Mendelova univerzita v Brně
Provozně ekonomická fakulta

Rozpoznávání podobností zdrojových kódů v systému Anton

Bakalářská práce

Vedoucí práce:
Ing. Dita Dlabolová

Richard Všianský

Brno 2017

Chtěl bych poděkovat své vedoucí práce Ing. Ditě Dlabolové za odborné vedení práce a cenné rady, které mi pomohly práci napsat. Dále bych rád poděkoval Mgr. Tomáši Foltýnkovi, Ph.D. za pomoc při zprovoznění systému Anton. Také bych chtěl poděkovat Ing. Jiřímu Lýskovi, Ph.D. za poskytnutí informací a dat pro mou práci.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Rozpoznávání podobností zdrojových kódů v systému Anton**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 21. května 2017 _____

Abstract

Všianský, R. Recognition of similarities in source codes in the system Anton. Bachelor thesis. Brno: Mendel University, 2017.
The bachelor thesis deals with the implementation of an extension to the system Anton, which can recognize similarities across source codes in PHP. The basis for the extension is the algorithm Greedy String Tiling. The solution shows the results of comparison both numerically and graphically. The solution was tested on illustrative data as well as actual data from teaching.

Keywords

Plagiarism, source codes, automated system, PHP, Greedy String Tiling.

Abstrakt

Všianský, R. Rozpoznávání podobností zdrojových kódů v systému Anton. Bakalářská práce. Brno: Mendelova univerzita v Brně, 2017.
V bakalářské práci je řešena implementace rozšíření do systému Anton, které dokáže rozpoznat podobnosti napříč zdrojovými kódy v jazyce PHP. Podstatou rozšíření je algoritmus Greedy String Tiling. Řešení ukazuje výsledky porovnání číselně i graficky. Řešení bylo otestování na ilustrativních datech i na skutečných datech pocházející z výuky.

Klíčová slova

Plagiátorství, zdrojové kódy, automatizovaný systém, PHP, Greedy String Tiling.

Obsah

1 Úvod

Rozpoznávání podobností v různých typech studentských prací je v současné době na mnoha školách součástí kontrolního procesu po odevzdání a před hodnocením dané práce. Existuje několik automatizovaných systémů, které jsou používány napříč univerzitami. Jedná se například o systém Theses, který vyvíjí a spravuje Masarykova univerzita v Brně a který používá i Mendelova univerzita v Brně. Mendelova univerzita vytvořila a nějaký čas provozovala i vlastní systém Anton – the Antiplagiator Online Solution, který ale následně ustoupil právě zmíněnému systému Theses.

Potřebný je však i systém, který zvládne kontrolovat práce odevzdané ve formě zdrojového kódu, jelikož výše zmíněné systémy pracují pouze s textovými soubory, jejichž porovnání se řídí odlišnými pravidly. Standardní metody, které používají běžné automatizované systémy navíc text porovnávají z hlediska řetězců znaků (Chýla, 2009), na zdrojový kód je ale nutné nahlížet z jiného úhlu pohledu, navíc existuje obrovské množství různých jazyků, jejichž různorodost proces více komplikuje. Okopírování zdrojového kódu od jiného studenta je na druhou stranu velice snadné a jednoduchými modifikacemi lze zajistit rozdílnou podobu výsledného kódu.

V následujícím textu se budeme zabývat problematikou metod k rozpoznávání podobností ve zdrojových kódech a vytvoření vlastního modulu do systému Anton, který je v současné době opět vyvíjen a nasazen zkušebně na Mendelově univerzitě, kde vlastní systém pro rozpoznávání podobností počítačových programů chybí. Jako programovací jazyk, ve kterém se budou podobnosti odhalovat, byl vybrán jazyk PHP, který je velice rozšířen a na univerzitě se vyučuje napříč více předměty.

2 Cíl práce

Cílem mé bakalářské práce je rozšíření antiplagiátorského systému Anton o možnost rozpoznávat a odhalovat podobnosti napříč programovými kódy napsaných v jazyce PHP a nasazení tohoto rozšíření do provozu na online serveru pro uživatele na Mendelově univerzitě, kde bude sloužit pro kontrolu studentských projektů a prací.

Během práce proběhne analýza současného stavu systému a jeho funkcionality, a používaných metod sloužících pro odhalování plagiátorství ve zdrojových kódech.

Řešení bude využívat současně zavedených technologií systému – framework Nette, MySQL databázi, jazyků HTML a PHP a kaskádových stylů. Řešení bude provozováno na unixovém serveru. Součástí řešení bude vytvoření a zavedení uživatelského rozhraní modulu do systému, respektující jeho nynější a zavedenou podobu. Následně bude otestována funkčnost rozpoznávání a odhalovaní podobností, a dojde ke kritickému ohodnocení nového modulu po technické a ekonomické stránce.

3 Strojové odhalování podobností

3.1 Plagiátorství a strojové vyhledávání podobností

Je důležité rozlišovat mezi plagiátem a pouhou strojovou podobností. V případě plagiátu se jedná o využití cizího díla nebo myšlenek, bez jejich řádného ocitování a bez uvedení příslušného zdroje. Jak uvádí Floryček (2015), hranice mezi plagiátem a parafrází/kompilací je velice tenká a často těžce určitelná. Je to jeden z důležitých faktorů, který je nutný při strojovém vyhledávání podobností brát v úvahu.

Podle autora systému Moss, Alexa Aikena (MOSS,2017) ze Stanfordské univerzity, dokáže systém sice rozpoznat podobné programy, ale už nedokáže určit, proč si jsou podobné. Tyto programy by tedy měly sloužit pouze k tomu, aby ušetřily kontrolujícímu uživateli čas, a nechat výsledné rozhodnutí na něm. Floryček (2015) dále ve své práci uvádí důvody plagiátorství mezi studenty, definuje a rozděluje jeho typy a doporučuje metody prevence.

3.2 Systém Anton

Systém Anton je antiplagiátorský systém vyvinut na Mendelově univerzitě, který byl nasazen v roce 2009 a jeho provoz trval až do roku 2014, kdy byl nahrazen systémem Theses. Hlavním důvodem změny systému byla u nového systému sdílená databáze prací napříč univerzitami, zatímco Anton prohledával dokumenty pouze ve své vlastní lokální databázi, takže systém nedokázal odhalovat shody textu ze zdrojů mimo univerzitu, což je v době přístupnosti prací na internetu velice potřebné (Foltýnek et al., 2009).

3.2.1 Znovu nasazení Antona a jeho aktuální stav

Jak uvádím ve svém příspěvku *„Deployment and improvements of system Anton“* z roku 2016, Anton se dočkal téhož roku znovu nasazení a obnovení funkcionality na adrese přístupné univerzitním uživatelům: `https://anton.mendelu.cz`, kde lze po požádání získat uživatelské údaje a systém používat. Systém se dočkal také několika změn a vylepšení.

3.3 Algoritmus systému Anton

3.3.1 Dekódování

Systém Anton porovnává v pracích textové dokumenty pěti rozdílných formátů: TXT, HTML, PDF, DOC a DOCX (Všianský a Dlabolová, 2016). Po vyplnění potřebných údajů (autor, název, rok organizace apod.) se daný soubor nahraje na server a tam se každé dvě minuty spustí proces, který je rozdělen na čtyři fáze. Soubor se nejprve dekóduje a převede do textového formátu, aby se vyloučily rozdíly různých formátů (Floryček, 2015).

3.3.2 Předzpracování

Následně proběhne proces předzpracování, kde se text upravuje tak, aby z něj byly vyjmuty všechny přebytečné informace. To zahrnuje operace:

- odstranění národních znaků,
- zmenšení všech písmen,
- nahrazení znaků nových řádků a tabulátorů za mezery,
- odstranění číslování stránek,
- odstranění všech znaků, co nejsou písmena,
- odstranění slov kratších než tři písmena,
- odstranění chybových hlášek vzniklých při dekódování (Anton, 2017).

Krom výše zmíněných systém obsahuje ještě další tři komplexnější operace v této fázi. Jedná se o záměnu synonym, o vyjmutí stop slov a kontrola délky slov (Všianský a Dlabolová, 2016). Kontrola synonym kontroluje každé slovo v textu s databází slov na serveru a v případě shody toto slovo v textu nahradí za základní slovo, ke kterému je odkazováno ze slova ve slovníku (Všianský a Dlabolová, 2016). Slovník lze upravovat přímo v uživatelském prostředí systému (Anton, 2017).

Dalšími slovy k odstranění jsou výše zmíněná stop slova. Procházka (2012) zmiňuje, že to jsou slova, která nenesou žádný obsahový význam, například předložky, spojky atd. (Floryček, 2015). Také to jsou slova, která se v kontextu dané práce budou často opakovat, jako například název univerzity (Všianský a Dlabolová, 2016). Tato slova jsou ručně zadávána do databáze systému (Anton, 2017).

Poslední úpravou v této fázi je detekce podezřelých znaků mezi slovy. Tato metoda spočívá v tom, že autor vloží do práce znaky, které jsou stejné barvy jako barva pozadí dokumentu a v tom případě takto spojená slova vytvoří nesmyslné řetězce (Floryček, 2015). Systém tento problém řeší počítáním délky slov a v případě, že počet podezřele dlouhých slov překročí stanovenou hranici, je uživatel na tuto situaci upozorněn a je na jeho vlastním prozkoumání, zda bylo s textem takto manipulováno (Všianský a Dlabolová, 2016).

3.3.3 Fáze tvorby hashů

V další fázi algoritmus postupně prochází celý text: každá osmice po sobě jdoucích slov je abecedně seřazena (tzv. chunk) a z této n-tice je vytvořen osmibytový hash (otisk) pomocí kryptografické funkce MD5, jenž z každého vstupu vytvoří rozdílný výstup (Rivest, 1992).

3.3.4 Fáze porovnání

Ve finální fázi algoritmu jsou všechny chunky vzájemně porovnány a systém zobrazí uživateli procentuální shodu s ostatními dokumenty, zároveň vše zanese do databáze. Tuto shodu systém zobrazí oboustranně, tedy jak shodu dokumentu A s dokumentem B, tak i procentuální shodu dokumentu B s dokumentem A.

Takto porovnané dokumenty si může uživatel prohlédnout s vyobrazením shodnosti v obou dokumentech.

V roce 2013 se Česká národní banka (dále jen ČNB) v čele s guvernérem Miroslavem Singerem rozhodla pomocí devízových intervencí změnit měnový kurz koruny tak, aby koruna vůči euru depreciovala nad stanovenou hranici (ČNB, 2013). Hlavním cílem tohoto kroku byl růst inflace přes 2 % a zabránění deflace a s ní spojeným pádem do deflační spirály. Taková situace nastává tehdy, když se v ekonomice vyskytuje příliš mnoho zboží a malý počet peněz. To vede k tomu, že cenová hladina klesá. (Finance.cz , 2004) Pro spotřebitele se jedná většinou o dobrou zprávu, za stejný obnos mohou zpočátku koupit více zboží. Na druhé straně však dochází ke snižování tržeb a zisků prodejců, snižují se náklady a postupem času může dojít až na propouštění zaměstnanců. Lidé však očekávají pokračování deflace a nižší ceny, a odkládají svou spotřebu. To celou situaci zhoršuje (Ludwig von Mises Institut, 2013).

---

ceska narodni banka dale jen cnb cele guvernerem miroslavem singerem rozhodla pomoci devizovych intervenci zmenit menovy kurz koruny tak aby koruna vuci euru depreciovala stanovenou hranici cnb hlavnim cilem tohoto kroku byl rust inflace pres zabraneni deflace spojenym padem deflacni spiraly takova situace nastava kdyz ekonomice vyskytuje prilis mnoho zbozi maly pocet penez vede tomu cenova hladina klesa finance pro spotrebitele jedna vetsinou dobrou zpravu stejny mohou zpocatku koupit vice zbozi druhe strane vsak dochazi snizovani trzeb zisku prodejcu snizuji naklady postupem casu muze dojit propousteni zamestnancu vsak ocekavaji pokracovani deflace nizsi ceny odkladaji svou spotrebu celou situaci zhorsuje ludwig von mises institut

---

Obrázek 1 Ukázka proměny textu v systému Anton, nahoře je originální text, dole je finální verze textu prošlá všemi úpravami.

3.4 Ostatní vlastnosti systému Anton

Systém nerozděluje uživatele do rolí, ale každý uživatel dostane přiřazeno právo k určitým funkcím jako administrátor dokumentů, jazyků, povolení nahrávat dokumenty. Lze tak každému uživateli vytvořit přesně daný přístup na míru (Anton, 2017).

Další důležitou vlastností jsou pravidla (rules), které umožňují nastavit několik základních parametrů systému, výčtově jde o:

- maximální velikost nahraného souboru,
- nastavení nahrazování synonym,
- nastavení délky podezřele dlouhých slov,
- nastavení minimálního počtu podezřele dlouhých slov (Všianský, Dlabolová, 2016).

Tato pravidla lze nastavit pro celý systém nebo pouze pro danou organizaci. Organizace sdružuje uživatele a dokumenty jedné společnosti, například jedné univerzity.

Systém je postaven na Nette, českým PHP frameworku, a využívá i PHP backend k realizaci algoritmu, databázová část je obstarána MySQL. Potřebný skript k fungování algoritmu se nachází ve složce `app/backend/do-all`, který následně spouští všechny další nutné soubory (Anton, 2017).

3.4.1 Framework Nette

Framework Nette je český PHP framework, který slouží ke tvorbě webových aplikací. Jednou z jeho součástí je šablonovací systém Latte, který umožňuje rychlou tvorbu webových stránek a zajišťuje jejich bezpečnost. Jeho poslední verze nese označení 2.4 (Nette Foundation, ©2017). Součástí distribuce Nette je i Adminer, nástroj pro správu databáze (Adminer, ©2017). V případě potřeby lze tedy databázi obsluhovat i bez příslušných možností v samotném systému Anton.

3.4.2 MVC architektura

Architektura webových aplikací vytvořených pomocí frameworku Nette je založena na model-view-controller přístupu. Jde o tří vrstvou architekturu, kde se první vrstva modelu stará o aplikační logiku. Grafické rozhraní je obsluhováno controllerem, v Nette je jejich obdobou presenter. Výsledná data zobrazuje uživateli view, tato vrstva využívá výše zmíněný šablonovací systém latte (Nette Foundation, ©2017).

3.4.3 Implementace nových částí

Takto vytvořené webové aplikace jsou velmi snadno rozšiřitelné. Presentery a modely mohou být sdruženy do společných adresářů, kterým se pak říká modul. V případě tvorby nové části aplikace je tedy nutné vytvořit presenter, modul a view s příslušnou funkcionalitou a vložit do daného adresáře. V této práci bude nutné zasáhnout i do zmíněného PHP backendu.

3.5 Vyhledávání podobností ve zdrojových kódech

Prvním bodem, který je nutný brát na vědomí, je způsob, kterým může docházet k vytváření podobností ve zdrojových kódech a jak je následně vyhledat. Podle Clougha (2000) je vyhledávání podobností ve zdrojových kódech snazší než vyhledávání podobností v přirozených jazycích. Jako důvod uvádí mnohem menší počet slov v povoleném slovníku. Takový slovník je podle něj konečný a specifikovaný, na rozdíl od slovníku přirozených jazyků, který je více komplexní a víceznačný. U porovnávání přirozeného jazyka mohou vytvářet problémy například homonyma, stejně vypadající slova s jiným významem, zatímco u programového kódu takové slova neexistují a každé klíčové slovo má přesně daný význam.

Clough (2000) zmiňuje některé transformace, kterými pozměněný program mohl při úpravě projít, například změnou komentářů nebo názvu proměnných, z těch složitějších změn Clough uvádí záměnu cyklů while za for. Dalšími možnými transformacemi kódu jsou změna pořadí operandů, přidávání nadbytečných částí kódů jako jsou prázdné proměnné nebo prázdné výpisy, změna iteračních cyklů za case (Whale, 2011). Takové změny jsou poměrně jednoduché a student pomocí nich může velmi rychle změnit podobu kódu, aniž by měnil jeho funkčnost.

Joy a Mirza (2015) zmiňují, že lze autorství kódu rozpoznat z jeho psané podoby, která vychází ze stylu kódování, který může být dán pokyny k danému jazyku nebo určen institucí, kde se kód píše: psaní komentářů, pojmenování proměnných, deklarace, organizace souborů a podobně.

Přestože automatizovaný systém najde podobnosti v kódu, musí uživatel daného systému dohlédnout nato, zda se opravdu jedná o plagiát. Zápis kódu a řešení zadání může ovlivnit i styl výuky, který vede studenty k jednotnému zápisu a podobnosti mohou při stejném zadání úkolu vzniknout zcela náhodně. Proto může být rozdíl mezi úmyslným plagiátem a náhodnou podobností malý a v některých případech nerozlišitelný.

3.6 Metody vyhledávání podobností ve zdrojových kódech

Ohno a Murao (2010) rozdělují metody, které se používají k vyhledávání podobností ve zdrojových kódech, do několika kategorií. Jako nejvíce používaný přístup označili takový typ metod, které měří strukturní, algoritmické a metrické podobnosti v kódu. Zmiňují další typ přístupu, který využívá „tokenizaci“ a normalizaci kódu, který se následně zobrazí v podobě grafu nebo v sekvenci tokenů. Shao (2015) token popisuje jako nejzákladnější jednotku jazyka a běžně se jedná o řetězec znaků. Jako příklad uvádí nahrazení všech čísel v kódu za ”INT“. Normalizace je taková série úkonů, při nichž jsou z kódu odstraněny komentáře, nadbytečné mezery a série bílých znaků (Floryček, 2015). Dalším běžným typem metod jsou takové, které generují vektor, jenž obsahuje informace o velikosti, komplexnosti specifických typů tokenů nebo jejich počty (Murao a Ohno, 2010).

Podle Prechelta (2000) se dělí metody na vyhledávání podobností do dvou kategorií: porovnání vlastnosti a struktury. První druh metod je popisován jako vzá-

jemné porovnávání vlastností kódů, mezi které patří počet tokenů, prázdných řádků, komentářů a průměrný počet znaků na stránku, zatímco druhý typ metod nejdříve vygeneruje sekvence tokenů a následně provede jejich porovnání (Arwin a Tahaghoghi, 2006).

3.6.1 Porovnání vlastností

Arwin a Tahaghoghi (2006) si berou za příklad Jonese (2001), který porovnává programy dle tří různých profilů:

- fyzický profil, který obsahuje fyzické atributy, například počet řádků, slov a znaků,
- halstead profil, jenž obsahuje výše zmíněné tokeny a jejich frekvenci – počet jednotlivých tokenů, počet rozdílných tokenů a jejich obsah,
- kompozitní profil je kombinace dvou výše zmíněných profilů.

Tyto profily jsou z programu vypočítány a následně normalizovány, finální podobnost je přibližně vyobrazena Eukleidovskou metrikou (Arwin a Tahaghoghi, 2006). Jak ale stejný autor uvádí, výsledky se mohou velice lišit, někdy je algoritmus příliš citlivý a někdy neodhalí ani zřejmé shody, jelikož je velmi jednoduché dané vlastnosti do programu vložit, například komentáře, které nemají na chod programu žádný vchod, mohou celý algoritmus porovnávání velice narušit a vlastnosti zkoumaného programu změnit.

3.6.2 Strukturní porovnání

Prechelt (2009) upřednostňuje tento typ metod, zvláště na silnějších počítačích. Jak dále zmiňuje, existují metody, které oba dva přístupy kombinují dohromady. Strukturní porovnání porovnává programy jako proudy tokenů, kde jsou nedůležité informace, jako komentáře a prázdné řádky, ignorovány (Prechelt, 2009).

3.7 Nástroje a programy pro vyhledávání podobností ve zdrojových kódech

V současné době existuje více programů pro řešení podobnost a odhalování potencionálního plagiátorství napříč zdrojovými kódy. V následujícím výčtu uvedu jen část těch nejznámějších.

3.7.1 MOSS

MOSS (Measure Of Software Similarity) patří mezi jedny z nejznámějších a nejpoužívanějších nástrojů pro odhalování podobností. Systém začal být vyvíjen v roce 1994 a v současné době je nabízen pro vzdělávací účely zcela zdarma ve formě Linuxového skriptu a emailového serveru MOSSu, který odesílá po spuštění skriptu výsledky porovnání (MOSS, 2017). MOSS podporuje řadu programovacích

a skriptovacích jazyků, jako například C, C++, Javu, C#, Javascript nebo i Matlab, Verilog, Perl a mnoho dalších (MOSS, 2017). Bohužel ale nepodporuje jazyk PHP.

MOSS používá metodu zvanou jako *„winnowing“,* do češtiny to lze přeložit jako „třídění“ nebo „prosívání.“ Tuto metodu popsal Aiken, Schleimer a Wilkerson ve své práci (2003): z textu jsou nejdříve odstraněny nerelevantní informace, z tohoto textu jsou vytvořeny k-znakové sekvence (k je zvoleno uživateli) a z těchto sekvencí jsou vygenerovány hashe, které jsou následně sdruženy do skupin o zvolené délce. Z těchto skupin je vybrán hash s nejmenší hodnotou, v případě shody, je vybrán ten nejvíce vpravo, tyto hashe pak tvoří takzvané otisky, které jsou nakonec popárovaný s informací o jejich pozici.

3.7.2 Jplag

Jplag je nabízený jako webová služba, která začala svůj vývojový cyklus v roce 1996 a podporuje jazyky C, C++, C#, Javu a SchemeR4S4 (Jplag, 2017). Jplag je založen na algoritmu YAP3, ale zefektivňuje ho (Prechelt, 2000). YAP3 od Wise nejdříve pročistí text a následně funkce převede na jejich základní ekvivalenty, poté je text ztokenizován, a ve druhé fázi algoritmu je porovnán (Arwin a Tahaghoghi, 2006). Hlavní výhodou dle tvůrců (Prechelt, 2000) oproti Mossu je uživatelské prostředí, které je plně v podobě HTML a pro uživatele příjemnější, další parametry už ale nemohli porovnat, protože nemají přístup ke spuštění MOSSu a k jeho detailnímu vyhodnocení.

Zdrojový kód je v případě této služby změněn na tokeny (využívá tedy strukturní porovnání), kde jsou tokeny uvedeny do kontextu programu (Arwin a Tahaghoghi, 2006). Ne každý prvek ve zdrojovém kódu je vhodný být vzorem pro token a pro rozdílné jazyky se tokeny různí (Prechelt, 2000). K následnému srovnání se používá „string tiling“, jehož cílem je najít množinu podřetězců, které jsou stejné a splňují autorem uvedená pravidla.

3.7.3 CloneDR

CloneDR neslouží ani tak k odhalování plagiátů, ale spíše k nalézání stejných částí kódů, které se opakují, a které by tak bylo vhodné mít uloženy pouze na jednom místě, aby se snížily náklady za údržbu, jelikož se při modifikaci nebo nalezení chyby již nemusí procházet všechny zdrojové soubory, kde se tyto části nachází (CloneDR, ©2016). Jedná se tedy o řešení vhodnější pro průmyslové použití než o systém vhodný pro univerzitní prostředí. CloneDR podporuje širokou škálu jazyků jako například C, C++, Foxtran ale i PHP (CloneDR, ©2016), čímž se liší od předchozích dvou popsaných nástrojů.

Program rozděluje kód na bloky, které mohou být od jednotlivých řádků až po celé soubory, a u nich identifikuje vlastnosti. Pokud se vlastnosti dvou bloků podobají, program je vyhodnotí jako klon. Jako algoritmus, který zajišťuje tento proces, je používáno srovnávání založené na abstraktním syntaktickém stromu (CloneDR, ©2016).

3.8 Přepisování mezi různými programovými jazyky

V dnešní době existuje ohromné množství různých programovacích jazyků a nabízí se možnost opisování kódu napříč nimi. Možnosti ještě zvyšuje rozvoj internetových programátorských komunit. Výše popsané metody slouží právě k porovnání kódu napsaného ve stejném jazyce, zatímco pro „inter-lingual“ plagiátorství je třeba najít a použít jiný přístup (Arwin a Tahaghoghi, 2006).

Jednou z metod je použití zprostředkujícího kódu (Arwin a Tahaghoghi, 2006). Na tomto principu je založen XPlag přístup (Arwin a Tahaghoghi, 2006). Tato metoda je dvoufázová, v první fázi pomocí tokenizace rozloží programy, které ještě předtím byly převedeny do zprostředkujícího kódu, a uloží informace o tokenech do invertovaných indexů, v další fázi jsou tyto informace porovnány a je vytvořen sestupný seznam s procentuálními údaji o podobnostech mezi dokumenty (Arwin a Tahaghoghi, 2006).

4 Metodika práce

4.1 Porovnání a tokenizace

Pro vypracování práce je využit algoritmus *"Greedy String Tiling"*, který popisuje Prechelt (2000) a jenž byl použit a nasazen v systému JPlag. Pro tento algoritmus vycházejícího z YAP3 je nejdříve nutné zajistit převod porovnávaného souboru ze standardní podoby na sekvenci tokenů. K tomu je použita funkce token_get_all, která je již defaultně obsažena v jazyce PHP (The PHP Group, ©2017). Její výhodou je jednoduché použití a pokrytí všech případů. Funkce umožňuje vypsat nejen kód převeden na tokeny, ale zvládá vypsat i jejich obsah a řádkovou lokaci v dokumentu, což je velice užitečné pro následné zpracování nalezené podobnosti.

Algoritmus přebírá takto přepsaný text a ten je v cyklu rozebrán dvěma fázemi:

1. algoritmus prochází každý neoznačkovaný token prvního dokumentu a srovnává s každým neoznačeným tokenem v druhém dokumentu. Jestliže je nalezenuta shoda, testuje se následující dvojice. Nejdelší sled shod je pak následně zaznamenán.
2. Zaznamenané shody jsou označeny a v další iteraci již nejsou brány v potaz.

Fáze se opakují, dokud není nalezena žádná shoda dosahující minimální požadované délky. Dané fáze se řídí třemi pravidly, které autor (Prechelt, 200) uvádí:

- každý token se může shodovat pouze s jedním dalším tokenem, v první fázi je tedy nutné vylučovat sledy, které se protínají,
- podřetězce jsou vyhledávány bez ohledu na jejich pozici,
- jsou upřednostňovány delší shody, které autor uvádí jako více spolehlivé.

Prechelt (2002) provedl testování tohoto algoritmu v systému JPLag, který úspěšně odolal pokusům o plagiátorství v následujících případech:

- změna formátování kódu – tato změna nemá díky použití tokenů, žádný vliv,
- využívání komentářů – komentáře jsou z kódu vyloučeny,
- změna identifikátorů a překlad do jiného jazyku – samotné názvy nejsou v algoritmu vůbec hodnoceny,
- změna konstant.

4.2 Porovnání řetězců

Krom výše zmíněné metody je použito i základní porovnávaní textových dokumentů v systému Anton. Konkrétně je ke každému souboru zdrojového kódu vytvořen soubor obsahující pouze názvy identifikátorů funkcí a dalších nezabalených texto-

vých řetězců, které umožňuje vyfiltrovat výše zmíněná funkce tokenizace. Porovnání těchto souborů umožňuje podrobnější přehled o rozsahu podobností, protože konkretizuje část tokenů. Pokud vykazují tyto soubory větší podobnost, je pravděpodobné, že nalezená shoda mezi zdrojovými kódy není náhodná.

4.3 Data pro testování

Vytvořený systém je otestován z hlediska jeho funkčnosti, tedy toho, jak dokáže odhalit podobnosti ve zdrojových kódech. Pro tyto účely jsem si vybral a vytvořil 7 souborů, které vychází ze souborů reálně použitých v předmětu Aplikační programové vybavení k řešení semestrálního projektu. Tyto data byla vybrána z toho důvodu, že pochází ze skutečné školní činnosti a je v něm použit jazyk PHP bez frameworků, které by vzájemné rozpoznávání podobnosti v kódu narušily, jelikož by byla část kódu díky jejich použití stejná.

První soubor představuje PHP backend pro profil, druhý soubor představuje jeho modifikovanou verzi, která by měla napodobit opisování. V kódu budou zaměněny všechny identifikátory a přidány komentáře.

Třetí soubor bude vycházet ze souboru prvního, kdy bude první polovina kódu stejná, zbytek bude převzat z jiného souboru. Čtvrtý soubor je kód, který nemá nic společného s předchozími soubory. Pátý soubor je kombinací třetího a čtvrtého souboru, kdy první polovina je stejná jako v případě třetího a druhá polovina stejná jako ve čtvrtém. Šestý soubor je stejný jako první soubor, pouze jsou za sebe zaměněny poloviny.

Dále je vyzkoušen i soubor získaný od dalšího studenta, který byl vytvořen na základě stejného zadání. Na tomto souboru je zajímavé pozorovat, jak výuka a zadání ovlivňuje podobnost dvou souborů od dvou různých autorů.

Data prověřují systém z nejvíce nápadných stylů opisování, jako je přepisování názvů, zabalování funkcí, přehazování části kódu, nahrazování části kódu svým, vkládáním komentářů.

Všechna data byla vyzkoušena přímo na nasazeném modelu v systému Anton na lokálním serveru. Testovala se také funkčnost grafického rozhraní, zobrazování náhledu podobností.

Data o současném řešení na Mendelově univerzitě byla získána z volného rozhovoru s Ing. Jiřím Lýskem, Ph.D., který je garantem a vyučujícím v předmětech, kde se programovací jazyk PHP používá. K tomu proběhla i ukázka samotného používaného programu. Krom těchto informací mi poskytl také několik desítek anonymizovaných projektových souborů, na kterých jsem testoval funkčnost systému a porovnával se současně používaným systémem.

5 Implementace řešení pro odhalování plagiátů ve zdrojových kódech

5.1 Analýza současného řešení a požadavků

Z osobního setkání (Lýsek, 2017) vyplynuly následující body:

- V předmětu APV (aplikačně programové vybavení) se testuje PHP projekt v programu JPlag, konkrétně v jeho offline verzi, která je dostupná je stažení.
- Projekty předmětu Webové aplikace nejsou testovány žádným automatizovaným systémem. Projekty se testují tzv. od pohledu, jelikož je jich menší počet a projekty jsou řešeny více individuálně, kde se bere ohled i na strukturu kaskádových stylů.
- Nahrání projektu do offline verze JPlagu je okamžité, vyhodnocení trvá několik vteřin. Výsledky jsou zobrazeny odděleně v HTML souborech, které poskytují zobrazení procentuálních podobností a zároveň umožňují porovnání souborů s vyobrazenými podobnými částmi.
- V úvahu jsou brány pouze shody nad 50 %.
- JPlag napomohl najít úspěšně plagiátorství. V případě shody se texty ručně zkontrolují, a navíc se ještě porovnávají v textové podobě.
- Jako požadavek bylo uvedena znalost dalších formátů, které se v projektech používají, jako jsou například HTML, CSS, latte.

Z těchto bodů jsem do svého řešení zapracoval následující vlastnosti:

- JPlag nemá nativní podporu pro jazyk PHP a místo toho používá obyčejný textový parser. Mé řešení má nativní podporu pro PHP.
- S vložením zdrojového kódu se automaticky vytvoří a porovná i soubor obsahující textové sekvence, výsledek tohoto porovnání je zobrazen s výsledkem porovnání tokenů.
- Rozšíření systému bylo navrženo s ohledem na budoucí rozšíření a je možné přidávat tokenizery i pro další jazyky a formáty.

5.2 Instalace rozšíření

Nejdříve uvedu postup instalace řešení do zaběhnutého systému Anton. V příloze jsou dané soubory uvedeny v příslušných složkách, které přesně odpovídají hierarchické struktuře systému. Stačí je pouze vložit na stejné místo.

Krom vložení těchto souborů je nutno změnit i několik souborů systému:

- `/app/templates/@layout.latte` je nutné modifikovat o zobrazování příslušných částí v menu uživatele. Nový layout je v elektronické příloze.
- `/web/css/screen.css` je nutné přidat třídu, která zajišťuje zobrazení tabulek v porovnání zdrojových kódů:

```
1.  .source_code {
2.      text-align:left;
3.      max-width: 600px;
4.      max-height: 800px;
5.      overflow:scroll;
6.      line-height:90%;}
```

Kód 1 CSS třída zajišťující správné vyobrazení porovnávaných kódů

Samozřejmostí je i změna databáze, která je detailně popsána v následující kapitole. Je důležité nezapomenout příslušným souborům nastavit správná přístupová práva, tak aby k nim měl serverový deamon přístup a mohl spouštět skripty.

Decoder a comparer je vhodné nastavit v chronu nejlépe na interval několika minut, aby se výsledky dostavily co nejrychleji. V případě, že se spouští zároveň s funkcí textového porovnání, doporučuji, aby této funkci předcházely, jelikož z kódu vzniká zároveň i textový soubor, který tak bude následně na svůj vznik i porovnán.

Uživatel systému, který bude chtít rozšíření používat, bude muset obdržet nová uživatelská práva. Pro spravování předmětů je novou rolí **cor-adm**, který může v rámci své organizace zakládat nové předměty/projekty, editovat je a mazat. Role **source-code-adm** umožňuje uživateli vkládat a editovat nové zdrojové kódy a zároveň nahlížet na výsledky v předmětech své organizace.

5.2.1 Verze PHP

V závislosti na nainstalované verzi PHP je možné změnit od verze 7 a více `/app/backend/decoders/parsers.php` řádek 10 *token_get_all($source)* na *token_get_all($source, TOKEN_PARSE)*. Přidaný parametr TOKEN_PARSE způsobí to, že kód bude rozlišovat mezi řetězci a rezervovanými slovy. Například pokud by student pojmenoval objekt *public*, parser by to špatně rozpoznal a považoval by to za public, označení viditelnosti (The PHP Group, ©2017). Pokud je to možné, doporučuji tento parametr přidat, jelikož bude rozpoznávání podobností přesnější. Při změně verze PHP již běžícího systému je nutné všechny výsledky vymazat a nechat vytvořit nové tokeny, protože je možné, že se změnily identifikační čísla reprezentující tokeny.

5.3 Návrh

5.3.1 Postup porovnávání obecně

Rozšíření s rozpoznáváním zdrojových kódů respektuje stejný návrh, jako má původní textové rozpoznání. Nejdříve tedy uživatel nahraje skrze uživatelské rozhraní webové aplikace na server příslušný dokument s požadovanými informacemi. Tuto činnost má na starost presenter `DocSourceCodesPresenter.php` ve složce s presentery `app/presenters/`, datovou stránku obsluhuje model `app/models/DocSourceCodes.php`.

Další fáze souboru přebírají PHP soubory ve složce `app/backend`. Soubor `10-decoder-sc.php` vybere všechny soubory se zdrojovými kódy, které ještě nebyly převedeny na souhrn tokenů, a tuto operaci provede. Krom toho skrze tokeny ještě získá vlastnosti jednotlivých souborů a vytvoří textový dokument s názvy funkcí. Jednotlivé parsery jsou pak uloženy ve složce a souboru `decoders/parsers.php`, kde se ukládají jako funkce. Pro operace se soubory na uložišti serveru existuje modifikovaný soubor `DocumentFilesSC.php`, který obsahuje funkce pro vytváření souborů, mazání apod.

Dalším na řadě je soubor `20-comparer-sc.php`, který každý dokument porovná se všemi neporovnanými dokumenty ve svém předmětu a formátu. Zdrojová funkce je uložena v souboru `lib/tokenizer.php`.

Prezentaci výsledků pak opět umožňuje presenter s příslušnými komponenty a šablonami. Grafické zvýraznění podobností pak obsahuje soubor ve složce s backendem: `lib/CompareDocumentsSourceCode.php`.

Všechny výsledky jsou ukládány do databáze. Jednotlivé body postupu jsou podrobně rozepsány v dalších kapitolách. Upozorňuji, že řádkování v ukázkách kódů odkazuje na jejich pozici v souborech, kde se nachází, tak aby čtenář měl přehlednější a rychlejší přístup při jejich vyhledávání.

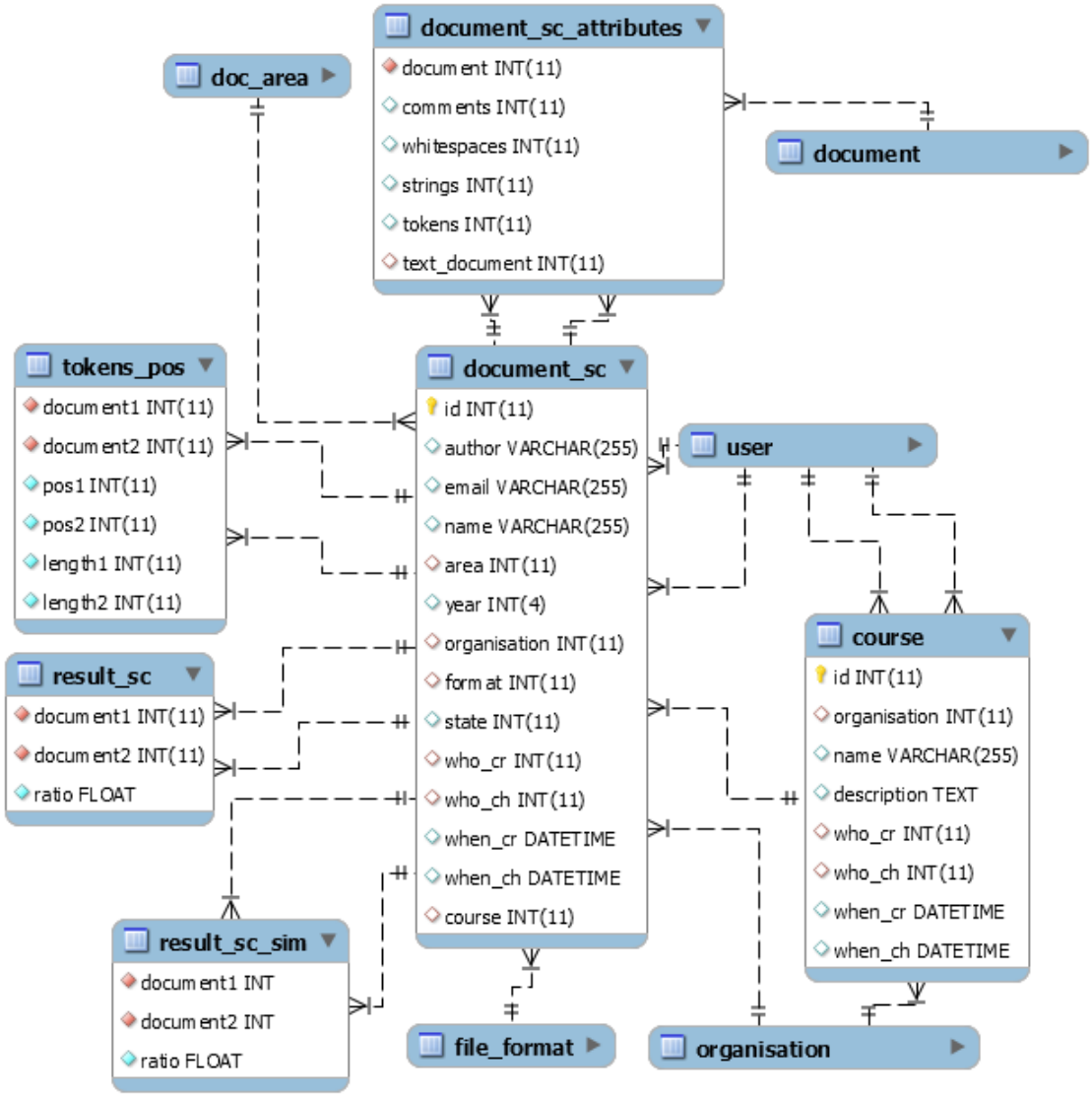
5.3.2 Databáze

Databáze systému běží na systému MySQL. V dalším textu se budu zabývat pouze tabulkami, které jsou přidány nově a u těch stávajících uvedu pouze případné modifikace. Schéma tabulek je uvedeno na obrázku dvě.

U dvou tabulek (document_sc a course) se nachází žurnálové údaje, které vychází z návrhu původní databáze. *Who_cr* a *who_ch* popisují, který uživatel (user) tabulku vytvořil, resp. změnil. Druhá dvojice *when_cr* a *when_ch* ukládá informace o vzniku, resp. změně.

Klíčovou tabulkou je tabulka **document_sc**, která ukládá potřebné údaje o dokumentech zdrojových kódů, tato tabulka vychází z tabulky document:

- *Id* je identifikační číslo, INTEGER s vlastností auto increment.
- *Author* je typu VARCHAR(255), ukládá jméno autora, ne toho, kdo soubor vložil.



Obrázek 2 Schéma databáze (zobrazeny jsou pouze nové tabulky, staré tabulky, které navazují, jsou nerozbalené)

- *Email* je typu VARCHAR(255) a obsahuje email na autora. Správný formát zajišťuje framework Nette při vkládání dokumentu.
- *Name* je typu VARCHAR(255) a reprezentuje název dokumentu.
- *Area* je cizí klíč na tabulku doc_area. Tato tabulka určuje, do které oblasti dokument spadá. Oblastí může být například Mendelova univerzita, nebo přímo konkrétní ústav.
- *Year* je INTEGER a označuje rok práce. V tomto případě je tento údaj užitečný pro porovnávání dvou prací, kdy je pravděpodobné, že pozdější projekt byl opsán z dřívějšího.

- *Organisation* je cizí klíč na tabulku organisation, jež představuje podnik, univerzitu, kde byl dokument vytvořen.
- *Format* představuje cizí klíč na tabulku file_format, která v tomto případě ukazuje programovací jazyk dokumentu, tedy PHP. Format může však obsahovat i další formáty, jako je třeba TXT, DOCx apod. Na hodnotu format je následně připraven tokenizer, tak je třeba, aby pro jazyk PHP byl nastaven decoder v tabulce format na „php-pa“, jinak systém nedokáže správně ztokenizovat daný dokument a neprovede tak porovnání. V systému jsou také porovnávány pouze zdrojové kódy stejného formátu.
- *State* je cizí klíč na tabulku antiplagiate_state, která informuje o stavu, ve kterém se dokument nachází. Pro textové dokumenty je využito celkem devět hodnot, pro zdrojové kódy nám stačí dvě:
  - 1 – dokument nebyl ještě ztokenizován.
  - 2 – dokument byl ztokenizován. V případě textového dokumentu je tato hodnota využita pouze přechodně a nikdy není uložena do databáze, proto zde může být použita pro zdrojové kódy
- *Course* je cizí klíč na tabulku course (viz dále).

Jako další rozeberu právě zmíněnou tabulku **course**, která reprezentuje údaje o příslušnosti dokumentu k projektu/předmětu. Postrádá totiž smysl hodnotit dva dokumenty, byť stejného formátu, které patří do dvou rozdílných projektů. Jednak by byly nalezené podobnosti pouze náhodné, a navíc by takové porovnání zbytečně zabralo výpočetní zdroje serveru. Atributy *id* a *organisation* mají stejné vlastnosti, jako u předchozí tabulky.

- *Name* je typu VARCHAR(255) a představuje název projektu/předmětu.
- *Description* je typu TEXT a umožňuje připsat případné detaily k projektu, například rok (v jednotlivých předmětech může docházet ke změnám projektů během času) a náplň práce.

Tabulka **document_sc_attributes** ukládá dodatečné informace k souboru. Ukládá převážně informace, které jsou při tokenizaci vyjmuty nebo přehlédnuty. Tyto údaje jsou získány automaticky.

- *Document* je cizí klíč ke zdrojovému souboru, kterého se informace týkají.
- *Comments* je typu INTEGER, který představuje, kolik komentářů se v daném souboru nachází. Komentáře jsou při porovnávaní ignorovány. Tento údaj také může posloužit při hodnocení prací, kde je třeba kód komentovat.

- *Strings* je typu INTEGER a představuje údaj o počtu nezabalených textových řetězců. Jedná se například o názvy proměnných. Pokud se tento atribut u dvou dokumentů shoduje, je velká pravděpodobnost, že budou podobné.
- *Whitespaces* je typu INTEGER ukládající počet bílých mezer v dokumentu. Pokud jsou dva dokumenty shodné, rozdílné číslo vykazuje pokus o změnu formátovaní.
- *Tokens* je INTEGER zobrazující počet tokenů dokumentu. Jedná se však jen o tokeny, které jsou brány v porovnávání, tedy nejsou v něm započítány bílé mezery, komentáře, komentáře nebo úvodní závorky v jazyce PHP.
- *Text_document* je cizí klíč na tabulku DOCUMENT, jedná se o důležitý údaj, který slouží ke správnému přiřazení výsledku textového porovnání.

Výsledky porovnání tokenů jsou pak uloženy v tabulce **result_sc**, kde *document1* a *document2* jsou cizí klíče odkazují na příslušnou dvojici dokumentů. *Ratio* je pak číselný údaj o shodě v intervalu 0-100, v systému je následně vyobrazen procentuálně.

Tabulkou se stejnou strukturou je **result_sc_sim**. Hlavním rozdílem je význam hodnoty ratio. Zatímco u předchozího případu se jednalo o podobu prvního dokumentu s druhým, tedy například první dokument mohl být 100% stejný jako druhý, zatímco druhý byl společný pouze v 1 %. Tento nepoměr způsobuje rozdílná délka dokumentů. Druhá tabulka toto bere v úvahu a započítává podobnost společnou pro oba dva dokumenty, kdy se jedná o dvojnásobek počtu společných tokenů vydělených celkovým součtem počtu tokenů obou dokumentů.

Konkrétní výsledky jsou ukládány do tabulky **tokens_pos**, nejedná se však o informace o společných sekvencích tokenů, ale o jejich řádkovém umístění v textu, které se používá pro grafické zobrazení shod**:**

- *document1* a *document2* jsou cizí klíče odkazují na příslušnou dvojici dokumentů. Je třeba podotknout, že jelikož algoritmus postupuje soubory od data vytvoření, je ID dříve nahraného dokumentu vždy na prvním místě. Při načítání podobností je třeba vždy na prvním místě zadat menší ID z dané dvojice.
- *Pos1* je typu INTEGER a udává řádkovou pozici dané shody v prvním dokumentu, resp. *Pos2* ukládá tu stejnou informaci k dokumentu dva.
- *Lenght1* a *lenght2* jsou typu INTEGER a zobrazují řádkovou délku k příslušným dokumentům včetně. Pokud je teda shoda jednořádková, atribut má hodnotu 1.

Pro chod systému je třeba ještě modifikovat jednu tabulku z původního návrhu, a to již zmíněnou tabulku **file_format**. Zde je pouze třeba doplnit atribut nový *source_code*, který může být libovolného typu, pouze v případě zdrojového kódu je v tomto atributu uložena nějaká hodnota a v případě textových formátů je atribut prázdný. Doporučuji pro ušetření pamětí vložit například bitovou hodnotu.

Na závěr bych chtěl upozornit na null hodnoty u většiny atributů. To vychází z původního návrhu databáze, kdy jsou takhle navrženy všechny hodnoty, nejspíše pro případ, že by systém některé z nich nezvládnul zapsat.

5.4 Backend

Ve složce `app/backend` jsou uloženy všechny soubory, které obstarávají hlavní činnost systému Anton. Nachází se tam jak porovnání textu, tak i zdrojového kódu, které zde popisuji.

5.4.1 Dekódování

Dekódování, tedy v tomto případě činnost převedení zdrojového kódu na řetězec tokenů, řídí soubor `10-decoder-sc.php`.

Na začátku se vytváří dva objekty, které spravují dokumenty zdrojových kódu a textových souboru. Textových, protože v této fázi dochází k vytvoření paralelního textového souboru pro další porovnání.

SQL dotazem se získávají všechny identifikační čísla zdrojových kódů, které mají jako state uveden jedna, tedy že ještě nebyly dekódovány. Pro manipulaci s databází se používá rozšíření dibi.

Pomocí funkce z objektu $repo je získán soubor se zdrojovým kódem. Umístění těchto souborů obstarává objekt ze souboru `DocumentFilesSC.php`, který téměř plně vychází z původního souboru `DocumentFiles.php`, jen je změna v konstantě umístění dat ke zdrojovým kódům. Jelikož Anton umisťuje soubory podle ID jejich dokumentů a zdrojové kódy mají vlastní tabulku, tedy i odlišné ID od textových, je nutno je ukládat na jiné místo, aby nedocházelo ke kolizím. Dále se testuje existence a obsah souboru, pokud jsou testy úspěšné, je obsah předán do funkce **getTokensPHP**(obsah). V případě nově přidaných formátů by bylo zde nutné vytvořit příslušné odkazy na funkce pro jednotlivé formáty, ale v této práci je řešen pouze formát PHP.

Tato funkce se nachází v souboru `decoders/parser`s a podrobně je popsána v další podkapitole Parsers.

Po získání výsledků tokenizace jsou tokeny zapsány do souboru v umístění zdrojového kódu s příponou tokens. Zároveň vznikají soubory obsahující řádkové lokace k tokenům a obsah všech nezabalených textových řetězců. Soubory mají příponu tokens_locations, resp. strings.

Vzniknou samozřejmě i příslušné záznamy k daným souborům. Pomocí funkce **dibi:getInsertID**() se nachází ID pro nově vytvořený textový dokument, vycházející ze souboru strings, a propojuje se skrze tabulku vlastností s dokumentem zdrojového kódu.

5.4.2 Parsers

Parsers je klíčovou součástí řešení, jelikož obstarává hlavní činnost rozložení souboru na tokeny. Bez něho by nebylo možné porovnávat kód z hlediska struktury.

Každý programovací jazyk má vlastní klíčové slova a specifická slova, takže je nutné mít pro každý z nich vlastní parser.

Programovací jazyk PHP má základní verzi parseru (v tomto případě tokenizeru) obsáhlou již ve všech svých distribucích. Volá se pomocí **token_get_all**(zdrojový_soubor) a ukládá výsledky do pole. Každý token obsahuje své identifikační číslo, řádkovou lokaci a obsah. Pomocí identifikačního čísla lze zjistit pomocí funkce token_name(číslo) zjistit název tokenu, ale název se neukládá. Je to zbytečné, protože čísla pro porovnání stačí, navíc zabírají méně místa a probíhají s nimi rychlejší výpočty. U jiných programovacích jazyků to však může být odlišné.

PHP parsers ve formě funkce **getTokensPHP**(kód) vrací pole obsahující následující hodnoty na daných pozicích:

- [0] – textový řetězec obsahující číselné sekvence tokenů, používají se pro uložení do souboru. Údaje jsou od sebe odděleny mezerou, stejně tak v případě dalších dvou řetězců.
- [1] – textový řetězec obsahující řádkové lokace k daným tokenům. Informace se párují podle pozice, tedy první token odpovídá prvnímu číslu v tomto souboru.
- [2] – textový řetězec obsahující nezabalené řetězce.
- [3] – číselný počet bílých míst.
- [4] – číselný počet komentářů.
- [5] – číselný počet řetězců.
- [6] – číselný počet porovnávaných tokenů.

V případě přidávání dalších formátů a jejich parserů je nutno dodržet výstupní pořadí a formát.

```
<?php
include("zacatek.php");
$tplVars["oszce"] = $stmt->fetchAll();
$stmt = $db->query("Schuzky");
$tplVars["scky"]=$stmt->fetchAll();
if(!empty($_POST["eee"])){$stmt->execute();}
$i=10;
```

```
262 323 320 323 320 366 319 320
320 366 319 323 320 323 320 366
319 327 359 320 323 320 366 319
320 317
```

Obrázek 3 Výsledek tokenizace v systému. Vlevo originální kód, vpravo sekvence tokenů

<?php **include("start.php");** **$prommene["jedna"] = $statement->naplnVsechny();** **$statement = $database->dotaz("setkani");** **$prommene["dva"]=$statement->naplnVsechny();** **if(!empty($_POST["eet"])){$statement->odesli();}** **$pocitadlo=15;**	**262 323 320 323 320 366 319 320 320 366 319 323 320 323 320 366 319 327 359 320 323 320 366 319 320 317**

Obrázek 4 Výsledek tokenizace kódu, který má stejnou strukturu, jako kód v obrázku 3

5.4.3 Porovnávání

Po první fázi dekódování nastává fáze porovnávání v souboru `20-comparer-sc.php`. Nejdříve soubor načte všechny dokumenty zdrojových kódů a ke každému souboru najde takové dokumenty, které spolu nemají žádný výsledek, sdílí formát a předmět. Navíc musí být dokument již dekódovaný, tzn. *state* musí být roven dvěma.

```
30. dibi::query( 'SELECT [id] AS [document_id] FROM document_sc
31.     WHERE id NOT IN (SELECT document1 FROM result_sc WHERE document2 = %i)
32.           AND id NOT IN (SELECT document2 FROM result_sc WHERE document1 = %i
33.           AND id <> %i
34.           AND format = %i
35.           AND state = 2
36.           AND course = %i
37.      ORDER BY [document_sc].[when_cr]' , $row['document_id'], $row[
    'document_id'], $row['document_id'], $row['format_id'], $row['course_id'];
```

Kód 2 SQL dotaz pro získání neporovnaných dokumentů s dokumentem, který právě porovnáváme

Ke každému souboru získáme jeho obsah a ten vložíme do funkce **stringTiling**, která vrátí pole výsledků. Této funkci je věnována další kapitola.

Dále je spočítán počet tokenů pro každý soubor a do pole jsou uloženy jejich řádkové lokace.

Výsledky funkce **stringTiling** jsou následně procházeny jedna po druhé a ukládány do databáze. Zde však musí dojít ke konverzi pozice tokenu v poli tokenů na jeho řádkovou lokaci, a zvláště musí být přepočítána délka shody na délku počtu řádků.

Pokud je například výsledek, že se dokumenty shodují na pozicích 34 a 84 a délka shody je 6 tokenů, je nutné najít pozici pro oba dva tokeny a také spočítat v obou případech pozici tokenu na konci shody. Následně se odečte kratší lokace od delší a přičte se jednička (pokud by byly na stejném řádku, byla by délka 0 a to není žádoucí.)

```
$length1 = $locA[$tile[0] + $tile[2] - 1] - $locA[$tile[0]] + 1;
```

Kód 3 Ukázka výpočtu řádkové délky. $locA je pole řádkových pozice, $tile[0] je pozice prvního tokenu ve shodě a $tile[2] je délka tokenové shody

Tyto údaje se pak uloží do tabulky **tokens_pos**. Následně se podělí počet shodných tokenů počtem celkových a výsledek se uloží do databáze v tabulce **result_sc**. Společná podobnost obou dokumentů se uloží do tabulky **result_sc_sim**. Cyklus se opakuje tak dlouho, dokud existují dvě neporovnané dvojice.

5.4.4 Porovnání tokenů

Nejdůležitějším bodem této práce je porovnávání tokenů. K tomu slouží funkce **stringTiling**(tokeny 1, tokeny 2) v souboru `lib/tokenizer.php`. Tato funkce jako vstup přijímá pole tokenů obou dvou souborů a výstupem je pole obsahující pole s těmito informacemi: pozice shody v prvních tokenech, pozice shody v druhých tokenech a délka shody.

Na začátku funkce jsou tokeny tzv. označkovány, to znamená, že každý token v poli je doplněn číselným údajem 0. To znamená, že daný token je neoznačen a může se testovat na shody. Označený token s číslem jedna už figuruje v nějaké shodě a je nesmyslné ho dále zkoumat. K tomuto označkování se používá funkce **prepareMarks**(tokeny).

Následuje porovnání velikosti obou polí tokenů. Algoritmus porovnává postupně všechny tokeny prvního pole se všemi z druhého pole. Tento postup se optimalizuje tím, že jako první pole se vezme to menší, což urychluje celý postup. Pokud si vezmeme nejhorší možný případ, kdy by pole neměly ani jednu shodu a první pole mělo délku tisíc a druhé jen deset, algoritmus by porovnával všech tisíc tokenů. Prohozením by kontroloval jen deset. Při prohození je nutné zaznamenat si prohození, aby na konci algoritmu došlo k opětovnému prohození, protože je očekáváno, že výsledky jsou ve stejném pořadí jako vstupy.

Po této části se nachází již hlavní tělo algoritmu. Hned na začátku je nastavena hodnota proměnné **$minimalMatchLength**. Tento bod je klíčový, protože určuje minimální délku shod. Platí pravidlo, že čím větší číslo je, tím budou nalezené shody pravděpodobně méně náhodné. Číslo by mělo být dostatečně velké, aby eliminovalo právě onu náhodnost, ale zároveň by nemělo být příliš velké, aby zvládlo zaznamenat i kratší shody. Hodnoty v rozmezí 1-3 jsou pak naprosto nemyslitelné, protože by se pravděpodobně kratší dokument s tím větším shodoval vždy na 100 procent. Tato minimální hodnota se zároveň zapíše do proměnné **$maxMatch**. Proměnná **$matches** je pak pole sdružující nalezené shody.

For cyklem se pak prochází všechny tokeny z prvního pole, které jsou označeny nulou. Tento token se pak porovnává postupně s každým tokenem z druhého pole. Pokud jsou tokeny stejné a oba dva jsou neoznačené, zvýší se pomocná proměnná **$j** o jednu hodnotu a testuje se stejnými pravidly další dvojice tokenů. Pokračuje se tak dlouho, dokud existují shodné dvojice.

Až tohle porovnání skončí, testuje se délka shody. Pokud je stejná jako hodnota **$maxMatch** a zároveň je **$matches** prázdné, uloží se nalezená shoda do **$matches**. Pokud pole **$matches** prázdné není, testuje se překrývání shod. Shody se totiž nesmí překrývat, jelikož by se pak stalo, že by se jeden token shodoval s více tokeny, a to by porušovalo pravidla algoritmu.

```
42. $j = 0;
43. while ( ($tokensA[$i + $j][0]  == $tokensB[$b + $j][0] )  && ( $tokensA[$i
    + $j] [1] == 0) && ( $tokensB[$b + $j][1]  == 0)  && ($i + $j <
    count($tokensA) - 1) && ($b + $j < count($tokensB) - 1)) {
44.   $j++;
45. }
46. if(($tokensA[$i+$j][0]==$tokensB[$b+$j][0])&&($tokensA[$i+$j][1]==0)&&(
    $tokensB[$b+$j][1]==0)){$j++;}
47. if ($j == $maxMatch) {
48.   $addMatch = true;
49.   if (count($matches) != 0 ) {
50.     for ($k = 0; $k < count($matches); $k++) {
51.         if ((($i >= $matches[$k][0]) && ($i <= $matches[$k][0] + $matches[$k][
    2] - 1 ))  or  ( ( $b >=  $matches[$k][1] )  && ( $b <=  $matches[$k][1] +
    $matches[$k][2] -  1) )) {
52.             $addMatch = false;
53.         }
54.     }
55. }
56. if ($addMatch == true) {
57.   array_push($matches, [$i, $b, $j]);
58. }
59. } else if ($j > $maxMatch) {
60.   $matches = [[$i, $b, $j]];
61.   $maxMatch = $j;
62. }
```

Kód 4 Vnitřní kód cyklu porovnání řetězců tokenů: porovnání tokenů a ukládání výsledků

Pokud je však délka shody větší než **$maxMatch**, vloží se do pole **$matches** nová trojice hodnot (pozice + délka) a **$maxMatch** se přepíše novou hodnotou.

Po této fázi pak dojde k označení všech shodných tokenů a výsledky jsou vloženy do pole **$tiles**, které se vrací jako hodnota celé funkce. Celý postup se opakuje, dokud jsou nejdelší shody delší požadovaná minimální délka. Ještě před vrácením dojde v případě otočení polí tokenů k jejich vrácení do původního stavu.

```
68. foreach ($matches as $match) {
69.     for ($j = 0; $j < ($match[2] - 1 ); $j++) {
70.         $tokensA[$match[0] + $j][1] = 1;
71.         $tokensB[$match[1] + $j][1] = 1;
72.     }
73.     array_push($tiles, [$match[0], $match[1], $match[2]]);
74. }
```

Kód 5 Označení tokenů, které se už vyskytly ve shodě

5.4.5 Grafické porovnávání

Grafické porovnání je nedílnou součástí řešení a pro uživatele přehledně znázorňuje místa, které se v kódu shodují. Je však zapotřebí se pak na takové shody podívat a vlastním zrakem porovnat, zda je podoba čistě náhodná, nebo se jedná o stejný či zmodifikovaný kód. Zároveň je třeba brát v úvahu, že algoritmus k sobě spojuje nejdelší shody. Může se stát, že algoritmus graficky nedokáže zobrazit stejné části kódu, protože se v kódu objeví jiná náhodná a delší/dřívější shoda, než jsou stejné konkrétní části, ale v procentuálním výsledku grafické zobrazení nehraje roli a čísla by měla být správná.

Zobrazení shod zajišťuje metoda **getFormatedSCDocument**($id, $matches, $order) ve složce `lib/CompareSourceCodeDocuments.php`. Tato metoda má tři vstupy:

- **$id** je číslo porovnávaného dokumentu, slouží pro najití správného souboru.
- **$matches** (shody) jsou výsledky z databáze (*tokens_pos*) pro daný dokument. Jedná se o pole dvojice ve formátu [pozice, řádková délka].
- **$order** (pořadí) značí, zda je dokument první, nebo druhý. Vstupem je písmeno „a“ nebo písmeno „b“. Tento vstup je důležitý pro metodu **insertAnchor**($count, $order), která v zobrazovaných dokumentech vytváří křížové odkazy. Když je $order (pořadí) nastavené na „a“ vytváří odkaz v podobě „b$count“ a naopak. Uživatel pak může shody okamžitě lokalizovat v druhém dokumentu pomocí kliknuti na šipku zobrazenou u počátku shodné sekvence.

Výstupem této funkce je pak textový řetězec obsahující HTML tagy. Tento řetězec lze vypsat na webové stránce a je správně naformátovaný pro zobrazení v prohlížeči.

Samotná funkce probíhá tak, že testuje řádky s výsledky shod. Pokud se shoduje pozice s číslem řádku, obarví se řádek žlutým pozadím a před číslo řádku se vloží kotva a odkaz na protější kotvu. Text kódu je třeba zabalit do funkce **htmlspecialchars**, která odskakuje speciální znaky, tak aby se zabránilo špatné interpretaci kódu a jeho spuštění. Zároveň se do proměnné **$length** uvede délka shody a každým dalším řádkem se od proměnné jedna hodnota odečte. Dokud je **$length** nenulová, barví se pozadí na žluto. Tímto postupem se postupně vypíše celý zdrojový kód.

```
32. $i = 1;
33. $length = 0;
34. while (($line = fgets($content)) !== false) {
35.     if(false !== $key = array_search($i, $positions))
36.          {$result .= insertAnchor($key,$order) . '<span style="background-
    color: yellow;">'.'Line ' .$i. ': '. htmlspecialchars($line) .'</span></br>‘;

37.          $length = $lengths[$key] - 1;
38.         }
39.     else {
40.             if($length>0){
41.                $length -=1;
42.                $result .= '<span style="background-color: yellow; " > ' . 'Li-
    ne ' . $i . ': '. htmlspecialchars($line) .'</span></br>';
43.             }   else {
44.       $result . =   'Line ' . $i . ': '. htmlspecialchars($line) . '<br>';
45.             }
46.         }
47.     $i++;
48. }
```

Kód 6 Cyklus, který vytváří HTML řetězec s vyznačením porovnání shod

5.5 Nette soubory

Aby backend vůbec fungoval, je nutné vytvořit vrstvu, přes kterou bude uživatel nahrávat a spravovat dokumenty. To má na starost framework Nette a jeho součásti tvořící webovou prezentaci systému Anton. V dalším textu budu popisovat pouze nejdůležitější funkce a body v souborech, jelikož část těchto souborů obsahuje funkce obsluhující vyobrazování správných komponentů na stránce, nebo pouze čte data a předává data z databáze. Takové funkce není třeba detailně popisovat, jejich obsah se často opakuje nebo využívá již připravených funkcí.

5.5.1 Presentery

Všechny presentery se nachází ve složce `app/presenters/`. Přidanou část starající se o **course** obsluhuje `CoursePresenter.php`. Jelikož v této části není nutné zabývat se s žádnými složitými operacemi, nachází se v presenteru jen funkce vytvářející komponenty pro šablony.

Dalším presenterem přidaným v tomto rozšíření obsluhující hlavní činnost, a to správu zdrojových kódů je `DocSourceCodesPresenter.php`. Tento presenter částečně vychází z `DocumentsPresenter.php` obsaženého v Antonu, protože obě dvě části mají podobný obsah, jen se zabývají jinými typy dokumentů a z tohoto faktu vychází všechny provedené modifikace.

Hlavní funkcí je zde **actionCompare**(ID dokumentu 1, ID dokumentu 2), která se stará o to, aby šablona získala správné data a mohla vyobrazit grafické podobnosti mezi dvěma dokumenty. Na začátku funkce je nutné porovnat identifikační čísla dokumentů, protože v tabulce **tokens_pos** budou výsledky vždy uloženy v pořadí od nejmenšího po největší ID dokumentu, takže je nutné položit dotaz ve správném pořadí. Výsledky se pak ukládají do proměnné **$matches1**, resp. **$mat-**

ches2. K dokumentům se získají i další údaje z tabulky **document_sc_attributes** a uloží se do šablony v proměnné **features1**, **features2**. Následuje načtení tabulek každého dokumentu a předání jejich ID společně s **$matches1 ($matches2)** do backendu v podobě funkce **getFormatedSCDocument**. Jak bylo uvedeno v předešlých kapitolách, této funkci se předají i písmenka „a“ a druhému zavolání „b“, aby se vytvořily správné křížové odkazy. Zavolá se i funkce porovnávající textové dokumenty přiřazené ke zdrojovým. Postupně se ještě načtou do proměnných v šabloně procentuální výsledky.

Další funkce v tomto presenteru se zabývají vytvářením komponent pro HTML šablonu.

5.5.2 Modely

Modely spojují presenter s databází a obsluhují ji. Nachází se ve složce `app/models`. Model `Course.php` krom volání zděděných metod volá jednu metodu svou a to je funkce **delete**(ID dokumentu), která z databáze vymaže příslušný řádek tabulky s daným identifikačním číslem.

Model `DocSourceCodes.php` je opět částečně stejný s modelem `Documents.php`. U funkce **getList** je rozdíl v přijímaném vstupu, zatímco u textového dokumentu lze vložit ID uživatele, zde je jako nepovinný parametr ID předmětu (course). Model tedy neumí zobrazit soubory jednoho uživatele, protože je pro uživatele užitečnější vidět soubory všech uživatelů v rámci jednoho projektu nebo předmětu. Více uživatelů tak může na jednom předmětu spolupracovat a jejich data se okamžitě sdílí. Funkce **dataCheck** byla změněna o počet potřebných údajů v poli, tak aby odpovídaly rozdílné struktuře tabulky **document_sc**.

Funkce **getForeignKeysSelec**t a **getForeignKeysJoin** byly pak obměněny o spojení s dalšími tabulky a daty, které s dokumenty zdrojových souborů souvisí, jako je například **document_sc_attributes**.

U funkce **getSuspiciousDocuments** přibyl nepovinný parametr ID course, což umožňuje nacházet podobnosti dokumentů jen napříč daným předmětem.

Novou funkcí je **getFileFormats()**, která vytahuje z databáze data všech souborových formátů, kde je zaplněn atribut source_code. Tato funkce se pak využívá při zobrazování formuláře vložení nového zdrojového kódu. Ke stejnému účelu slouží taky funkce **getCourses**(ID organizace), která vrací všechny předměty příslušné uživatelovi organizaci.

Pro zobrazení všech informací v seznamu předmětů je využita funkce **getCoursesList**(ID organizace), která pro každou organizaci vrátí její předměty s informacemi o počtu souborů a autorovi předmětu.

GetCoursesCount(ID organizace) a **getCoursesDocsCount**(ID předmětu) jsou pomocné funkce pro získání počtu předmětů organizace, resp. počtu dokumentů v jednotlivém předmětu.

Funkce **getMatches**(ID dokumentu 1, ID dokumentu 2) vrací všechny výsledky shod z tabulky **tokens_pos** mezi dvěma zadanými dokumenty. Metoda **getFeatures**(ID dokumentu) vrací informace z dodatečné tabulky **document_sc_attributes** pro konkrétní zdrojový kód. **GetTextDocumentID**(ID doku-

mentu) pak vrací přímo ID textového dokumentu s texty řetězců ze zdrojového kódu. Tyto metody se používají při grafickém zobrazení shod. Poslední dvě metody **getTextDocumentResults** a **getResults** vrací procentuální výsledky shod.

5.5.3 Komponenty a šablony

Komponenty a šablony jsou soubory, které vytváří HTML strukturu pro zobrazení uživateli v internetovém prohlížeči. Systém Anton je navržen tak, že všechny formuláře a seznamy jsou uloženy v komponentech. Pro zobrazení těchto částí pak šablona volá funkce z presenteru, které je vytvoří. Šablony se nachází ve složce `app/templates` a komponenty jsou uloženy ve složce `app/components`. Šablony jsou napsány všechny ve formátu latte. Komponenty jsou napsány buď v latte (seznamy) nebo v PHP (formuláře). Některé šablony obsahují paginator, jedná se o již obsažený soubor v systému Anton, který umožňuje stránkovat a procházet výsledky v seznamech.

Šablony zobrazující část course se nachází ve složce `course`. Nachází se zde tři stránky: **editace** (edit), **seznam** (default) a **přidání** nové šablony (add).

Stránka **editace** volá dva komponenty: **courseChangeForm** a **coursesList**. Této stránce je nutné předat ID editovaného řádku v tabulce *course*. Komponent courseChangeForm obsahuje třídu, která vytváří nový formulář s následujícími prvky (názvy formulářových prvků musí být stejné jako jejich obdoba v databázi, aby se data správně nahrála):

- Textové pole pro jméno (name).
- Textové pole pro popis (description).
- Skryté pole pro ID, kam se vkládá hodnota právě editované tabulky (id).

Do textových polí se nastavují výchozí hodnoty přezvané z editované tabulky.

Komponent **coursesList** se stará o zobrazení seznamu všech předmětů se všemi potřebnými parametry, tento komponent je také obsažen i v dalších dvou šablonách. Komponent vytváří HTML tabulku, která v hlavičkovém řádku obsahuje nadpisy pro následující údaje, které pak pomocí cyklu vypíše o všech předmětech:

- ID předmětu.
- Jméno předmětu.
- Popis předmětu.
- Datum a čas poslední změny.
- Uživatele, kdo provedl poslední změnu.
- Akce, které obsahují další soubor latte CommonListActions. Tento soubor vytváří základní tlačítka pro každý řádek tabulky, jako je odkaz na smazání nebo editaci.

Stránka **seznamu** volá výše zmíněný coursesList. Stránka přidání pak vytváří komponent **courseCreateForm**, který je prakticky totožný s editačním formulářem, jen chybí skryté pole pro ID a nejsou nastaveny žádné výchozí hodnoty.

Šablony pro hlavní část porovnávání zdrojových kódů se nachází ve složce `DocSourceCodes` a je jich zde poměrně větší množství než v předešlé části:

- Add – přidání nového dokumentu.
- Edit – editace základních informací o dokumentu.
- Compare – grafické porovnání dvou srovnávaných dokumentů.
- Courses – seznam všech předmětů organizace, do které spadá uživatel.
- Default – seznam všech zdrojových kódů v systému.
- Detail – tabulka s informacemi o konkrétním dokumentu.
- detailCourse – tabulka se všemi dokumenty daného předmětu.
- suspCourse – seznam všech maximálních výsledků srovnání v rámci jednoho předmětu.
- suspDetail – tabulka s výsledky všech srovnávacích testů daného dokumentu.

Šablona **add** se skládá z komponentu **docSourceCodeCreateForm**, vytvářející formulář pro nahrání nového zdrojového kódu. Formulář ještě před vykreslením načítá data z databáze. Formáty, které spadají do kategorie zdrojových kódů, a předměty, které patří do stejné organizace, co uživatel. Krom toho čte z pravidel systému maximální povolenou velikost souborů. Formulář pak obsahuje tyto položky, které jsou všechny povinné:

- Textové pole pro jméno autora dokumentu, ne toho, kdo soubor nahrává do systému (author).
- Textové pole pro email autora dokumentu (email). Probíhá kontrola formátu.
- Textové pole pro název dokumentu (name).
- Textové pole pro rok vzniku dokumentu (year). Může obsahovat pouze číslo.
- Výběrové pole pro formát (format).
- Výběrové pole pro předmět (course).
- Výběrové pole pro oblast (area).
- Nahrání souboru (upload).
- Skryté pole pro stav dokumentu (state). Nastavuje se 1, což značí nově nahraný a nedekódovaný dokument.
- Skryté pole pro ID organizace (organisation).

Šablona **edit** obsahuje komponentu **SourceCodeChangeForm**, která vytváří formulář podobný tomu při nahrávání nového souboru jen s těmi rozdíly, že nelze měnit formát a předmět dokumentu. Také uživatel nemůže změnit už nahraný dokument, protože je možné, že dokument už byl dekódován. Samozřejmostí je načtení výchozích hodnot právě editovaného dokumentu.

Compare nevytváří žádný seznam, ani formulář, proto neobsahuje komponent. Místo toho zobrazuje tři základní celky. Prvním je porovnání zdrojových kódů dvou dokumentů. V tabulce jsou uvedeny vzájemné podobnosti a pod tím jsou vedle sebe zdrojové kódy, jejichž výsledek je získán z backendové funkce grafického porovnání. Důležité je zde zabalit tyto výsledky do HTML tagu `<pre>`, který způsobí, že jsou výsledky naformátovány přesně i s mezerami. Uživatel tak může lépe porovnat podezřelý kód. V další části je tabulka obsahující doplňující údaje o dokumentech z databázové tabulky *document_sc_attributes* obou dvou dokumentů vedle sebe. Na závěr stránky se nachází procentuální údaje o vzájemné podobnosti textových dokumentů a opětovné grafické znázornění jejich podobnosti používající původní systémové řešení.

Šablona **courses** používá komponent **DocSourceCodesCoursesList**, vytvářející seznam s informacemi o všech dokumentech:

- ID předmětu.
- Jméno uživatele, co předmět vytvořil.
- Datum a čas vytvoření.
- Název předmětu.
- Popis předmětu.
- Počet souborů.
- Odkaz na výsledky porovnání souborů.
- Odkaz na seznam všech souborů v předmětu.

V šabloně **default** se volá komponent **DocSourceCodesDocumentList** poskytují informace o všech dokumentech zdrojových kódů v systému:

- ID dokumentu.
- Jméno autora.
- Název dokumentu.
- Název předmětu.
- Název formátu.
- Rok dokumentu.
- Název organizace.
- Status zpracování.
- Odkaz na výsledky porovnání.
- Datum a čas poslední změny.

- Uživatel, co provedl změnu.
- Seznam akcí: smazání dokumentu, editace, zobrazení detailu.

Šablona detail zobrazuje tabulku vlastností daného dokumentu. Tato tabulka je v komponentě **SourceCodeDocumentDetail**. Krom výše zmíněných vlastností vypisuje i další informace:

- Počet tokenů.
- Počet komentářů.
- Počet textových řetězců.
- Počet bílých míst.
- ID textového dokumentu přiřazeného ke zdrojovému kódu, včetně odkazu na detail a na výsledky porovnání.
- Odkaz na stažení originálního dokumentu.

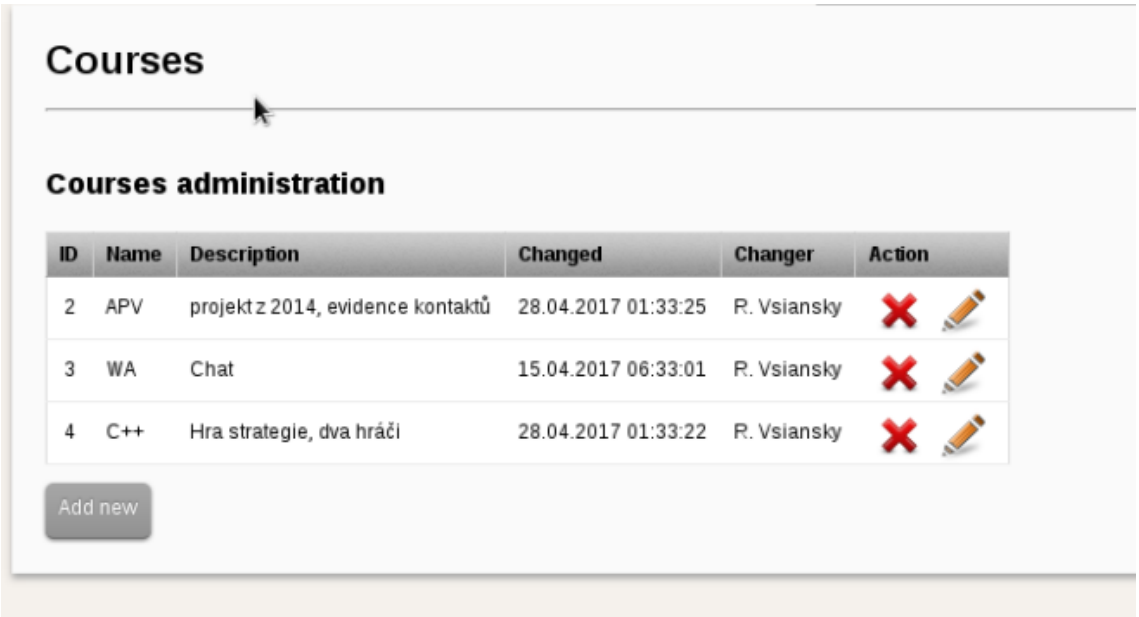
Šablona **detailCourse** pomocí komponentu **SourceCodeCourseList** vykresluje stejná data jako seznam všech zdrojových kódů (default). Jenom s tím rozdílem, že vypisuje data daného předmětu.

SuspCourse obsahuje v sobě komponent **CourseSuspiciousList**, která vypisuje data pro maximální výsledky společných podobností dokumentů daného předmětu. Co řádek, to jeden dokument s procentuálním výsledkem největší shody. Krom základních údajů, jako je název dokumentu, rok apod. se v tabulce nachází ještě dané procento a odkaz na **suspDetail** daného souboru. Ten využívá komponent **SourceCodeDocumentSuspDetail**, který se dělí na dvě části. První tabulka je shodná s tabulkou ze šablony detail, pod ní je pak vykreslen seznam všech shod, kde jsou stejné základní údaje jako u suspCourse, ratio vztahující se k dané dvojici dokumentů a odkaz na šablonu compare s vyobrazením obou dokumentů.

5.6 Grafické rozhraní

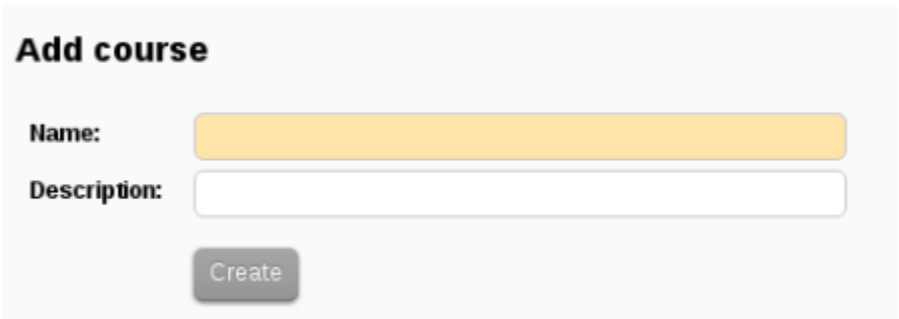
V dalším textu se budu zabývat klíčovými stránkami pro uživatele, které obsluhují hlavní funkce rozšíření. Některé stránky jsou si totiž velice podobné nebo dokonce stejné, například editace a přidání dokumentů jsou prakticky totožné, liší se pouze ve vyplnění výchozích hodnot, bylo by proto zbytečné je zde uvádět dvakrát.

5.6.1 Administrace předmětů



Obrázek 5 Základní seznam administrace předmětů

Na obrázku 5 je znázorněn seznam administrace předmětů. Uživatel zde vidí ID, název, popis a informace o poslední změně – kdo a kdy ji provedl. Pomocí tlačítka v podobě červeného křížku uživatel předmět smaže, tlačítkem s podobou tužky edituje. Dole se nachází tlačítko pro vytvoření nového předmětu. V případě přidání nového předmětu stačí zadat pouze údaj o jeho názvu a volitelným atributem je popis.

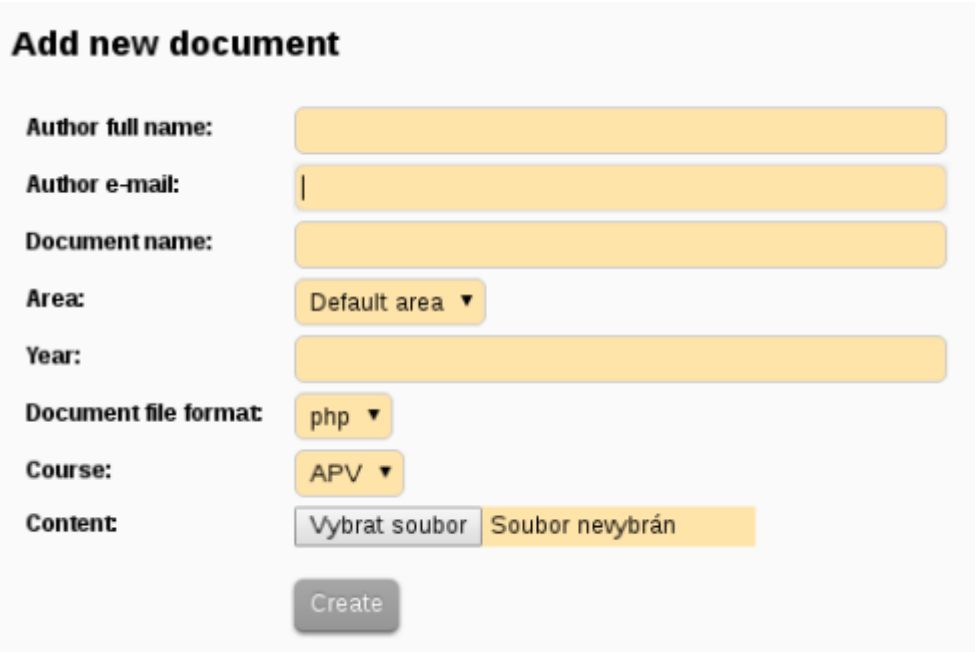


Obrázek 6 Přidání nového předmětu

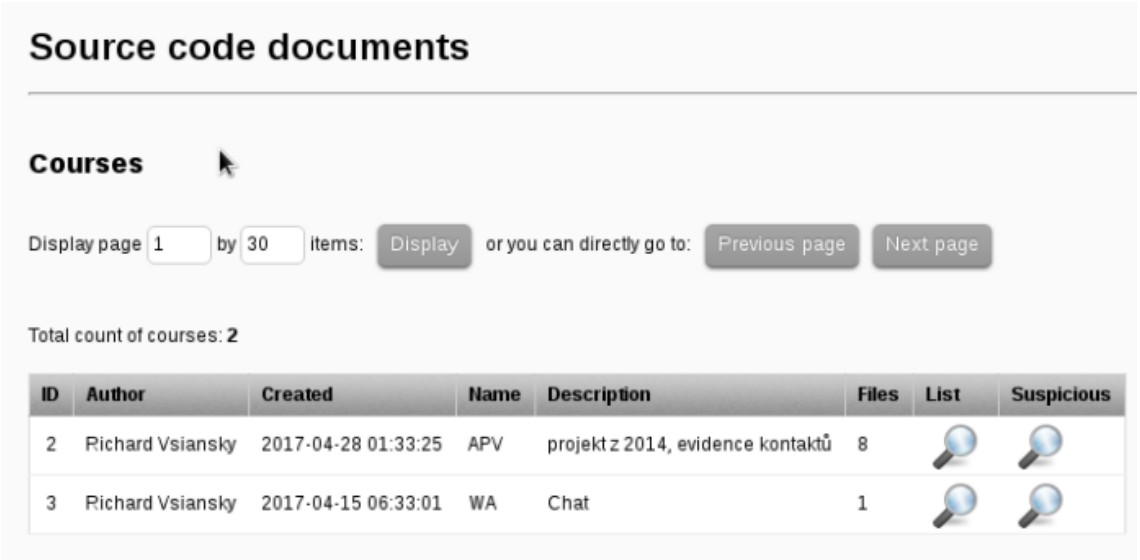
5.6.2 Přidávání dokumentů

Přidávání dokumentů zdrojového kódu vychází z formuláře pro přidání nového dokumentu. Uživatel zde uvede všechny potřebné údaje a soubor nahraje do políčka content pomocí standardního souborového prohlížeče svého operačního systé-

mu. Systém hlídá formát dokumentu, musí se shodovat s formátem vybraným v poli format. Všechny údaje jsou povinné, formulář se odešle tlačítkem create. Stránka se dále nepřesměrovává, uživatel tudíž může změnit jen patřičné údaje a nahrát další dokument.



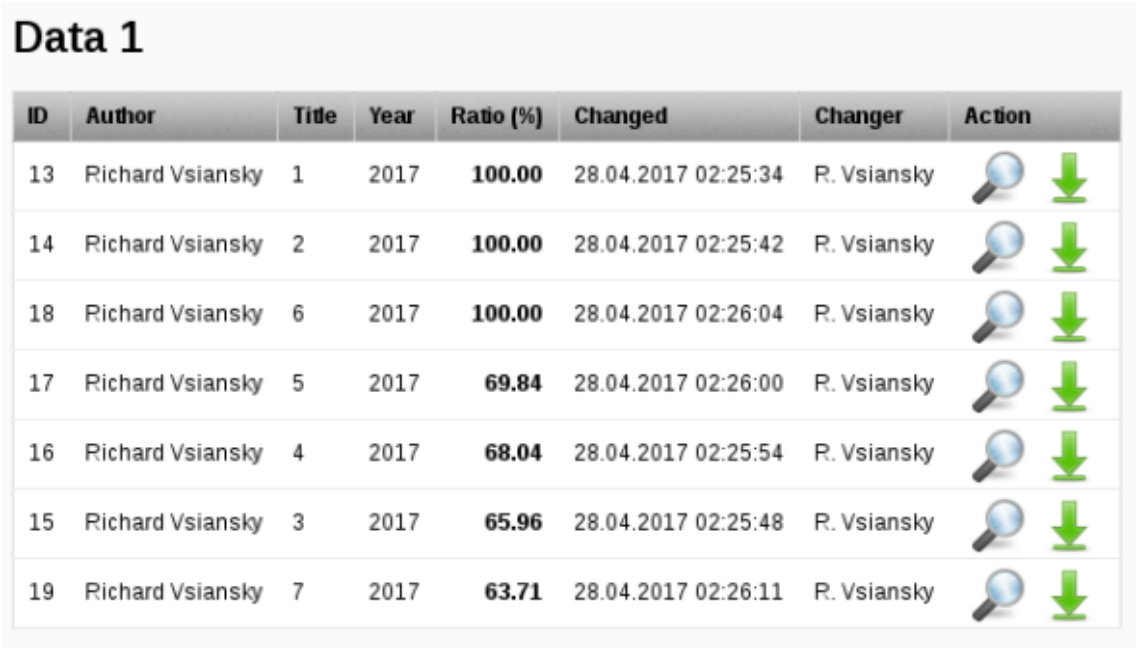
Obrázek 7 Přidání nového dokumentu se zdrojovým kódem



Obrázek 8 Administrace zdrojových kódů dle předmětů

5.6.3 Zdrojové kódy podle předmětů

Administrace zdrojových kódů podle jejich přiřazení k předmětům je klíčovou stránkou pro uživatele. Krom informací o předmětu je zde údaj o počtu souborů a odkazy na další dvě podstránky: seznam všech dokumentů v předmětu a seznam maximálních výsledků porovnání. Uživatel kontrolující výsledky bude hlavně kontrolovat výsledky porovnání.



Obrázek 9 Výsledky porovnání v předmětu „Data 1“

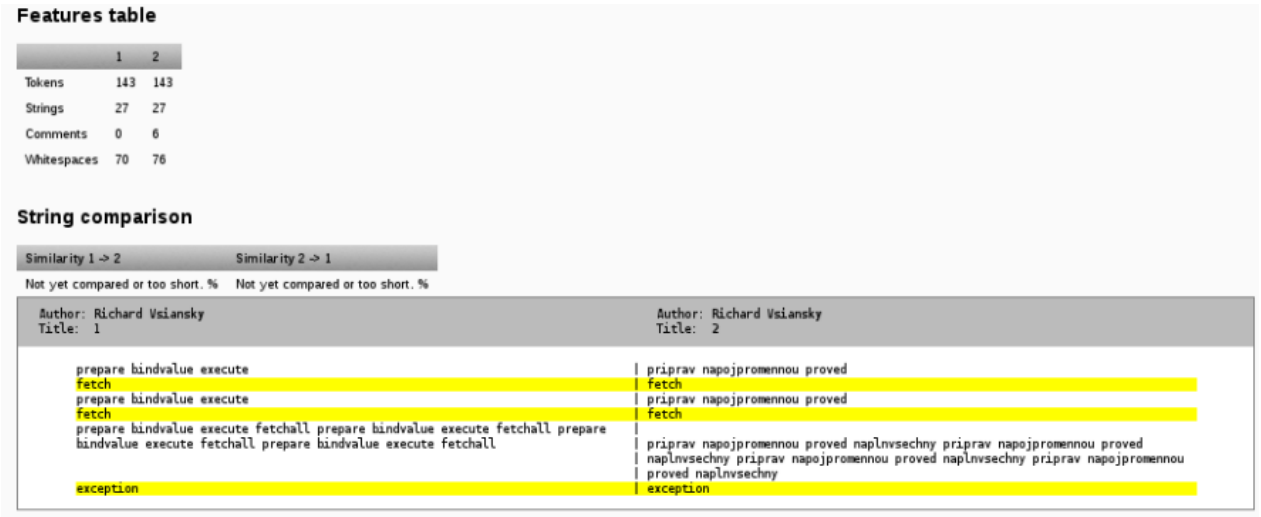
V těchto výsledcích má každý dokument jeden svůj řádek, kde uživatel vidí autora, rok a procentuální vyjádření nejvýše dosaženého výsledku společné shody (z tabulky result_sc_sim). Například řádek dokumentu s ID 13 na obrázku 9 ukazuje, že existuje k dokumentu 100% shodný dokument. Po rozkliknutí na tlačítko lupy se dostane uživatel na seznam všech výsledků daného dokumentu, kde lze vyčíst i další informace o výsledcích a pokračovat na detailní porovnání. Vzhledem k velikosti je screenshot dané stránky zobrazen v příloze A. Krom informací o daném dokumentu v horní tabulce je pod ní zobrazen seznam všech výsledků shod, tlačítko lupy odkazuje na stránku s detailním zobrazením podobností mezi dvěma dokumenty.

V první části stránky je grafické porovnání podobností kód (viz obrázek 10). Kód je na stránce zobrazen ve stejném formátování, jako v originálním souboru. Podobné řádky jsou obarveny žlutým pozadím. Jelikož tokenizer PHP ukládá pouze řádkové pozice, není možné zobrazit přesné umístění shod. Poslední a první řádky v jedné shodě mohou obsahovat části, které do shody nepatří, ale obarveny jsou stejně žlutě. Avšak vzhledem k častému formátování kódu, kdy se dbá na to, aby

souvislé části kódu vždy psaly na nový řádek, by to nemělo znamenat příliš velký problém. Pokud by toho chtěl student využít a napsal celou práci do jednoho řádku, grafické zobrazení by nefungovalo, ale stále by systém vykazoval správné procentuální ohodnocení. Každá shoda má přiřazené vlastní číslo a odkaz na část kódu ve druhém souboru.

V druhé části stránky (viz obrázek 11) se nachází tabulka s vlastnostmi obou dokumentů a tabulka vlastností. Jedná se o doplňkové údaje, ze kterých uživatel může rychleji rozhodnout, zda se jedná o úmyslné plagiátorství nebo náhodnou shodu. Na obrázku vidíme dva dokumenty, kde byly přepsány pouze funkce a proměnné. Tabulka vlastností ukazuje stejné číslo, textová podoba je stejná, jen je na první pohled zřejmé, že se funkce jen jinak jmenují. Uživatel proto může okamžitě rozpoznat pokus o opsání.

Obrázek 10 Grafické znázornění podobností kódu.



Obrázek 11 Textová podobnost a tabulka vlastností

5.7 Testování

5.7.1 Testování na vlastních datech

Jak popisuji v metodice práce, testování proběhlo na dvou setech dat. První data byla záměrně vytvořena tak, aby otestovaly schopnost systému odhalovat nejběžnější a nejshodnější typy opisování. Pro jednoduší prezentaci jsem přehled očekávaných výsledků uvedl do tabulky. Je třeba dodat, že procentuální výsledky jsou založeny na shodách řádků, ale jelikož každý řádek může obsahovat libovolný počet tokenů, je toto číslo jen orientační. Minimální délka nalezené shody je 7.

Tab. 1 Předpokládané a získané výsledky vlastních dat

Číslo dat	Shodná data	Předpokládaný orientační výsledek	Získaný výsledek	Modifikace
1	2	100 %	100 %	Přepsané identifikátory, funkce, proměnné, přidané komentáře
1	3	50 %	59,4 %	Stejná polovina
5	3	50 %	49,21 %	Stejná polovina
5	4	50 %	69,8 %	Stejná polovina
1	6	100 %	100 %	Prohození kódu v polovině
7	1,2	Soubor s rozdílným přístup	55,24 %	Náhodné shody dvou stejných zadání
4	1,2,7	0 %	20,62 %, 20,62 %, 17,01 %	Rozdílné dokumenty

Testování na vlastních datech ukázalo, že je systém odolný vůči nejlínějšímu typu opisování, kdy student pouze přepíše názvy a identifikátory, ale nijak nezasáhne do kódu. Na výsledek nemají vliv ani přidané komentáře.

U souborů, kde se testovala obecná schopnost nalézt shody, jsou také výsledky pozitivní. V grafickém porovnání lze vidět, že poloviny kódu jsou přesně identifikovány a spojeny dohromady. Občas se vyskytla i náhodná shoda, která procentuální výsledek lehce narušila, ale při grafickém pohledu může uživatel okamžitě rozpoznat, kolik kódu je ve skutečnosti stejných.

Systém je také odolný na prohazování větších částí kódu. V případě menších je nutné nastavit nižší citlivost na rozpoznávání shod, tzn. snížit minimální délku nalezené shody v nastavení backendu. K tomu je však přistupovat opatrně, aby systém nerozpoznával náhodné změny. Číslo 7 zhruba odpovídá podobnosti třem řádkům, když se nejčastěji na jednom řádku vyskytují tři tokeny. Současné školní řešení JPlag má minimální shodu nastavenou na číslo pět.

U výsledku dvou souborů se stejným zadáním je shoda vyšší: 55,4 %. Tento výsledek je zapříčiněn jednak tím, že byly oba dva soubory tvořeny se stejným zadáním, ale zároveň i se stejným postupem naučeným ve škole, kdy se využívají stejné technologie. Zadání navíc nebylo složité a samotné soubory nejsou příliš dlouhé, ostatně napříč celým souborem se používají asi jen tři funkce.

Poslední řádek tabulky 1 ukazuje náhodné shody napříč zcela rozdílnými dokumenty, jazyk PHP je jednoduchý, a proto je pravděpodobné, že se v něm bude vyskytovat mnoho shod.

Krom těchto ilustrativních dat byl systém po dobu vývoje testován i s dalšími daty, podle kterých se opravovaly chyby v algoritmu a zkoušela obecná funkcionalita.

5.7.2 Zátěžový test se získanými daty

Získaná data obsahují 66 projektů. Vyšší počet dat měl hlavně otestovat funkčnost systému porovnávat větší kvantum souborů a otestovat, zda zvládne porovnat všechny soubory, které jsou do něj nahrány. Soubory jsou ze skutečné školní práce. Během jejich porovnání se neobjevily žádné chyby, systém úspěšně porovnal všechny soubory a správně graficky vyobrazil podobnosti. Jelikož však k datům neexistují žádné výsledky, jak by měly shody přesně dopadnout, nelze z výsledků nic dalšího vyčíst.

6 Diskuze

Rozšíření systému Anton bylo otestováno na dvou setech dat, ale samotné výsledky nemusí mít velkou vypovídací hodnotu. V první části jsem použil několika ilustrativních souborů, ale komplexní test by měl použít souborů mnohem více, je však obtížné vytvořit vlastní data, která by splňovala dané požadavky na shodu. Data by totiž měla respektovat logickou strukturu normálních projektů, v tomto případě univerzitních. Vytvoření mnoha umělých souborů s danými vlastnostmi, by tedy vůbec nemuselo odpovídat realitě, kdy jsou kódy psány jiným, přirozeným způsobem.

Důležitým nastavením rozšíření je minimální délka shody. Zde se do konfliktu dostávají dva zájmy – mít co největší citlivost, resp. odhalovat pouze nenahodilé shody, a zároveň schopnost odhalit nejmenší změny ve dvou dokumentech. V současném algoritmu je toho velmi těžké dosáhnout a dost záleží na velikostech porovnávaných projektů. Čím větší projekt, tím je větší možnost nastavit číslo větší za cenu ztráty informace o drobnějších změnách. V práci jsem nastavil minimální délku na sedm, což zhruba odpovídá tří řádkovým shodám v daných souborech, kdy se jedná o soubory s jednoduchými funkcemi a krátkými řádky kódu. Toto nastavení dokázalo přesně rozpoznat delší shody, ale úplně neeliminovalo náhodné shody. V případě složitějších a větších projektů mohou být vyzkoušeny větší hodnoty a hledat ideální délku, ale v případě projektů řešených na univerzitě je číslo sedm vyhovující.

Jednou z vlastností PHP souborů je časové omezení běhu funkce. V případě většího množství porovnávaných souborů by se mohlo stát, že by funkce nedokázala porovnat všechny soubory. Prodloužení doby výkonu zvládne změnit administrátor serveru.

Při porovnávání prací je však vždy nutné brát na vědomí, že automatizovaný systém neodhaluje opisování, ale pouze shody. Konečný uživatel musí vždy rozhodnout o tom, zda se jedná o plagiát, nebo ne. I 100 % shodné soubory mohly vzniknout bez jakékoliv závislosti. Důležité také je brát ohled i na složitější metody opisování, jako jsou třeba záměny cyklů, vkládání zbytečných proměnných a funkcí nebo opisování jednotlivých myšlenek při řešení složitějších úkolů.

6.1.1 Porovnání se současným školním řešením

Důležitým smyslem tvorby rozšíření byla jeho potřeba. Současné řešení používané v předmětu APV JPlag nepodporuje PHP a porovnává ho jako kdyby se jednalo o normální text, zatímco vytvořené řešení v této práci má přizpůsobení pro formát PHP. Tento rozdíl se například projevuje v porovnání dvou stejných souborů z většího setu dat. Zatímco JPlag ukazuje u dvojice stejných dokumentů pouze 96,5 %, systém Anton ukáže shodu 100 %. Rozdíl nejspíše spočívá v tom, že textová pravidla pro porovnávání kódu nedokáží správně rozkouskovat veškeré PHP tagy, funkce a další znaky. Krom toho ve výsledcích školního řešení lze vidět, že se porovnává i podoba například komentářů v textu, které by neměly v strukturním

porovnání hrát žádnou roli nebo obsah SQL dotazů. Nejedná se tedy o porovnání přizpůsobené zdrojovým kódům a projekty by tak se stejným úspěchem mohly být řešeny i ve standardních nástrojích určených pro textové práce a není třeba zvláštního zacházení. V případě Antona porovnávat zdrojové kódy jazyka PHP už má smysl.

Dalším rozdílem je uživatelské prostředí, které je v systému JPlag nezměněné již desítky let a ukazuje pouze graficky zvýrazněné podobnosti v kódu, zatímco Anton zobrazuje i textovou podobnost názvů funkcí a tabulku vlastností. Uživatel tak má k dispozici větší množství informací, které mu pomohou se rozhodnout, zda se jedná, nebo nejedná o plagiát. V grafické podobnosti taky chybí formátování podle originálního dokumentu. Na jednu stranu je formátování nedůležitou částí kódu, ale taky se z něj dá vyčíst, jak který uživatel formátuje, či podle stylu odsazení a řádkování rozpoznat jednotlivé autory.

Systém zavedený na škole zároveň funguje v offline verzi, což přináší výhodu, že zkoušející může práce porovnat i bez přístupu na internet, na druhou stranu systém Anton výsledky ukládá do online databáze a více uživatelů se může podílet na jednom předmětu. Výsledky jsou pořád ke zhlédnutí a v případě nových dokumentů se pouze aktualizují, JPlag mezitím vytváří vlastní HTML strukturu a soubory pro zobrazení a zabírá na disku více místa.

Systém JPlag ale obsahuje podporu pro velké množství programovacích kódů (zvláště pro Javu) a formáty, které nezvládne, zvládne porovnat alespoň jako text. Systém Anton podporuje pouze PHP, což bylo předmětem této práce a textové formáty samostatně. Nevýhodou Antonu je také to, že zvládá nahrát k projektu jen jeden soubor, zatímco JPlag dokáže přečíst celé složky a ty spojit dohromady. V Antonu se o to musí postarat uživatel sám.

Systém JPlag dále neumožňuje ukládat o dokumentech žádné další informace, Anton ukládá informace o titulu či roku a uživatel je schopen jednotlivé práce hned identifikovat.

6.1.2 Ekonomické zhodnocení

Systém poskytuje více výsledných informací než současně používané řešení, proto by mohl šetřit uživateli čas, který stráví nad rozhodováním o plagiátorství. Například je automaticky provedena analýza textové podobnosti důležitých částí (názvů funkcí), která může poskytnout mnoho užitečných informací o možném opsání. Pokud se shoduje strukturní porovnání, lze z této části vyčíst to, jestli se jedná i opsání úplné (čili funkce jsou stejné) nebo jestli jsou funkce přeložené/zabalené (odpovídají obsahově, ale jsou třeba napsané jinak, nebo v jiném jazyce).

Dalším bodem, kde by se šetřil uživatelův čas je spolupráce mezi více uživateli na jednom předmětu. Po kontrole nebo ohodnocení by každý cvičící do systému nahrál pouze své opravované práce a výsledky by se promítly po spuštění skriptu (který běží automaticky) všem. Nemuselo by tedy docházet ke stahování a upravování prací všech studentů na počítači zkoušejícího, který by měl kontrolu na starost.

Systém by také mohl být užitečný při hledání stejných částí kódu napříč více soubory v jednom projektu. Uživatel by tak mohl najít podobné funkce, které by mohl sjednotit do společného souboru a následně je jen volat. Takový postup by šetřil místem a následné změny v kódu by byly rychlejší a pohodlnější.

7 Závěr

Rozpoznávání studentských projektů je na škole porovnáváno v nedostatečně účinném programu. V práci se dosáhlo vytvoření rozšíření pro systém Anton, které dokáže rozpoznávat podobnosti v programovacím jazyku PHP. V první části práce se nachází přehled stávajících řešení, který pomohl pro vyhodnocení jednotlivých nástrojů a metod používaných pro automatizované rozpoznávání, kde jsem hledal takové řešení, které by bylo vhodné pro systém Anton. Následně došlo k realizaci a implementaci daného řešení.

V práci jsem přiložil instalační pokyny k navrženému rozšíření, nahrání na server s běžícím systémem je pak jednoduché. Systém podporuje grafické a procentuální vyhodnocení podobnosti s uživatelským rozhraním pro uživatele a rozšíření je napsáno v souladu s použitými technologiemi původního systému.

Rozšíření jsem také otestoval na dvou sadách dat, které vyzkoušely rozšíření na základní typy opisování zdrojových kódů a obecnou funkcionalitu porovnávání zdrojových kódů.

Rozšíření by mělo poskytovat lepší a přesnější výsledky při porovnávání projektů v PHP než v současně používaném řešení. Zároveň je připraveno pro budoucí využití v dalších formátech.

7.1.1 Návrhy na další rozšíření

Systém je připraven implementaci dalších formátů a nebylo by na škodě přidat v rámci dalších prací formáty, které se také hodnotí na škole, jako například CSS, HTML, Javu, C++ apod.

Zároveň chybí podpora pro projekty, systém by byl mnohem lepší, pokud by zvládl nahrát najednou všechny soubory daného projektu, roztřídit jejich soubory dle formátu a ukázat společný výsledek.

Dalším diskutabilním bodem je parser/tokenizer. V této práci používám základní tokenizer, který je obsažen přímo v jazyce PHP a jeho výsledky jsou dostatečné. Ale případný tokenizer, který by zvládnul nejen transformovat kód na tokeny, ale popsat tokeny ve správném kontextu, by výsledky ještě zlepšil. Například teď se nijak neliší obsah funkce a obsah mimo funkci, i když se jedná o důležitou informaci.

Příjemným rozšířením pro uživatele by mohlo být vytvoření skriptu, který by dokázal nahrát všechny projekty automaticky z dané složky, tak aby uživatel nemusel nahrávat projekty jeden po druhém v online prostředí.

8 Literatura

ADMINER, ©2017. *Adminer - Správa databáze v jednom PHP soubor*u [online]. [cit. 2017-04-30]. Dostupné z: https://www.adminer.org/cs/

AIKEN, A., SCHLEIMER, S., WILKERSON, D.S., 2003. *Winnowing: Local Algorithms for Document Fingerprinting*. Stanford, Stanford University.

ANTON, 2017. *Zdrojové kódy a provozovaná aplikace na školním serveru* [software].

ARWIN, CH., TAHAGHOGHI, S.M.M., 2006. *Plagiarism Detection across Programming Languages*. Melbourne: School of Computer Science and Information Technology RMIT University. Dostupné také z: http://crpit.com/confpapers/CRPITV48Arwin.pdf.

CLONEDR, ©2016. *Clone Doctor: Software Clone Detection and Reporting* [online]. Austin: Semantic Designs, [cit. 2017-05-14]. Dostupné z: http://www.semdesigns.com/products/clone/

CLOUGH, P., 2000. *Plagiarism in natural and programming languages: an overview of current tools and technologies*. Sheffield: Department of Computer Science, University of Sheffield, Dostupné také z: http://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf.

FLORYČEK, J., 2015. *Optimalizace antiplagiátorského řešení na Mendelově univerzitě v Brně.* Brno: Mendelova univerzita v Brně. Dostupné také z: http://theses.cz/id/vgizl0/zaverecnaprace.pdf.

FOLTÝNEK, T., PROCHÁZKA. T., RYBIČKA, J., 2009. *Plagiarism Detection System at Mendel University in Brno, Czech Republic.* [DVD-ROM]. In IVKI 2009. Inovácia výskumu katedier informatiky. s. 50-53. ISBN 978-80-8094-579-4.

CHÝLA, R., 2009. *Detekce plagiátorství*. [online]. Praha: Nakladatelství Ikarus, ISSN 1212-5075. [cit. 2016-11-17]. Dostupné z: http://ikaros.cz/detekce-plagiatorstvi.

JONES, E. L., 2001. *Metrics based plagiarism monitoring*. In Proceedings of the Sixth Annual CCSC Northeastern Conference. Middlebury, Vermont, s. 1–8.

JOY, M., MIRZA, o., 2015. *Style Analysis For Source Code Plagiarism Detection*. In: Plagiarism across Europe and Beyond: Conference Proceedings. Brno: MENDELU Publishing Centre, s. 53–61. ISBN 978-80-7509-267-0.

JPLAG, 2017*. JPlag - Detecting Software Plagiarism* [online]. Karlsruhe: Institute for Program Structures and Data Organization, [cit. 2017-05-14]. Dostupné z: https://jplag.ipd.kit.edu/

LÝSEK, J., 2017. *Ústní sdělení dne 13. dubna 2017*

MOSS, 2017. *A System for Detecting Software Similarity* [online]. [cit. 2017-04-30]. Dostupné z: http://theory.stanford.edu/~aiken/moss/

MURAO, H., OHNO, A., 2011. *A two-step in-class source code plagiarism detection method utilizing improved cm algorithm and sim*. International Journal of Innovative Computing, Information and Control. ICIC International, 7(8), 4729–4739. ISSN 1349-4198. Dostupné také z: http://www.ijicic.org/ijicic-10-05012.pdf.

NETTE FOUNDATION, ©2017. *Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework* [online]. Nette Foundation, [cit. 2017-04-30]. Dostupné z: https://nette.org

PRECHELT, L., MALPOHL, G., PHILIPPSEN, M., 2000. *JPlag: Finding plagiarisms among a set of programs*. Karlsruhe: Fakultat fur Informatik Universit at Karlsruhe. Dostupné také z: http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf.

PROCHÁZKA, D., 2012. *SEO: cesta k propagaci vlastního webu*. [online]. Grada Publishing a.s., [cit. 2014-11-27]. Dostupné z WWW: https://books.google.cz/books?id=KSN0AgAAQBAJ

RIVEST, R., 1992. *The MD5 Message-Digest Algorithm* [online]. MIT Laboratory for Computer Science and RSA Data Security, Inc., [cit. 2016-10-24]. Dostupné z: https://www.ietf.org/rfc/rfc1321.txt

SHAO, Z., 2015. *Compilers and interpreters*. [online]. New Haven: Yale University, [cit. 2016-11-17]. Dostupné z: http://flint.cs.yale.edu/cs421/lectureNotes/c02.pdf.

THE PHP GROUP, ©2017. *PHP - tokenizer* [online]. [cit. 2017-05-14]. Dostupné z: http://php.net/manual/en/book.tokenizer.php

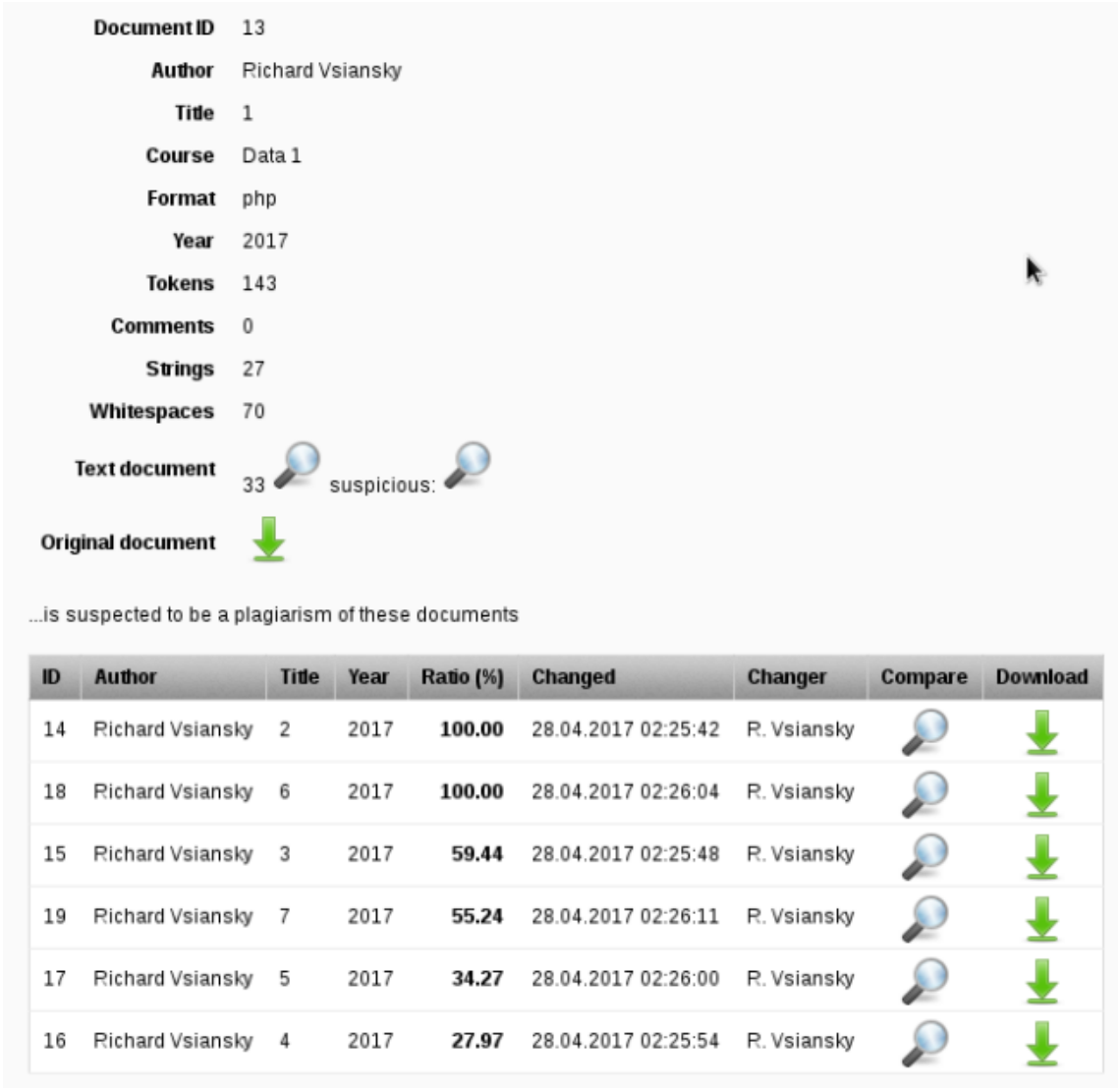
VŠIANSKÝ, R., DLABOLOVÁ, D., 2016. Deployment and improvements of system Anton. PEFnet 2016. Brno: Mendelova univerzita v Brně.

WHALE, G., 1990. *Identification of Program Similarity in Large Populations*. The Computer Journal, 33(2). str.140-146. Dostupné také z: http://comjnl.oxfordjournals.org/content/33/2/140.full.pdf.

9 Seznam obrázků

Přílohy

A pohled na výsledky srovnání v rámci jednoho dokumentu



Document ID	13
Author	Richard Vsiansky
Title	1
Course	Data 1
Format	php
Year	2017
Tokens	143
Comments	0
Strings	27
Whitespaces	70
Text document	33 suspicious:
Original document	

...is suspected to be a plagiarism of these documents

ID	Author	Title	Year	Ratio (%)	Changed	Changer	Compare	Download
14	Richard Vsiansky	2	2017	**100.00**	28.04.2017 02:25:42	R. Vsiansky		
18	Richard Vsiansky	6	2017	**100.00**	28.04.2017 02:26:04	R. Vsiansky		
15	Richard Vsiansky	3	2017	**59.44**	28.04.2017 02:25:48	R. Vsiansky		
19	Richard Vsiansky	7	2017	**55.24**	28.04.2017 02:26:11	R. Vsiansky		
17	Richard Vsiansky	5	2017	**34.27**	28.04.2017 02:26:00	R. Vsiansky		
16	Richard Vsiansky	4	2017	**27.97**	28.04.2017 02:25:54	R. Vsiansky		

B obsah elektronické přílohy

`Anton` – složka se soubory rozšíření, nutné nakopírovat a přepsat do složky s antonem

`Dataprotestovani.zip` – ilustrativní příklady použité v testování

`Zatezova_data.zip` – výsledky z JPlagu použité při porovnání