)
public void scheduledDeleteTask() {
    int threshold = 30;

    this.tasks = this.tasks.stream().filter(task -> {
        if (Duration.between(task.getLastAccessed(), LocalDateTime.now()).toMinutes() >= threshold) {
            logger.info("Task ID %d not accesed for %d minutes, purging...".formatted(task.getId(), threshold));
            return false;
        }

        return true;
    }).collect(Collectors.toSet());
}
```

Obrázek 4.10. Implementace automatického čištění dat ve třídě *TaskService*.

Všechny aktuálně uložené tasky jsou zkontrolovány na čas posledního přístupu, a pokud přesáhl hranici 30 minut, je daný task odstraněn. Tato skutečnost je pro případ potřeby zalogována službou, a uživatel je o této skutečnosti informován – pokud se na již smazaný task (či k němu nahraný soubor) dotáže, je upozorněn na jeho smazání chybovou hláškou a HTTP kódem 410 – *Gone*.

Druhou možností je manuální smazání uživatelem pomocí HTTP metody *DELETE* na koncovém bodě `/task/TID` (kde *TID* je identifikátor tasku). Task (i jeho soubory) jsou z paměti služby odstraněny, a uživateli je vrácen HTTP kód 204 – *No Content*. Oproti automatickému mazání je možné mazat i samotné nahrané soubory, například pokud je již uživatel služby (či jiný systém, který s ní komunikuje) nechce dále analyzovat. Může tak provést opět zavoláním metody *DELETE* na koncový bod `/task/TID/submission/SID` (kde *TID* je identifikátor tasku a *SID* identifikátor nahraného souboru).

4.3 Implementace zásuvného modulu

Zásuvné moduly v aplikaci Moodle mají přesně danou strukturu, název i adresář, ve kterém se mají nacházet – podrobný popis lze nalézt v oficiální dokumentaci[64]. Dle konvence bylo rozhodnuto o názvu modulu *plagiarism_plagcheckr*, který se tak bude nacházet v adresáři `plagiarism/plagcheckr`. Nejdůležitější části struktury modulu jsou:

- **amd/** – složka, ve které se nacházejí javascriptové *AMD (Asynchronous Module Definition)* moduly[65] – jedná se o způsob, jak členit kód jazyka *JavaScript* do opakovaně použitelných komponent;
- **classes/** – obsahující objektově orientovaný kód modulu;
- **db/** – v této složce jsou uvedeny informace o práci modulu s databází či naslouchače událostí (jako je např. odevzdání úkolu studentem);
- **lang/** – zde se nachází jazykové překlady modulu;
- **templates/** – adresář, ve kterém jsou umístěny HTML šablony modulu, psané v jazyce Mustache, a které výrazně usnadňují tvorbu uživatelského prostředí Moodle;
- **lib.php** – obsahující definice základních funkcí, jež jsou vyžadovány rozhraním dostupným pro moduly pro odhalování plagiátorství;
- **settings.php** – zde jsou zadefinovány možnosti konfigurace modulu, primárně podrobnosti připojení k službě;
- **styles.css** – definuje kaskádové styly použité pro uživatelské prostředí modulu.

4.3.1 Databázové schéma

Implementace databázového schématu (dle návrhu, představeném v podsekci 3.2.2) je definována v souboru `db/install.xml`. Aplikace Moodle podporuje vícero databázových serverů[66], a proto pro definování databázových tabulek a relací využívá abstraktní vrstvu, nazvanou *XMLDB*[67]. Ta ustanovuje jednotný zápis, v podobě XML dokumentu, který se při instalaci překládá do příslušného dialektu jazyka SQL, dle použitého serveru.

Velkou nevýhodou tohoto řešení je absence základních funkcí, které jsou dnes již součástí všech SQL serverů – příkladem mohou být chybějící cizí klíče či použití constraintů – ty schéma *XMLDB* sice podporuje, avšak Moodle je při čtení z historických důvodů ignoruje[68].

4.3.2 Naslouchání a příprava dat

Zásuvné moduly aplikace Moodle mají možnosti naslouchat událostem, které vytvářejí jiné moduly – může jít o příhlášení uživatele, otevření kurzu, ale i odevzdání úkolu studentem – právě této události je naslouháno. Po jejím přijmu je vytvořena tzv. *adhoc* úloha – jedná se o jeden ze dvou způsobů, jak zpracovávat v Moodle data na pozadí, aniž by uživatel, který proces spustil, musel čekat na jeho dokončení. Adhoc (z lat. *ad hoc – pro tento případ*) úlohy jsou vytvářeny vždy jednorázově pro jeden účel, a po jejich dokončení jsou smazány. Lze tak snadno jakýkoliv výpočetně náročnější proces přesunout na pozadí. Druhým typem jsou *naplánované* úlohy (angl. *scheduled tasks*) – jde o procedury, které jsou opakovaně vykonávány; může jít například o smazání starých uživatelů, čištění mezipaměti apod. Kód adhoc úlohy je k vidění na obrázku 4.11.

```php
public function execute(): void {
    // Get task data
    $cmid = $this->get_custom_data()->cmid;
    $courseid = $this->get_custom_data()->courseid;

    // Parse data to objects
    $context = context_module::instance($cmid);
    $course = get_course($courseid);
    $assign = new assign($context, null, $course);
    $fs = get_file_storage();
    $task_id = task_creation_endpoint::new_for_cmid($cmid)->task_id;

    // Iterate over enrolled users
    trigger_error("Submitting data for analysis for course module $cmid", E_USER_NOTICE);
    foreach (get_enrolled_users($context) as $user) {
        // Get user's latest submission
        if ($submission = $assign->get_user_submission($user->id, false)) {
            // Get all files for this latest submission & submit them
            $files = $fs->get_area_files($context->id, 'assignsubmission_file', 'submission_files', $submission->id, "id", false);
            foreach ($files as $file) {
                $filepath = $fs->get_file_system()->get_local_path_from_storedfile($file);
                trigger_error("Submitting file {$file->get_id()}", E_USER_NOTICE);
                new submission_creation_endpoint($task_id, curl_file_create($filepath, $file->get_mimetype(), $file->get_id()));
            }
        }
    }

    // Check n times for the result - if still not processed, something went horribly wrong -> log it
    for ($n = 0; $n < 60; $n++) {
        $status = new task_status_endpoint($task_id);
        if (!$status->task->is_finished()) {
            trigger_error("API results for course module $cmid still not done, waiting further...", E_USER_NOTICE);
            sleep(5);
        } else {
            $results = new task_results_endpoint($task_id);
            $results->task->save($cmid);
            trigger_error("Saving API results for course module $cmid", E_USER_NOTICE);
            return;
        }
    }

    trigger_error("Tried getting results for course module $cmid too many times! Aborting...", E_USER_ERROR);
}
```

Obrázek 4.11. Implementace adhoc úlohy pro odesílání dat službě.

Úloze jsou předána data pomocí funkce `set_custom_data` (volané při jejím vytvoření), a následně mohou být získána pomocí funkce `get_custom_data` – tento způsob je nutný, jelikož jsou všechny úlohy ukládány a následně získávány z databáze; tak mohou být naplánovány, aniž by musely být uložené v paměti PHP procesu. Zároveň data nejsou ztracena např. v případě výpadku stroje. V datech je uložen jak identifikátor kurzu, ve kterém se aktivita nachází, tak i identifikátor samotné aktivity. Z těchto identifikátorů jsou následně pomocí funkcí poskytnutých jádrem Moodle vytvořeny PHP třídy, se kterými je dále pracováno. Nejdůležitější z nich je třída `assign` – jde o interní reprezentaci aktivity *Úkol* (představené v podsekci 3.1.1), odpovídající aktivitě, do které student odevzdal své řešení (a tím vyvolal výše zmíněnou událost).

Posléze funkce prochází všechny uživatele (kteří jsou oprávněni do dané aktivity nahrávat svá řešení), a pro každého z nich je zavolána funkce `get_user_submission`, jež vrátí objekt posledního nahraného řešení, nebo hodnotu `false`, pokud ještě žádné uživatel nenahrál. Moodle totiž umožňuje vícenásobné odevzdávání a verzování řešení; z hlediska nejen rychlosti, ale i způsobu hodnocení učiteli, je zbytečné analyzovat předchozí verze řešení. Zároveň by bylo nutné řešit konflikty mezi verzemi, které by pravděpodobně obsahovaly velký překryv (uživatel například pouze opravil gramatické chyby či přidal jeden odstavec).

Po odeslání všech souborů následuje cyklus, který v intervalech kontroluje stav celé analýzy – k tomu je využit klíč `state` v datech vrácených službou. V případě, že kontrola trvá příliš dlouho (zde více než pět minut), je úloha ukončena, a problém je zaznamenán. Pokud se tak nestane, a data jsou správně vyhodnocena, úloha je po jejich uložení do databáze taktéž ukončena.

4.3.3 Komunikace s externí službou

Při odesílání dat externí službě jsou, kromě dat vyňatých z databáze aplikace, nastaveny i některé parametry koncových bodů, zmíněné v sekci 4.2 – prvním z nich je jazyk analýzy, který je nastaven dle výběru uživatele v dané aktivitě (viz konfigurace v podsekci 4.3.4). Dále je štítek každého souboru, pro snazší práci při ukládání vyhodnocených dat, nastaven na identifikátor daného odevzdání, získaný z databázové tabulky aplikace.

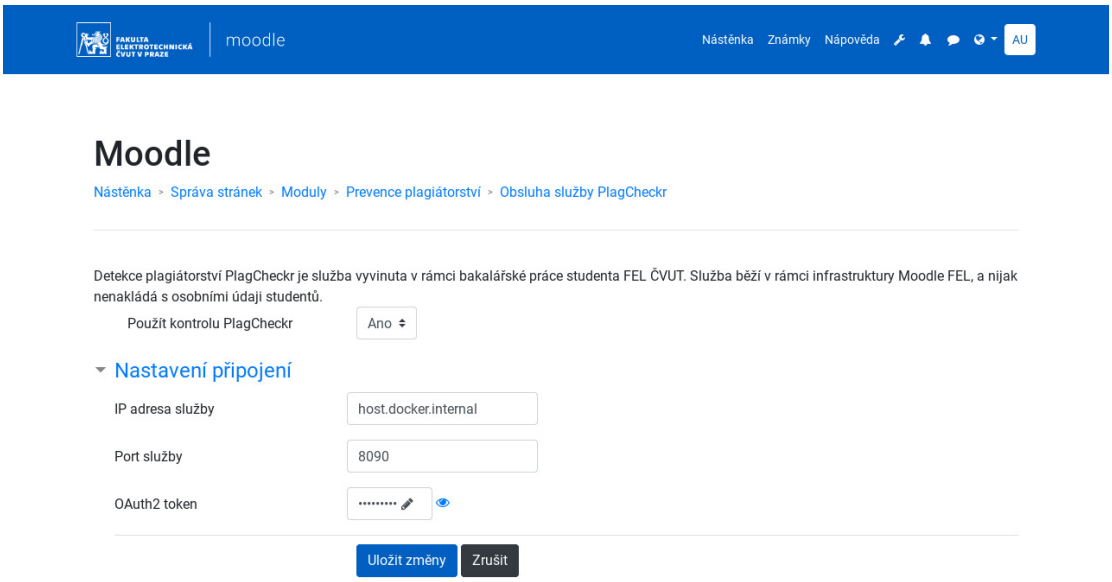
Implementace komunikace s externí službou se nachází v adresáři `classes/api` – nalezneme zde abstraktní třídu `abstract_endpoint`, ve které jsou řešeny technické podrobnosti spojení, a následně její potomci specifikují koncový bod služby, ke kterému se vztahují – např. třída `task_creation_endpoint` se vztahuje k *URI* `/task/TID` za pomocí metody *POST*. Spojení je provedeno za pomocí PHP rozšíření *cURL*[69], využívající stejnojmennou systémovou knihovnu.

Tato knihovna obdržená data uloží do proměnné typu `stdClass`[70]; jedná se o obecnou třídu s dynamickými proměnnými. Z ní jsou následně data převedena do plnohodnotných tříd a objektů, které odpovídají externímu datovému modelu, poskytnutém externí službou, avšak reprezentovaném v jazyce PHP. Důvodem je zajištění datové integrity; PHP není striktně typovaný jazyk, a proto by mohlo dojít k chybné interpretaci dat při dalším zpracování. Spolu s interním datovým modelem (reprezentující databázové tabulky ve formě objektů) je jejich definice uložena v adresáři `classes/model`.

Třídy externího datového modelu disponují metodou `save()`, která je převede na databázové záznamy dle modelu v podsekci 3.2.2. Vzhledem k tomu, že informace o každé analýze jsou ve vrácených datech uvedeny dvakrát (poprvé, když byl první soubor zkoumán s druhým, a podruhé, když byl druhý soubor zkoumán s prvním), je zde tato skutečnost ošetřena, aby tak nebyla v databázi zbytečně uvedena duplicitní informace – záznam je uložen pouze tehdy, když je databázový identifikátor prvního řešení větší než identifikátor druhého.

4.3.4 Konfigurace modulu

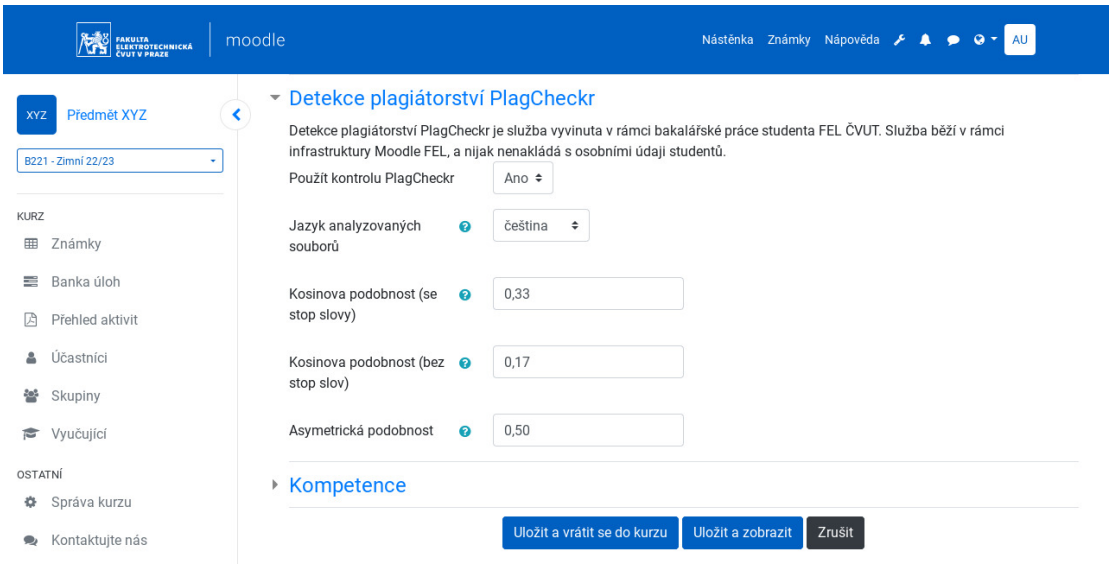
Konfigurace zásuvného modulu je rozdělena na dvě části – první, *administrační*, se vztahuje globálně pro všechny kurzy a aktivity v aplikaci. Nachází se zde konfigurace připojení k externí službě, spolu s možností aktivace či deaktivace celého modulu. K této stránce mají přístup pouze pověření administrátoři spravující instanci aplikace. Uživatelské rozhraní této konfigurace je zachyceno na obrázku 4.12.



Obrázek 4.12. Konfigurace připojení zásuvného modulu k externí službě.

Pro připojení ke službě je třeba zásuvnému modulu vytvořit autentizační token *OAuth 2.0*[71]. Ten je jednak použit pro rozlišení autorů tasků (aby tak uživatel služby neměl přístup k datům jiných uživatelů), a jednak pro samotné oprávnění službu používat. Tyto přístupové tokeny jsou spravovány přímo na stroji, na kterém služba běží.

Druhá část konfigurace, *uživatelská*, se vztahuje pouze na aktivitu, ve které je nastavena, a slouží k vypnutí či zapnutí analýzy na této úrovni, nastavení jejího jazyka, či vah použitých při zobrazení vážené podobnosti (více v podsekci 4.3.5). K této konfiguraci mají přístup vyučující daného kurzu. Její rozhraní je zachyceno na obrázku 4.13.

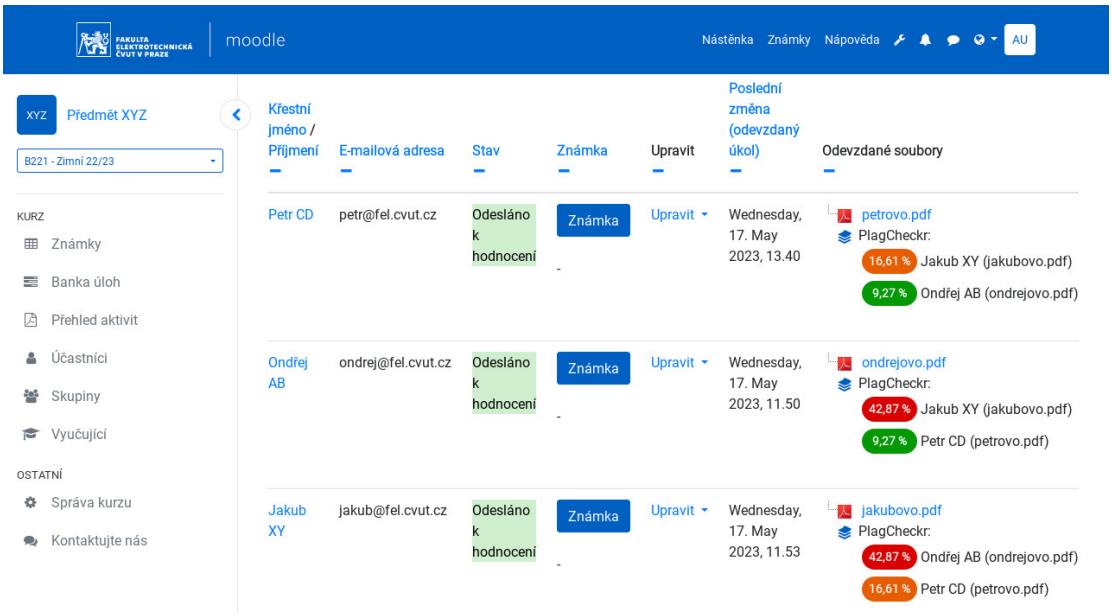


Obrázek 4.13. Konfigurace zásuvného modulu pro vybranou aktivitu.

Při uložení nastavení jsou zvolené váhy zkontrolovány, a případně přepočteny, aby jejich suma byla rovna jedné. Doporučenými hodnotami těchto vah se nadále zabývá podsekce 5.2.4.

4.3.5 Používání modulu

Samotný proces odevzdávání a hodnocení řešení se v aplikaci nemění – student je pouze při nahrávání upozorněn, že jím odevzdané soubory budou podrobeny analýze. Z pohledu vyučujícího je však obohacen celkový přehled odevzdaných úkolů, který může být vidět na obrázku 4.14.

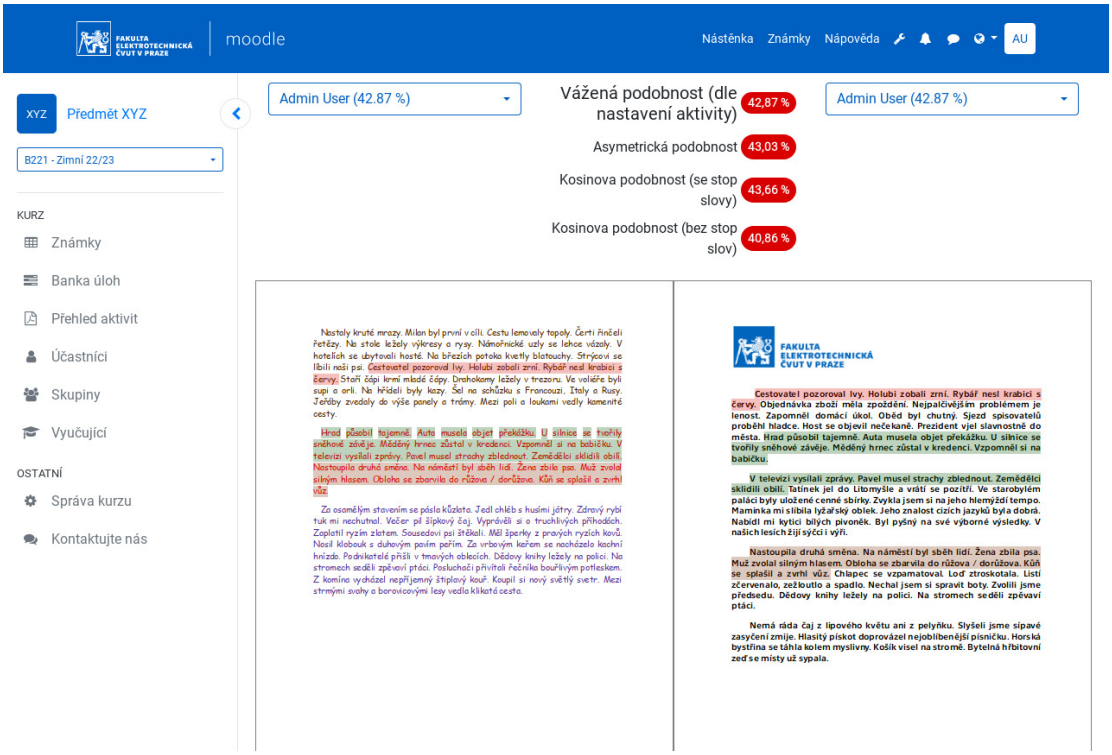


Obrázek 4.14. Uživatelské rozhraní přehledu odevzdaných řešení se zapnutou analýzou *PlagCheckr*.

Na příkladu je vidět, že příslušný úkol odevzdali tři studenti, každý jedním souborem. Oproti standardnímu rozhraní aplikace Moodle jsou zde do posledního sloupce přehledu přidány informace o analýze dotyčného souboru. Přesněji jde o výpis maximálně čtyř porovnání, seřazených dle míry vážené podobnosti – ta je uvedena v barevné buňce.

Největší shoda, kterou modul označil červeně jako téměř jistý plagiát, se nachází mezi řešeními *Ondřeje* a *Jakuba*, konkrétně 43%. Oranžovou barvou, značící nejisté podezření na plagiát, jsou označena řešení *Jakuba* a *Petra* se shodou téměř 17%, a barvou zelenou, značící běžnou shodu mezi dvěma neprovázanými soubory, jsou nakonec označena řešení *Ondřeje* a *Petra* s procentuální mírou 9%.

Na barevnou buňku s vyjádřením podobnosti je možné kliknout – tím se uživateli zobrazí podrobnosti o konkrétním porovnání dotyčných souborů. Příklad rozhraní, pro nejvíce podezřelé porovnání souborů *Ondřeje* a *Jakuba* z příkladu výše lze spatřit na obrázku 4.15.



Obrázek 4.15. Uživatelské rozhraní detailu jednoho porovnání.

V záhlaví stránky jsou na okrajích nadepsána jména dotyčných studentů – ta zároveň fungují jako selektory, pro rychlé přepínání mezi jednotlivými pracemi. Mezi jmény jsou poté vypsány hodnoty všech numerických metrik, včetně jejich váženého průměru. Níže na stránce již následuje samotné vizuální vyznačení podobných částí – ty mohou být zvýrazněny najetím myši na vyznačený úsek. Jednotlivé úseky jsou taktéž barevně odlišeny – k tomuto účelu bylo předdefinováno osm barev, s přihlédnutím na jejich míru rozlišitelnosti.

4.4 Zjištěné problémy

V průběhu implementace řešení byly zjištěny různé limitace knihoven, jazyků či aplikace Moodle jako takové. Těmto problémům je věnována tato podsekce.

Prvním problémem, který byl zjištěn v rané fázi implementace, byla chybějící validace PDF souborů, nahraných studenty. Standard souborů PDF[72] je poměrně rozsáhlý; jeho aktuální revize z roku 2020 čítá 986 stran. Aplikace Moodle v současné době nijak nekontroluje platnost souborů, a tak se může snadno stát, že student nahraje soubor s neplatnými znaky či poškozeným fontem. Při čtení takovéhoto souboru může dojít k deformaci či dokonce poškození čteného textu.

Souvisejícím problémem je různá interpretace dat uložených v souboru – text je zpracováván dvakrát, poprvé na straně služby (pomocí knihovny *Apache Lucene*), a podruhé při vykreslování na straně Moodle (pomocí knihovny *PDF.js*). Snadno se tak může stát, že knihovny mohou data reprezentovat jinak; obzvlášť u knihovny *PDF.js*, kde výsledkem není prostý text, ale HTML kód – kde např. Java vidí znak nového řádku `\n`, tam JavaScript začne nový HTML element.

Aby se zabránilo následným chybám v převodu jedné interpretace do druhé, jsou pozice mapování textů spočteny pouze dle počtu znaků české abecedy. Pokud tedy

budeme hledat znak s takovouto pozicí 5 v textu *nové dveře*, nebude tímto znakem mezera, ale písmeno *d*.

Dalším potenciálním problémem je způsob, jakým je implementována lemmatizace v knihovně *Apache Lucene*. Ta vznikla již v roce 2009[73] na základě výzkumu švýcarské univerzity Neuchatel[74], a od té doby nebyla změněna. Její princip spočívá v osekávání běžných koncovek českých slov a skloňování na uniformní tvar; nejedná se tak o kořenový základ slova jako takový. Zároveň stemmer v knihovně nijak neupravuje předpony slov, což částečně omezuje možnosti analýzy.

Posledním problémem, se kterým bylo třeba se vypořádat, byla velikost výsledných dat analýzy. Původní řešení externího datového modelu, resp. části mapování textu, počítalo s objektem a klíči `startA`, `endA`, `startB` a `endB`. To však způsobilo vysokou míru redundance, a výsledná data dosahovaly velikosti jednotek až nižších desítek megabytů. Po objevení tohoto problému bylo původní řešení nahrazeno současným, tj. pole polí čtyř hodnot, což snížilo objem dat až o 90%.

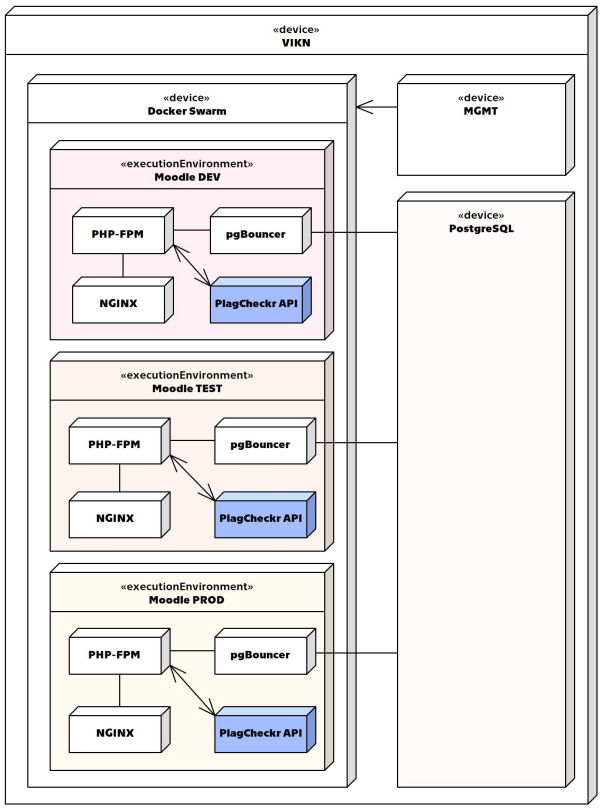
Kapitola 5
Provoz

5.1 Nasazení

Jak již bylo rozebráno v kapitole 4, výsledné řešení se skládá ze dvou částí – zásuvného modulu do aplikace Moodle, a externí služby v jazyce Java. Nasazení zásuvného modulu je poměrně jednoduché – *Moodle FEL ČVUT* pro správu aplikace *Moodle* používá *Git* repozitář na fakultním GitLab serveru[75]. Stačí ho tedy, jako ostatní moduly, které instance používá, přidat do rezpozitáře jako tzv. *submodul*[76]. Po nasazení modulu aplikace sama detekuje jeho přítomnost, a spustí instalační skripty, které např. vytvoří tabulky v aplikační databázi. Nasazení externí služby je rozebráno níže.

5.1.1 Docker a infrastruktura

Celý systém *Moodle FEL ČVUT* běží ve *virtualizační infrastruktuře Karlovo náměstí* (*VIKN*), a používá pro správu prostředí a potřebných služeb izolované Docker kontejnery[77], každý se specifickým účelem. Pro ty výpočetně náročnější existuje tzv. swarm[78], který Dockeru umožňuje rozmístit kontejnery mezi vícero fyzických strojů, a rovnoměrně tak rozdělit jejich zátěž. Taktéž existuje tzv. *mgmt*, sloužící pro správu *swarmu*, monitoring a spouštění pipeline (více v podsekci 5.1.2). Zjednodušené schéma této infrastruktury se nachází na obrázku 5.1.



Obrázek 5.1. Schéma infrastruktury systému Moodle FEL.

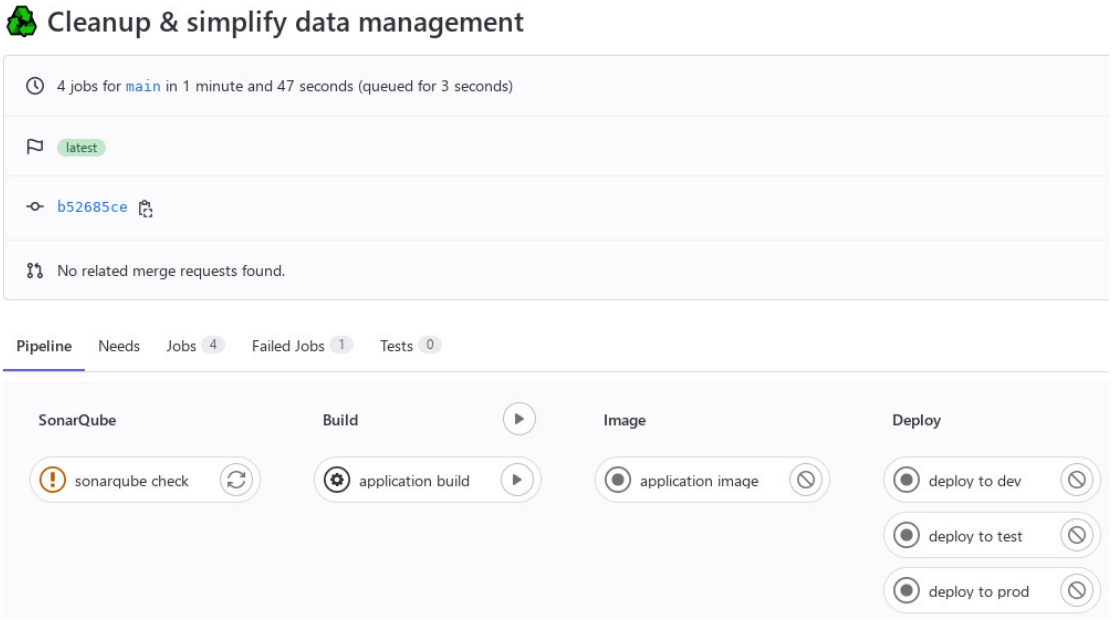
Moodle FEL ČVUT nabízí tři prostředí – vývojové (*DEV*) pro účely vývojářů, testovací (*TEST*) pro účely testování nových funkcionalit osobami mimo vývojový tým, a produkční (*PROD*), na kterém probíhá výuka. V současné době se ve *swarmu* nachází tři kontejnery pro každé z těchto prostředí; *php-fpm*[79] (exekutor PHP kódu), *nginx*[80] (webový server) a *pgBouncer*[81] (optimalizující připojení k databázi přepoužíváním připojení).

Diagram výše již zahrnuje čtvrtý kontejner, ve kterém poběží externí služba *PlagCheckr*. V testovací fázi bude tedy dočasně dostupná pouze systému *Moodle FEL ČVUT*; vzhledem k provedené kontejnerizaci však lze bez potíží v budoucnu přesunout službu na samostatný stroj, případně poskytnout zájemcům mimo *FEL ČVUT* přímo připravený kontejner k vlastnímu použití.

5.1.2 Pipeline a proces nasazení

Nasazení do výše zmíněné infrastruktury probíhá za použití *CI/CD pipelines*[82], poskytnutých aplikací *GitLab*. Pipeliny mohou být definovány jako sada neměnných instrukcí, jejichž cílem je automatizace dodání nových verzí aplikace. Základní jednotkou pipeline je tzv. *job* – úkon, jako například kompilace aplikace, vyčištění dočasných souborů apod. Tyto joby jsou následně shlukovány do etap (angl. *stage*), které určují za jakých podmínek se mají úkony provádět – při každém commitu do repozitáře, při navýšení verze aplikace apod.

Následné spuštění jak celé, tak i části *pipeline* je možné jak příkazem při používání gitu, tak i v grafickém rozhraní *GitLabu* (viz obrázek 5.2).



Obrázek 5.2. Ukázka rozhraní CI/CD Pipeline v aplikaci GitLab.

Zde jsou ve sloupcích vidět jednotlivé etapy, a pod každou dotyčné joby. Fáze *SonarQube* slouží ke statické kontrole kódu za pomocí stejnojmenné aplikace, a ke spuštění jednotkových testů služby. Druhá fáze celou službu přeloží do strojového kódu, který je následně vložen do Docker image ve třetí fázi. Poslední fáze slouží pro samotné nasazení do jednotlivých prostředí.

5.2 Testování

Implementované řešení bylo třeba řádně otestovat. K tomu sloužily jak jednotkové testy (popsané v podsekci 5.2.1), tak i reálná data poskytnuta v rámci analýzy[4] (s jejichž pomocí je provedeno testování v dalších podsekcích), na které tato práce staví. Součástí dat jsou uměle vytvořené plagiáty; jde o jednu stoprocentní kopii, jednu 35% - 70% kopii (vytvořenou vsunutím nesouvisejícího textu do kopie), a jeden dokument vzniklý poskládáním částí vícero jiných dokumentů dohromady.

5.2.1 Jednotkové testy

Jednotkové testy jsou jedním ze způsobů, jak automaticky testovat zdrojový kód. Jejich cílem je kontrola funkčnosti jednotlivých částí – jednotek aplikace. Tyto testy byly implementovány v externí službě, za pomocí knihovny *JUnit*[83], a to pro větší část služby – testy bylo pokryto okolo 64% kódu.

Tyto reporty byly následně automaticky za pomocí pipeline exportovány do aplikace *SonarQube*, která slouží ke statické analýze kódu a kontrole jeho kvality. Její rozhraní je provozováno v rámci *Centra znalostního managementu FEL ČVUT*.

5.2.2 Rychlost analýzy

Při testování rychlosti celé analýzy bylo nejprve pohlíženo na rychlost odpovědí koncových bodů. První test spočíval v nahrání sady již zmíněných testovacích dat, z nichž byl vybrán vzorek 67 souborů. Dle dat analýzy provedené službou soubory obsahovaly mezi 150 a 900 tokenů – slov. Celý sekvenční proces zabral přibližně 7 vteřin, a velikost výsledných dat analýzy byla přibližně 1,5 MB.

Je nutné podotknout, že těchto výsledků bylo dosaženo bez potřeby asynchronního zpracování dat – tedy ve chvíli, kdy byl k tasku nahrán poslední soubor, byla analýza dokončena. Použití již dříve zmíněné anotace `Async` tak není nutné.

Služba však byla podrobena i pokročilejší analýze rychlosti – za použití aplikace *VisualVM*[84]. Ta poskytuje grafické rozhraní, ve kterém je možné detailně pozorovat nejen celkové využití prostředků Java aplikacemi, ale i využití konkrétními funkcemi. Tomuto druhu optimalizace běhu programů se říká *profilování*, a jeho výsledek je zachycen na obrázku 5.3.

Name	Total Time (CPU)	
http-nio-8090-exec-8	1,368 ms	(100%)
cz.cvut.fel.czm.plagcheckr.api.controller.SubmissionController.**createSubmission** (Int	1,368 ms	(100%)
cz.cvut.fel.czm.plagcheckr.model.internal.Task.**createSubmission** (String, java.io.Ir	1,366 ms	(99.8%)
cz.cvut.fel.czm.plagcheckr.model.internal.EvaluatedSubmission.**<init>** (Integer, S	1,137 ms	(83.1%)
cz.cvut.fel.czm.plagcheckr.model.internal.Submission.**<init>** (Integer, String, (	1,137 ms	(83.1%)
org.apache.pdfbox.text.PDFTextStripper.**getText** (org.apache.pdfbox.pdmc	966 ms	(70.6%)
org.apache.pdfbox.pdmodel.PDDocument.**load** (java.io.InputStream)	149 ms	(10.9%)
cz.cvut.fel.czm.plagcheckr.model.internal.Submission.**lemmatizateText** ()	16.7 ms	(1.2%)
Self time	2.37 ms	(0.2%)
cz.cvut.fel.czm.plagcheckr.model.internal.Submission.**lambdanew0** (cz.c	1.58 ms	(0.1%)
org.apache.pdfbox.text.PDFTextStripper.**<init>** ()	0.451 ms	(0%)
org.apache.pdfbox.pdmodel.PDDocument.**close** ()	0.247 ms	(0%)
cz.cvut.fel.czm.plagcheckr.model.internal.SubmissionTokenList.**<init>** ()	0.0 ms	(0%)
Self time	0.018 ms	(0%)
cz.cvut.fel.czm.plagcheckr.model.internal.Submission.**updateComparisons** (boc	228 ms	(16.7%)
Self time	0.156 ms	(0%)

Obrázek 5.3. Výsledek profilování služby aplikací *VisualVM*[84].

Ve sloupci vlevo je vždy napsán plný název funkce – tedy včetně plného názvu třídy, ve kterém se nachází, a vpravo sloupec se samotnou dobou volání. Ta je pro přehlednost barevně rozlišena na absolutní (vlevo) v milisekundách, a relativní (vpravo) v procentech – tato doba je důležitější; pro účel *profilování* nejsou absolutní časy reprezentativní, jelikož běh profileru celou aplikaci zpomaluje. Zároveň se s relativními časy mnohem lépe pracuje při případné analýze; snadněji vidíme nejnáročnější část kódu.

Modrou barvou jsou zde zvýrazněny dvě hlavní fáze analýzy textu, jak byly popsány v podsekci 2.2; vrchní *předzpracování textu* a spodní *porovnání* s ostatními soubory. Jak je možné vidět, samotné předzpracování textu zabírá většinu času (zde 83% doby zpracování požadavku) i v případě větších tasků (zde byl test proveden na výše zmíněném vzorku 67 souborů, ke kterým byl přidán další). Tato fáze je následně rozložena na další volání uvnitř této funkce; ta jsou zvýrazněna zelenou barvou.

Samotná lemmatizace textu zabírá pouze 1% doby běhu celého požadavku; necelých 82% doby zabírá načtení a následná extrakce textu z PDF souboru, poskytnutá knihovnou *Apache Lucene*. V případě potřeby rychlejšího běhu služby je tak nahrazení této knihovny, případně i vlastní implementace, pravděpodobnou cestou.

5.2.3 Parametry mapování textů

V implementační části práce byla zmíněna přítomnost parametrů, které ovlivňují výsledné mapování mezi texty nahraných souborů:

- **krátkodobá chyba** – počet slov v řadě, která se neshodují;
- **dlouhodobá chyba** – celkový počet slov v aktuálním úseku, která se neshodují;
- **minimální délka** – minimální délka úseku, který může být uložen.

Stanovení minimální délky úseku bylo nejsnazší – po několika testech s poskytnutými soubory byl tento parametr ustálen na hodnotě 4 – jde o dostatečný počet slov, aby šlo o relevantní záznamy (přibližně polovina věty), ale zároveň není příliš vysoký, aby propouštěl části upraveného plagiátu.

Parametry krátkodobé a dlouhodobé chyby se ukázaly jako obtížnější k určení – jejich chování je sice deterministické, ale přizpůsobení pro různé styly plagiátorství je poměrně složité. Hledání optimálních hodnot zahrnovalo testování většiny kombinací výše uvedeného vzorku (kterých je celkem $49 \cdot 48 = 2532$). Nakonec byly kompromisem mezi nízkou a vysokou citlivostí určeny hodnoty pro krátkodobou chybu max. 2 chybná slova v řadě, a pro dlouhodobou chybu 10% v daném úseku.

Krátkodobá chyba může být poměrně nízká – v nejhorším případě algoritmus nezachytí opsaný text jako jeden celistvý úsek, ale jako dvě či více oddělených části. Větším kompromisem je však dlouhodobá chyba – zde byla citlivost více patrná. Poměr maximálně jednoho chybného slova na 10 shodných by však měl vystačit na většinu případů.

Budoucí možnosti rozšíření funkcionality služby (a s ní spojeného zásuvného Moodle modulu) je přidání možnosti citlivosti vyhledávání – ideálně přednastavenými úrovněmi. Uživatel by tak měl možnost přizpůsobit analýzu dle svých potřeb.

5.2.4 Parametry vah podobností

Jak bylo předesláno v podsekci 4.3.4, vyučující má možnost nastavit váhy jednotlivých numerických metrik, za pomocí kterých je následně spočtena tzv. *vážená podobnost*, která je zobrazena v celkovém přehledu odevzdaných řešení. Tyto váhy se dají přizpůsobit konkrétní úloze a použití; existuje zde několik pravidel, kterými by se měl uživatel řídit.

Vzhledem k rozdílným přístupům k výpočtu podobností by uživatel měl nejprve určit váhu asymetrické podobnosti a sumu vah obou kosinových podobností. Asymetrická podobnost je díky svému výpočtu více vhodná pro texty s bohatší slovní zásobou, jako například slohová cvičení. Naopak kosinová podobnost se díky práci s informací o četnosti výskytu jednotlivých slov více hodí pro technické texty, obsahující odborné pojmy (obsahují-li tak dva texty právě 10× stejné slovo, jedná se o poměrně podezřelý jev).

Váha kosinové podobnosti se stopslovy by měla být téměř vždy nastavena na nižší hodnoty, než váha kosinové podobnosti bez stopslov. Důvodem jsou právě zahrnutá stopslova, která se často opakují, a mohla by tak vyvolávat falešný dojem podezření. Vyjimkou jsou velmi krátké texty (do cca 100 slov), u kterých se opakování slov v textu tolik nevyskytuje; stopslova tak mohou analýze přidávat větší kontext. Zároveň u velmi dlouhých textů (více než 2000 slov) se díky velké slovní zásobě hodnoty obou kosinových podobností blíží stejné hodnotě. Výše zmíněné počty slov však nejsou pevné hranice, ale slouží spíše pro orientační přehled – konkrétní hranice záleží na charakteru textu.

Na základě těchto poznatků jsou vyučujícím nabídnuty vzorová nastavení vah asymetrické podobnosti (A), kosinové podobnosti bez stopslov (KB) a kosinové podobnosti se stopslovy (KS) dle typu úlohy:

- **slovní úlohy, 1 strana A4** – `{A = 0,35; KB = 0,35; KS = 0,3}`;
- **slohové cvičení, 2 strany A4** – `{A = 0,5; KB = 0,2; KS = 0,3}`;
- **slohové cvičení, 5 stran A4** – `{A = 0,5; KB = 0,22; KS = 0,28}`;
- **protokol laboratorní práce** – `{A = 0,45; KB = 0,2; KS = 0,35}`.

Kapitola 6
Závěr

V této bakalářské práci bylo úspěšně navrženo, implementováno, a následně i otestováno řešení detekce plagiátorství pro systém *Moodle FEL ČVUT*, v podobě externí služby napsané v jazyce *Java*, a zásuvného modulu do aplikace *Moodle*, který se službou komunikuje. Modul službě předává data (v podobě PDF souborů s obsaženým textem), a nazpět dostává informace o podobnosti obsahu těchto souborů. Tato podobnost je vyjádřena jak numericky, tak i graficky (v podobě pozic podobných částí textů).

Nejprve byla rozebrána teorie ohledně plagiátorství – nejen její definice, ale i současné způsoby její prevence na *FEL ČVUT*. Bylo tak názorně ukázáno na dvou příkladech – *offline* zkoušky v učebně, a zkoušky za použití výukového systému, jakým může být například *Moodle FEL ČVUT*. Posléze se práce detailně zabývala problematikou zkoumání podobností dvou textů – od přípravy textu k analýze (pro zvýšení její účinnosti), až po samotné porovnání, které následně lze vyjádřit jak numericky, tak i graficky. Nechybělo ani podrobné rozebrání vícero metod, jak numerické hodnoty vypočíst.

Dále byl představen návrh řešení zadaného problému. Na základě aktivit (používaných v *Moodle FEL ČVUT*) a příslušných statistik o jejich využití bylo rozhodnuto, že se výsledné řešení bude zabývat *PDF* soubory, ze kterých bude posléze extrahován text. Na základě této informace byla navrhnuta architektura celého systému, včetně datových modelů služby a zásuvného modulu. Pro názornost byla taktéž uvedena řešení třetích stran, zabývajících se tímto problémem – jak s integrací do aplikace *Moodle*, tak i samostatné systémy.

Následně byla popsána implementace navrženého řešení – od výběru použitých technologií, až po popis hierarchie zdrojového kódu externí služby i zásuvného modulu. Nechyběl ani popis uživatelských obrazovek, se kterými se setká vyučující při používání tohoto modulu. Zároveň zde byly sepsány problémy, které bylo třeba v rámci implementace vyřešit; ať už šlo o limitace použitých knihoven, či aplikace *Moodle* jako takové.

Nakonec se práce věnovala provozu výsledného řešení – byla zde popsána současná infrastruktura systému *Moodle FEL ČVUT*, s přihlédnutím na nasazení zmíněné externí služby. Řešení bylo posléze otestováno nejen za pomocí jednotkových testů (s pokrytím okolo 64%), ale i na poskytnutých datech – byla odzkoušena rychlost celé analýzy, i vyladění některých jejích parametrů. Tímto byl splněn i poslední bod cílů, definovaných v zadání této práce.

Budoucí práce na implementovaném řešení by měla zahrnovat získání a zapracování zpětné vazby od vyučujících, a následné uveřejnění celého systému na produkční instanci *Moodle FEL ČVUT* na začátku akademického roku 2023/2024. V případě zájmu ze strany fakulty by možná rozšíření mohla zahrnovat pokrytí více aktivit či podporu ručně psaného textu za pomocí *OCR*. Zároveň může být systém nabídnut i celoškolské instanci *Moodle-vyuka*, potažmo i jiným vysokým školám v České republice. Budoucí rozšíření by zároveň mohlo brát v potaz situaci s používáním generativní umělé inteligence, jakou je například *ChatGPT*[85] americké společnosti *OpenAI*[86], která rapidně získala na popularitě na přelomu roku 2023 – i proto bohužel tato práce neměla možnost použití umělé inteligence studenty pro účely plagiátorství zohlednit včas.

Literatura

[1] *Moodle - Open-source learning platform — Moodle.org* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodle.org/

[2] VMWare Inc. *Spring Boot* [online]. Dostupné z: https://spring.io/projects/spring-boot

[3] *Moodle FEL ČVUT — Předměty na jednom místě* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodle.fel.cvut.cz/

[4] Bureš, Ondřej. *RPAPS 2021 - Rozšíření Moodle FEL - 2021*. Dostupné z: https://campuscvut.sharepoint.com/:w:/s/FEL-13393/CZM/PHPdev/EcntkJxLY_tJpwCAZLDmqNQB_LU3GpjAtj5S5cvJYM1MIA?e=9DfyZq

[5] *Institucionální program - Veřejný web - České vysoké učení technické v Praze* [online]. [cit. 12. 05. 2023]. Dostupné z: https://www.cvut.cz/rozvoj/institucionalni-program

[6] *Scio - Národní srovnávací zkoušky - Nezávislé a spravedlivé přijímačky* [online]. [cit. 11. 05. 2023]. Dostupné z: https://www.scio.cz/nsz/

[7] *Proctortrack — Trusted Exam Integrity — Remote Online Proctoring* [online]. [cit. 11. 05. 2023]. Dostupné z: https://www.proctortrack.com/

[8] Swauger, Shea. *Software that monitors students during tests perpetuates inequality and violates their privacy* [online]. Dostupné z: https://www.technologyreview.com/2020/08/07/1006132/software-algorithms-proctoring-online-tests-ai-ethics/

[9] Lupton, Andrew. *Western students alerted about security breach at exam monitor Proctortrack* [online]. Dostupné z: https://www.cbc.ca/news/canada/london/western-students-alerted-about-security-breach-at-exam-monitor-proctortrack-1.5764354

[10] Petráček, Vojtěch. *PŘÍKAZ REKTORA č. 04/2018 k ochraně a zpracování osobních údajů na ČVUT* [https://www.cvut.cz/sites/default/files/content/a28c8f76-aca0-400d-a8c4-cfd2c76489e4/cs/20181002-prikaz-rektora-c-042018-k-ochrane-a-zpracovani-osobnich-udaju-na-cvut.pdf].

[11] *BRUTE* [online]. [cit. 11. 05. 2023]. Dostupné z: https://cw.felk.cvut.cz/brute/student/

[12] Votroubek, Tomáš. *Improving Plagiarism Detection*. ČVUT v Praze, Fakulta elektrotechnická, 2018. Bakalářská práce. Dostupné z: http://hdl.handle.net/10467/73914

[13] pajecawav. *moodle/moodle — ghloc* [online]. Dostupné z: https://ghloc.vercel.app/moodle/moodle?branch=MOODLE_311_STABLE

[14] Heer, Niklas. *Latest speed comparison results* [online]. Dostupné z: https://niklas-heer.github.io/speed-comparison/

[15] *Releases - MoodleDocs* [online]. Dostupné z: https://docs.moodle.org/dev/Releases#Moodle_1.0

[16] *Plagiátorství — Masarykova univerzita* [online]. Dostupné z: https://www.muni.cz/o-univerzite/uredni-deska/plagiatorstvi

[17] Valášek, Lukáš a Klézl, Tomáš. *Šéf rady: Obviněním z plagiátorství se ÚSTR pokouší očernit svého nového ředitele - Aktuálně.cz* [online]. [cit. 15. 05. 2023]. Dostupné z: https://zpravy.aktualne.cz/r~45ce31e4c4a511ec8d900cc47ab5f122

[18] Snapes, Laura. *Taylor Swift to face plagiarism trial over Shake It Off lyrics* [online]. Dostupné z: https://www.theguardian.com/music/2021/dec/10/taylor-swift-to-face-plagiarism-trial-over-shake-it-off-lyrics

[19] Kottová, Anna a ČTK. *Bývalý prorektor Kovář se dopustil plagiátorství, rozhodla etická komise filozofické fakulty* [online]. Dostupné z: https://www.irozhlas.cz / zpravy-domov / martin-kovar-plagiat-uk-historik-prorektor-univerzity-karlovy_1902141407_ako

[20] *Safe Exam Browser - News* [online]. [cit. 11. 05. 2023]. Dostupné z: https://safeexambrowser.org/news_en.html

[21] Knuth, Donald E. *The Art of Computer Programming*. Addison Wesley Publishing Co, 1968. ISBN 9780201038019.

[22] *Plagiarism API — Moodle Developer Resources* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodledev.io/docs/apis/subsystems/plagiarism

[23] *Ouriginal: Text-matching solution - Plagiarism prevention* [online]. [cit. 11. 05. 2023]. Dostupné z: https://www.ouriginal.com/

[24] *Empower Students to Do Their Best, Original Work — Turnitin* [online]. [cit. 11. 05. 2023]. Dostupné z: https://www.turnitin.com/

[25] Pytelka, Petr. *Jak kvalita lemmatizace ovlivňuje výsledky vyhledávání dokumentů v českém jazyce*. Praha: Vysoká škola ekonomická v Praze, 2012. Diplomová práce.

[26] *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. [cit. 12. 05. 2023]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000665&local_base=KTD

[27] *WordNet* [online]. [cit. 11. 05. 2023]. Dostupné z: https://wordnet.princeton.edu/

[28] Rambousek, Adam, Horák, Aleš a Pala, Karel. Sustainable long-term WordNet development and maintenance: Case study of the Czech WordNet. *Cognitive Studies*. 12, 2018, ročník 2018. Dostupné z DOI: 10.11649/cs.1715.

[29] Blahuš, Marek. *Extending Czech WordNet Using a Bilingual Dictionary [online*. Masarykova univerzita, Fakulta informatikyBrno, 2011 [cit. 2023-05-12. Diplomová práce. Dostupné z: https://theses.cz/id/lvxu9j/ SUPERVISOR: prof. PhDr. Karel Pala, CSc..

[30] Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*. feb, 1966, ročník 10, s. 707.

[31] *Assignment activity* [online]. Dostupné z: https://docs.moodle.org/401/en/Assignment_activity

[32] *Forum activity* [online]. Dostupné z: https://docs.moodle.org/401/en/Forum_activity

[33] *Quiz activity* [online]. Dostupné z: https://docs.moodle.org/401/en/Quiz_activity

[34] *Plagiarism plugins - MoodleDocs* [online]. [cit. 11. 05. 2023]. Dostupné z: https://docs.moodle.org/dev/Plagiarism_plugins

[35] *Odevzdej.cz - Odhalování plagiátů v seminárních pracích* [online]. [cit. 11. 05. 2023]. Dostupné z: https://odevzdej.cz/

[36] Brandejsová, Jitka, Brandejs, Michal a Suchomel, Šimon. *Dvanáctileté zkušenosti z provozu systémů na odhalování plagiátů* [online]. Dostupné z: https://is.muni.cz/clanky/2018_odhalovani_plagiatu

[37] *Theses.cz - Vysokoškolské kvalifikační práce* [online]. [cit. 11. 05. 2023]. Dostupné z: https://theses.cz/

[38] Borkovcová, Anna. *Kontrola plagiátů Odevzdej.cz v Moodle* [online]. Dostupné z: https://online.upce.cz/kontrola-plagiatu-odevzdejcz-v-moodle

[39] *About Us — About Turnitin, Our Mission & Values — Turnitin* [online]. Dostupné z: https://www.turnitin.com/about

[40] *Ouriginal by Turnitin — Turnitin* [online]. Dostupné z: https://www.turnitin.com/products/ouriginal

[41] Sklenár, Michal. *Návrh a implementace modulu antiplagiát pro Moodle FIT*. ČVUT v Praze, Fakulta informačních technologií, 2016. Diplomová práce.

[42] *Moodle-výuka* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodle-vyuka.cvut.cz/

[43] Boone, Kevin. Developing micro-microservices in C on Red Hat OpenShift. 2020. Dostupné z: https://developers.redhat.com/blog/2020/08/27/developing-micro-microservices-in-c-on-red-hat-openshift

[44] JetBrains s.r.o. *Microservices* [online]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2021/microservices/

[45] Wulf, Josh. 7 Best Programming Languages for Microservices. Dostupné z: https://camunda.com/blog/2022/09/seven-best-programming-languages-for-microservices/

[46] *Spring Framework* [online]. [cit. 11. 05. 2023]. Dostupné z: https://spring.io/projects/spring-framework

[47] Rykov, Mikhail. Is Kotlin Faster Than Java?. 2022. Dostupné z: https://www.baeldung.com/kotlin/kotlin-java-performance

[48] Medvedev, Mikhail. *Building a microservice with Rust* [online]. Dostupné z: https://medium.com/tenable-techblog/building-a-microservice-with-rust-23a4de6e5e14

[49] *Apache PDFBox — A Java PDF Library* [online]. [cit. 11. 05. 2023]. Dostupné z: https://pdfbox.apache.org/

[50] *pdf-extract — Rust text processing library // Lib.rs* [online]. [cit. 11. 05. 2023]. Dostupné z: https://lib.rs/crates/pdf-extract

[51] *Using web services - MoodleDocs* [online]. [cit. 11. 05. 2023]. Dostupné z: https://docs.moodle.org/401/en/Using_web_services

[52] Fielding, Roy Thomas a Taylor, Richard N. *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine, 2000. Disertační práce. AAI9980887.

[53] Jarý, Miroslav. *[MDL-77683] Incorrect usage of HTTP codes in webservices - Moodle Tracker* [online]. Dostupné z: https://tracker.moodle.org/browse/MDL-77683

[54] carusogabriel. *PHP: rfc:unbundle_xmlprc* [online]. [cit. 15. 05. 2023]. Dostupné z: https://wiki.php.net/rfc/unbundle_xmlprc

[55] Ingerson, Brian, Evans, Clark C. a Ben-Kiki, Oren. *Yet Another Markup Language (YAML) 1.0* [online]. [cit. 15. 05. 2023]. Dostupné z: https://yaml.org/spec/history/2001-08-01.html

[56] *Apache Lucene - Welcome to Apache Lucene* [online]. [cit. 11. 05. 2023]. Dostupné z: https://lucene.apache.org/

[57] *Class Hierarchy (Lucene 9.4.2 common API)* [online]. Dostupné z: https://lucene.apache.org/core/9_4_2/analysis/common/overview-tree.html

[58] *MapStruct – Java bean mappings, the easy way!* [online]. [cit. 11. 05. 2023]. Dostupné z: https://mapstruct.org/

[59] *PDF.js* [online]. [cit. 11. 05. 2023]. Dostupné z: https://mozilla.github.io/pdf.js/

[60] *Apache License, Version 2.0* [online]. [cit. 13. 05. 2023]. Dostupné z: https://www.apache.org/licenses/LICENSE-2.0.html

[61] *Centrum znalostního managementu FEL ČVUT* [online]. [cit. 11. 05. 2023]. Dostupné z: https://czm.fel.cvut.cz/cs/

[62] *CZM / Services / Microservices archetype ■ GitLab* [online]. [cit. 11. 05. 2023]. Dostupné z: https://gitlab.fel.cvut.cz/czm/services/microservices-archetype

[63] *The Collaborative API Development Platform - Insomnia* [online]. [cit. 11. 05. 2023]. Dostupné z: https://insomnia.rest/

[64] *Plugin types — Moodle Developer Resources* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodledev.io/docs/apis/plugintypes

[65] *Why AMD?* [online]. [cit. 11. 05. 2023]. Dostupné z: https://requirejs.org/docs/whyamd.html

[66] *Installing Moodle/ - MoodleDocs* [online]. [cit. 11. 05. 2023]. Dostupné z: https://docs.moodle.org/402/en/Installing_Moodle#Create_an_empty_database

[67] *XMLDB Documentation - MoodleDocs* [online]. [cit. 11. 05. 2023]. Dostupné z: https://docs.moodle.org/dev/XMLDB_Documentation

[68] *Moodle in English: Moodle and his way of handling databases* [online]. [cit. 11. 05. 2023]. Dostupné z: https://moodle.org/mod/forum/discuss.php?d=445096

[69] *PHP: cURL - Manual* [online]. [cit. 11. 05. 2023]. Dostupné z: https://www.php.net/manual/en/book.curl.php

[70] The PHP Group. *PHP: stdClass - Manual* [online]. Dostupné z: https://www.php.net/manual/en/class.stdclass.php

[71] *OAuth 2.0 — OAuth* [online]. Dostupné z: https://oauth.net/2/

[72] *Document management – Portable document format.*

[73] *Czech Stemmer [LUCENE-2067* [online]. Dostupné z: https://github.com/apache/lucene/issues/3143

[74] Dolamic, Ljiljana a Savoy, Jacques. Indexing and Stemming Approaches for the Czech Language. *Inf. Process. Manage.* USA: Pergamon Press, Inc., nov, 2009, ročník 45, č. 6, s. 714-720. ISSN 0306-4573. Dostupné z DOI: 10.1016/j.ipm.2009.06.001. Dostupné z: https://doi.org/10.1016/j.ipm.2009.06.001

[75] *Dashboard — GitLab* [online]. [cit. 15. 05. 2023]. Dostupné z: https://gitlab.fel.cvut.cz/dashboard/projects/starred

[76] *Git - Submodules* [online]. [cit. 15. 05. 2023]. Dostupné z: https://git-scm.com/book/cs/v2/Git-Tools-Submodules

[77] *Docker: Accelerated, Containerized Application Development* [online]. [cit. 15. 05. 2023]. Dostupné z: https://www.docker.com/

[78] *Swarm mode overview — Docker Documentation* [online]. Dostupné z: https://docs.docker.com/engine/swarm/

[79] *Home - PHP-FPM* [online]. [cit. 15. 05. 2023]. Dostupné z: https://php-fpm.org/

[80] *nginx* [online]. [cit. 15. 05. 2023]. Dostupné z: https://nginx.org/en/

[81] *PgBouncer - lightweight connection pooler for PostgreSQL* [online]. [cit. 15. 05. 2023]. Dostupné z: https://www.pgbouncer.org/

[82] *CI/CD pipelines — GitLab* [online]. Dostupné z: https://docs.gitlab.com/ee/ci/pipelines/

[83] *JUnit 5* [online]. Dostupné z: https://junit.org/junit5/

[84] *VisualVM: Home* [online]. [cit. 15. 05. 2023]. Dostupné z: https://visualvm.github.io/

[85] *Introducing ChatGPT* [online]. Dostupné z: https://openai.com/blog/chatgpt

[86] *OpenAI* [online]. Dostupné z: https://openai.com/

Příloha A
Elektronická příloha

Elektronická příloha se sestává ze tří částí:

- **plagcheckr_service.zip** – *ZIP* archiv zdrojového kódu externí služby;
- **moodle_plagiarism_plagcheckr.zip** – *ZIP* archiv zdrojového kódu zásuvného modulu do aplikace *Moodle*;
- **benchmark.zip** – *ZIP* archiv zdrojových kódů testovacích skriptů v jazycích *Java* a *Rust*, použitých pro ověření rychlosti extrakce textu z *PDF* souboru (viz podsekce 4.1.1).

Zdrojový kód externí služby i zásuvného modulu byl po celou dobu implementace verzován za pomocí nástroje *git*, a uložen na fakultní instanci aplikace *GitLab*.