VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKA POKROČILÝCH KONSTRUKCÍ JAZYKA PYTHON NA ZÁKLADĚ POSKYTOVÁNÍ ZPĚTNÉ VAZBY KE STUDENTSKÝM KÓDŮM

TEACHING ADVANCED PYTHON THROUGH AUTOMATIC FEEDBACK TO STUDENT CODES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR JOHN**
AUTHOR

VEDOUCÍ PRÁCE **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **John Petr**
Program: Informační technologie
Název: **Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům**
Teaching Advanced Python through Automatic Feedback to Student Codes
Kategorie: Informační systémy
Zadání:

1. Seznamte se se způsoby poskytování zpětné vazby programátorům v oblasti vysoceúrovňových dynamických jazyků a existujícími implementacemi obecných postupů.
2. Zpracujte přehled typů příkladů, které se používají v kursech zaměřených na výuku programování, aby si studenti osvojili pokročilé konstrukce těchto jazyků.
3. Navrhněte a implementujte systém, který bude schopen analyzovat zaslaný studentský kód, ověřit správnost a poskytnout zpětnou vazbu k nevhodně zvoleným konstrukcím, s případným odkazem na materiály vysvětlující vhodnější řešení.
4. Vyhodnoťte vytvořený systém v interakci s reálnými studenty, seznamujícími se s programováním v Pythonu.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle dohody s vedoucím

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Tato bakalářská práce se zabývá problematikou vyučování jazyka Python pomocí systému, který studentům poskytne zpětnou vazbu k jejich řešení. Cílem práce bylo vytvořit systém schopný vyhodnocovat studentská řešení a poskytovat zmíněnou zpětnou vazbu. Důraz práce je kladen na zpětnou vazbu, která je studentům poskytována, a také na možnosti spojené s omezením zdrojů použitých při řešení. Vytvořený systém poskytuje zpětnou vazbu na základě analýzy abstraktních syntaktických stromů sestavených z odevzdaného řešení, umožňuje připojit doporučení z programů třetích stran, a také umožňuje nastavit omezení zdrojů, knihoven a funkcí, které mohou být při řešení použity. Systém byl použit v letním semestru při výuce předmětu ISJ a 60 % studentů vylepšilo svoje řešení na základě doporučení ze systému. Tyto skutečnosti dokazují, že systém je možné využít při výuce jazyka Python v praxi.

Abstract

This bachelor thesis focuses on the topic of teaching Python programming language assisted by automated system that can provide feedback to submitted solutions. The goal of this thesis was creation of automated system that could evaluate student solutions and provide feedback. The main emphasis was on provided feedback and options for limiting resources that can be used. Created system provides feedback based on the analysis of abstract syntax trees assembled from submitted solutions, allows an administrator to attach tips from third party programs and set restrictions on resources, libraries and functions, that can be used. System was used during summer semester in ISJ course and 60 % of students improved their solution based on feedback given by system. This suggests that created system can be used during Python tuition.

Klíčová slova

jazyk Python, automatické vyhodnocování, automatické výukové systémy, zpětná vazba, doporučení ke kódu

Keywords

Python programming language, automated evaluation, intelligent tutoring systems, feedback, code suggestions

Citace

JOHN, Petr. *Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr John
3. června 2020

Poděkování

Rád bych poděkoval panu Doc. RNDr. Pavlu Smržovi, Ph.D za poskytnutou pomoc a konzultace při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Kvůli stále rostoucímu počtu zájemců o informační technologie a programování klesá úroveň individuálního přístupu, kterého je možné dosáhnout ve školním prostředí. Tento přístup postrádají i ti zájemci, kteří se rozhodnou naučit se programovat bez pomoci vyučujících. Individuální přístup poskytuje mnoho výhod, které mohou zájemcům pomoci v cestě k jejich cíli. Mezi tyto výhody patří možnost od vyučujícího získat názor na řešení zájemce, a to takzvanou zpětnou vazbu. Touto zpětnou vazbou se rozumí například doporučení použití lepších struktur nebo varování před jejich špatným použitím. Jednou z možností, jak získat tuto zpětnou vazbu, je vyvinutí systému, který by byl schopen ji automatizovaně poskytovat. Návrhem a vývojem takového systému se zabývá tato bakalářská práce.

Podobné výukové systémy jsou v dnešní době kvůli zmíněným důvodům velmi oblíbené nejen při výuce programovacích jazyků. Výuka programování bude i do budoucna velmi důležitá a snahy o zvýšení efektivity jejího vyučování budou nadále pokračovat, stejně jako u výuky v ostatních oborech.

Zpětné vazba může představovat například informace o tom, zdali byly pro řešení problému použity správné konstrukce. Je ji možné rozdělit do dvou kategorií. První kategorii lze chápat jako obecnou zpětnou vazbu. Tato zpětná vazba je relevantní v jakémkoli řešení a představuje způsoby řešení, které by neměly být použity. Druhou kategorií je zpětná vazba zaměřená na doporučení týkající se konkrétního řešení. K vytvoření a poskytování této zpětné vazby je zapotřebí znát konkrétní problematiku, kterou má daný kód řešit.

Při vytváření práce jsem se také zaměřil na bezpečnost spuštění řešení odevzdaných do systému, a to omezením spuštěných částí. Bez tohoto omezení by studentská řešení mohla například využívat knihoven, které při řešení nejsou povoleny nebo poškodit nebo znepřístupnit zařízení, na kterém by bylo toto vyhodnocování provedeno. Díky tomuto rozhodnutí se systém liší od bakalářských prací řešených v předchozích letech. Téma mě velice zaujalo zejména kvůli mému zájmu o efektivní výuku programovacích jazyků, a to nejen ve školních podmínkách. Tyto systémy jsou podle mě perspektivní z toho důvodu, že je možné od nich získávat zpětnou vazbu velmi rychle, na rozdíl od ručního vyhodnocení. Je tak možné kód vylepšit a poté některá vylepšení konzultovat.

Jelikož je možné systém použít k bodovému ohodnocení studentských řešení bylo při vývoji nutné poskytnout administrátorovi systému možnost vyhledat ve vyhodnocených řešeních plagiáty a také provést vyhodnocení všech odevzdaných řešení najednou. Použití tohoto vyhledávání dává smysl v projektech, které je možné řešit více způsoby a je díky němu možné zamezit případnému doslovnému převzetí řešení z předchozích běhů kurzu nebo sdílení řešení mezi studenty. Vyhodnocování většího množství studentských řešení

jednoho projektu může být použito vyučujícím předmětu při vyhodnocování projektu po jeho konečném termínu.

Systém byl na začátku vývoje rozdělen na dvě části, první část byla vytvořena pro použití studenty pro vyhodnocování jejich projektů a druhá byla určena pro vyhodnocování projektů vyučujícím. Toto rozdělení bylo v projektu zachováno, ale do obou částí byly přidány možnosti. Jednou z těchto přidaných možností je zmíněná detekce plagiátů, která byla přidána do části určené pro vyučujícího. V druhé části byla upravena struktura zpětné vazby. Bylo přidána možnost rozdělit detekci konkrétní zpětné vazby do více kroků a to hlavně proto, že v některých příkladech by bylo možné poskytovat velké množství zpětné vazby, které by mohlo studenta odradit od použití.

V první kapitole je obsažen rozbor současného stavu a potřebný přehled teorie, v druhé části popis návrhu tohoto systému, jeho rozdělení na moduly a popis případů použití, které musí být systém schopen obsloužit, v třetí části je popsán postup implementace vytvořeného systému a čtvrtá část obsahuje postup použitý při testování vytvořeného systému.

Kapitola 2

Rozbor řešené problematiky

V této kapitole jsou představeny hlavní existující metody, kterými se vyučují programovací jazyky, společně s jejich efektivitou, a chyby studentů, které se často vyskytují v jejich řešeních. Dále jsou popsány problémy s bezpečností automatizovaného vyhodnocení studentských řešení a také automatická detekce plagiátů a existující řešení, která se při automatické detekci používají.

2.1 Efektivita studijních metod

Výuka programovacích jazyků probíhá několika hlavními způsoby. Tyto způsoby, jejich použití na Vysokém učení technickém fakultě informačních technologií v bakalářském studiu a porovnání efektivity podle [1].

Laboratoře

Výuka v laboratořích je u předmětů častá, díky zmenšeným skupinkám je možné věnovat se každému ze studentů individuálněji než velkému počtu studentů přítomných na přednáškách. Studenti si také mohou problém vyzkoušet a při otázkách se mohou na zpětnou vazbu zeptat. Díky malým skupinkám toto řešení vyžaduje od lektorů velké množství času. Laboratoře byly hodnoceny jako nejlepší 88 % účastníků průzkumu.

Projekty

Použití projektů ve výukovém programu je velmi časté, studenti řeší většinu projektů individuálně, u některých je povoleno řešit tento projekt ve skupinkách. Oba typy projektů přináší vlastní výhody. Projekty pro jednotlivce zaručí, že student, který projekt vyřeší pochopí danou problematiku, týmové projekty zase rozvíjí schopnost studentů pracovat v týmu a také použít verzovací systémy a ostatní nástroje pro spolupráci. U projektů ale není možno zaručit takové množství zpětné vazby jako u laboratoří. Projekty byly hodnoceny druhé, studenti věří, že velké množství projektů jim pomůže pochopit koncepty a rozšířit jejich už existující schopnosti a proto 74% studentů hodnotilo jako 2. nejlepší způsob.

Přednášky

Přednášky jsou nejčastějším způsobem, se kterým se studenti setkají. Jedná se o metodu vyučování na základě předávání zkušeností s podporou vizualizace, například prezentace.

Efektivitu přednášek ovlivňuje velké množství faktorů. Mezi tyto faktory patří například počet studentů na přednášce, čas, ve kterém se tato přednáška odehrává a další. Přednášky byly hodnoceny jako 3. nejlepší způsob 65 % studentů.

Ostatní efektivní metody

Mezi tyto metody patří tutoriály a semináře s 63 % a problémové vyučování s 56 %. Tyto způsoby jsou na fakultě zastoupeny méně, tutoriály často pořádá studentská unie nebo sami studenti. Koncepty problémového vyučování se zřídka projevují v přednáškách a laboratořích.

Zvýšení efektivity projektů

Projekty jsou tedy jak efektní tak oblíbené studenty i lektory. Díky této skutečnosti je snaha efektivitu tohoto dále zvýšit. Projektům chybí zpětná vazba, která by byla studentům dostupná ještě před odevzdáním a ohodnocením jejich řešení. Někteří lektoři tento nedostatek řeší takzvaným pokusným odevzdáním. Student, který projekt odevzdá k němu dostane alespoň částečnou zpětnou vazbu a to ať v podobě bodů tak v podobě výstupu testové sady. Toto řešení je dobré, ale student nemusí toto odevzdání stihnou a také zvyšuje nároky na lektory. Proto je snaha vytvořit systém, který by mohl studentům poskytovat zpětnou vazbu automaticky. Díky tomuto systému by studenti mohli při řešení projektu postupovat svým tempem a zároveň by měli přístup ke zpětné vazbě. Systémy zabývající se touto problematikou patří do skupiny takzvaných inteligentních výukových systémů (intelligent tutoring systems) a jsou známy pod zkratkou ITS vycházející z jejich anglického názvu. Tyto systémy jsou v praxi využívány kromě zmíněného použití například pro výuku přirozených jazyků, přírodních věd a dalších. Při implementací těchto systémů je často využita umělá inteligence, která má za úkol studentům poskytovat odpovídající zpětnou vazbu, ale systémy často spoléhají na definici problémů, které má student řešit, lektorem, který je využívá. Více o ITS dotupné v [5]. Systémy se často dělí na dvě skupiny podle strategie [17], kterou využívají k poskytování zpětné vazby. První strategie je založená na vyzývání studenta k odpovědi (Prompting-Answer Strategies) a druhá na poskytnutí odpovědi (Giving-Answer Strategies). Dále jsou použity jen anglické názvy strategií a to z důvodu jejich častějšího použití.

- Giving-Answer Strategies – Tato strategie je založená na přímém zvýraznění chyby. Pokud studentem odevzdané řešení obsahuje chybu je tato chyba označena a úkol je studentem opakován dokud chybu neopraví nebo mu je správné řešení přímo představeno.
- Prompting-Answer Strategies – Tato strategie je založená na snaze získat od studenta správnou odpověď. Student je upozorněn na to, že jeho řešení obsahuje chybu, ale kde je tato chyba obsažena, nebo její řešení zůstává na studentovi. Studentovi mohou být za tímto účelem poskytovány doporučení, které mu ale nemohou řešení zcela odhalit.

2.2 Existující řešení

Zlepšit efektivitu výuky programovacích jazyků se již snaží velký počet systémů. Tyto systémy je možné rozdělit do několika základních typů popsaných níže. Rozdělení založeno na [12, 11]. Všechny příklady jsou systémy ITS využívající umělé inteligence ke své funkci.

Systémy založené na příkladech

Tento způsob je založený na vysvětlování problémů a jejich řešení, poté co jsou studenti seznámeni s problémem je studentovi prezentován podobný problém, který má za úkol vyřešit. Tímto způsobem studenti naučí aplikovat stejné řešení i pro problémy stejného typu. Příkladem tohoto systému je webový nástroj *NavEx* [19]. Každý příklad obsahuje anotace s vysvětlením každého důležitého řádku. Studenti mohou otevírat vysvětlení řádek po řádku, nebo mohou otevřít pouze anotace řádků, které jim dělají problémy. Kód díky tomu může být zobrazen jako blok, namísto od klasického roztrhaného zobrazení, které je často použito v studijních materiálech. Sytém také poskytuje možnost studentovi vybírat pouze příklady, které odpovídají jeho úrovni.

Dalším způsobem je prezentování studenta s problémem a s šablonou, do které má student doplnit kód tak, aby výsledné řešení odpovídalo požadavkům. Zástupcem tohoto nástroje je *ADAPT* [7], který byl vyvinut pro výuku jazyka prolog. Lektor v při použití tohoto systému studenty nejdříve seznámí s několika řešeními. Poté jsou studenti prezentováni šablonou řešení, která obsahuje vynechané části. Pokud student potřebuje pomoc při řešení problému systém tento problém rozloží na komponenty a vysvětlí je studentovi. Pokud student požádá o pomoc je doplněna část kódu. Student takto může žádat až do úplného doplnění, kdy je mu problém vysvětlen.

Systémy založené na simulaci

Tyto systémy řeší situaci kdy studenti nemůžou pochopit, jak funguje část kódu kvůli tomu, že se jeho části, například stavy proměnných, mění dynamicky. Jedním ze způsobů, jak snížit úroveň abstrakce a zjednodušit tak problém pro studenty je zavedení konkrétních objektů. Příkladem takového systému je *Scratch* [13], který se snaží o vysvětlení za pomocí vizualizace. Bloky kódu se vizuálně zanořují a jsou od sebe odděleny barvou. Cílem je objasnit studentům tok programu. Studenti, kteří začínají s programováním si mohou díky těmto systémům vytvořit základní programovací schopnosti.

Systémy používající dialogy

Tyto systémy se snaží o komunikaci se studentem za použití dialogů. Jak systém tak student mohou položit otázku. Příkladem je program *PROPL* [10], který studentům pomáhá vytvořit řešení v pseudokódu podobnému přirozenému jazyku. Systém se snaží za pomocí dialogu opravit studentovy chyby a mylné představy. Pokud studentovo řešení není pochopeno nebo není podle systému správné systém použije dialogy, podle kterých se snaží upřesnit samotné řešení.

Systémy založené na analýze programu

Analýza existujících řešení může být pro studenty přínosná. Na tomhle způsobu je založena tato metoda. Tyto systémy se spíše snaží o vysvětlení jednotlivých konstrukcí jazyka než na programování jako takové. Studenti jsou prezentování s částí kódu, který analyzují a ladí. Úkolem studenta může být například vyhodnotit program řádek po řádku, odhadnout, jak se program zachová nebo identifikovat chyby v kódu a zodpovědět, proč tyto chyby nastaly. Vstup od studenta se porovná s očekávaným vstupem. Podle rozdílu těchto vstupů je studentovi vygenerována zpětná vazba.

Systémy založené na zpětné vazbě

Toto řešení je založené na poskytování zpětné vazby ke studentským řešením jako jediný prostředek k učení. Tyto systému spoléhají na to, že zpětná vazba systému bude přínosná pro studenty při řešení zadaného úkolu a pomůže jim při zlepšování jejich už existujících schopností. Systém může například obsahovat sady ideálních a chybových pravidel. Ideální pravidla jsou ta pravidla, která reprezentují ideální řešení problému. Chybová pravidla představují způsoby, které by při řešení neměly být použity. Studentovo řešení je poté vyhodnoceno a použitá pravidla jsou porovnána s předem zmíněnými skupinami. Pro chybná pravidla použitá studentem je vygenerována zpětná vazba a student je nasměrován na správné řešení.

Příkladem tohoto systému je *ELM-ART* [4], který přidává k již existujícím skupinám pravidel třetí takzvanou suboptimální skupinu. Tato skupina slouží k označení způsobů, které problém řeší, ale jsou pomalé nebo náročné na zdroje. Tento systém provádí kontrolu ve dvou krocích identifikace snahy studenta a detekování chyb. Při identifikaci snahy systém na základě předem definované sady vygeneruje různé způsoby, které poté použije k vytvoření hypotézy o řešení. Student vyřešil problém správně pokud tento hypotetický algoritmus odpovídá studentskému programu. Jinak systém prohledává chybová pravidla která použije k vygenerování zpětné vazby.

Systémy založené na spolupráci

Zatímco předchozí systémy se snaží o podporování jednotlivců, systémy založené na spolupráci se snaží o podporu konverzace mezi studenty. Tyto systémy se snaží pomoci studentům diskutovat rozumnou, ohleduplnou a opodstatněnou formou. K tomuto účelu jim poskytují společný pracovní prostor, ve kterém mohou vyjádřit svoje nápady a myšlenky například grafickou formou za pomocí tvarů. Tyto tvary mají předem definovaný význam (například argument nebo tvrzení) a text zadaný studentem, který představuje samotnou myšlenku. Studenti mohou tyto tvary spojovat a ke každému spoji se vyjádřit, například souhlas nebo nesouhlas. Tyto systémy se musí vypořádat se studenty, kteří se nezapojují a s možným odbočením od tématu. Při použití systémů se doporučuje dostatečné oddělení studentů a to z toho důvodu, aby studenti byli nuceni používat systém jako primární způsob komunikace a bylo jednodušší vyhodnotit, jejich přínos k diskusi. Vyhodnocená diskuse je poté předána pedagogovi, který ji může použít k vylepšení příkladu.

2.3 Chyby studentů v kódu

Chyby studentů se často opakují a proto je nutné je rozdělit do tříd, které budou nápomocny pro jejich odhalování a také je možné na základě jejich studie navrhnout zpětnou vazbu. Chyby studentů jsou závislé na gramatice použitého jazyka. Rozdělení níže je zaměřené na chyby objevující se v jazyce Python. Více o obecných chybách, ale i o chybách v jiných programovacích jazycích dostupné v [2].

Syntaktické chyby

Tyto chyby vznikají v případě, že zdrojový kód programu neodpovídá gramatice programovacího jazyka. V projektech se tyto chyby často objevovaly v podobě nekonzistentního použití mezerníků a tabulátorů. Jejich kombinování nemusí syntaktickou chybu způsobit,

ale zdrojový kód poté často obsahuje neodpovídající odsazení. Méně častou chybou bylo použití nedefinovaných proměnných nebo funkcí.

Logické chyby

Chyby tohoto typu je možné klasifikovat do tří hlavních tříd. Každá z těchto tříd představuje jiný logický problém s řešením. Více o logických chybách v [6].

- Chyby algoritmu – algoritmus vybraný studentem je špatný na abstraktní úrovni. Algoritmus, který se student snaží použít problém neřeší.
- Misinterpretace – student zadaný problém pochopí jinak, než je myšleno zadání a jeho řešení díky tomu neodpovídá zadaným požadavkům. Řešení může na rozdíl od předchozího problému v některých případech fungovat, ale plně neodpovídá zadání. Toto se v tomto roce projevilo například u projektu 8, kde měli studenti za úkol vytvořit generátorovou funkci, která bude ze vstupní kolekce postupně vybírat první objekt odpovídající zadané porovnávací funkci. Někteří studenti místo generování jednotlivých položek vraceli kolekci všech těchto objektů. Řešení by odpovídalo případě, kdy by zadaná kolekce obsahovala pouze jeden prvek nebo byla prázdná.
- Mylná představa – tyto chyby představují neznalost programovacího jazyka. Student si není vědom toho, jak přesně se chová konstrukce jazyka. V projektech se například často opakovalo použití seznamu jako implicitní hodnotu funkce. V jazyce Python v této situaci dojde k vytvoření seznamu při definování funkce. Tento seznam je poté předáván referencí. Při prvním volání funkce, je seznam prázdný, při dalších volání však seznam obsahuje všechny přidané objekty.

2.4 Bezpečnost

Při spouštění studentských kódu je zapotřebí dbát na zabezpečení zařízení na kterém jsou tyto kódy vyhodnocovány. Studentské kódy mohou obsahovat části, které jsou nebezpečné pro zařízení, na kterém se vyhodnocují. Kód může tyto části obsahovat jak záměrně tak omylem, například kvůli chybě nebo studentem neočekávanému chování kódu.

Virtualizace

Jedním ze způsobů, jak se s touto skutečností vypořádat je virtualizace prostředí, ve kterém jsou studentské kódy vyhodnocovány. V případě, že kód obsahuje škodlivou část je její vykonávání provedení ve virtuálním prostředí, kde nemůže omezit funkci vyhodnocovacího systému. Některé operační systémy poskytují možnost vytvořit virtuální prostředí nativně. Na platmě Linux je možné použít nástroj `seccomp`[1] (secure computing mode). Tento nástroj limituje operace, které může ním virtualizovaný program provést. Pokud se virtualizovaný program snaží provést nepovolenou operaci je ukončen signálem `SIGKILL` nebo mu zaslán signál `SIGSYS`. Další možností řešení je použití programu, který vytvoří virtuální stroj, nebo kontejner. Rozdíl mezi virtuálním strojem a kontejnerem je následovný, kontejnery mezi sebou mohou sdílet jedno jádro operačního systému, jinak jsou od sebe izolované, Na druhou stranu každý virtuální stroj má vlastní jádro operačního systému, díky tomuto používají kontejnery méně zdrojů. Mezi programy pro vytváření kontejnerů patří například rozsáhle

[1] Více v manuálových stránkách http://man7.org/linux/man-pages/man2/seccomp.2.html

používaný program Docker[2], který je dostupný na velkém počtu platforem včetně mac OS a Windows. Při své činnosti docker spouští kontejnery v jednom Linuxovém virtuálním stroji a k virtualizaci používá vestavěné virtualizační funkce tohoto jádra. Při použití virtualizace přichází v úvahu využít nástroje vmware [3].

Omezení zdrojů

Dalším způsobem je omezení zdrojů, je kterým má student přístup a případné zastavení vyhodnocování v případě, že jsou tyto limity překonány. Limit v tomhle případě může být chápán jako jednotka využití zdroje, například jako maximální množství paměti, kterou může program při svém vykonávání využít, nebo jako seznam knihoven, které může student při svém řešení použít.

2.5 Abstraktní syntaktický strom

Abstraktní syntaktický strom je stromová reprezentace syntaktické struktury zdrojového kódu. Tato reprezentace nemusí obsahovat všechny syntaktické detaily. Například v ní mohou být vynechána interpunkční znaménka. V jazyce Python můžou tato znaménka být například středník, který je použitý pro oddělení příkazů na stejném řádku nebo čárka, která se používá pro oddělení položek v seznamu [8]. Každý uzel představuje konstrukci programovacího jazyka. Jazyk Python obsahuje modul ast[4], který dovoluje jednoduše vytvářet tyto stromy z validních zdrojových kódu. Pokud zdrojový kód obsahuje syntaktickou chybu není možné sestavit abstraktní syntaktický strom. Tento modul je používán standardním interpretem při kompilaci.

Následující fragment kódu

```
print("sudé" if odd else "liché")
```

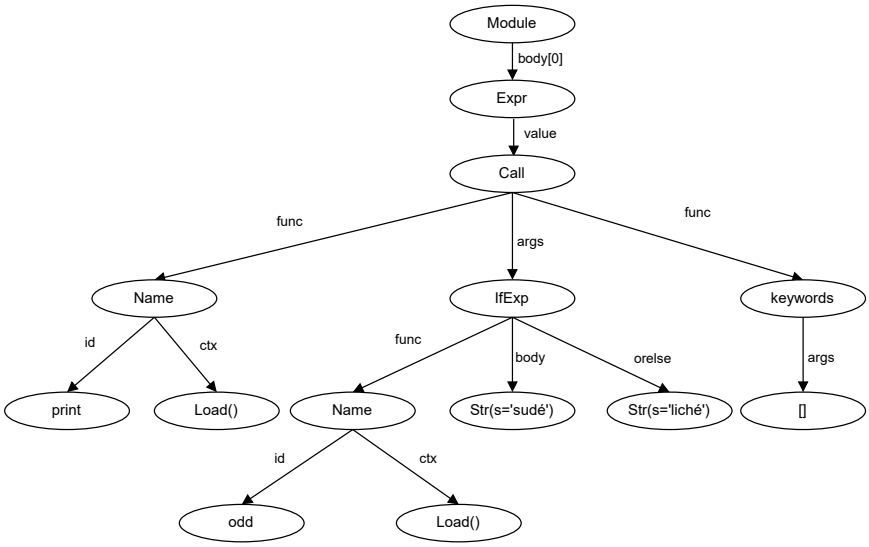
Odpovídá ve verzi jazyka 3.6.9 stromu

```
Module
(
  body=
  [
    Expr
    (
      value=Call
      (
        func=Name(id='print', ctx=Load()),
        args=
        [
          IfExp
          (
          test=Name(id='odd', ctx=Load()),
          body=Str(s='sudé'),
          orelse=Str(s='liché')
```

[2] Více o programu Docker https://www.docker.com/
[3] Více o programu vmware https://www.vmware.com/uk/products/workstation-pro.html
[4] Více o modulu ast [5]

```
            )
        ],
        keywords=[]
    )
  )
 ]
)
```

Grafickou reprezentaci tohoto stromu je možné nalézt v 2.1. Zápis vytvořeného stromu je značně složitý oproti zdrojovému příkazu, ale na rozdíl od zápisu v jazyce Python se jedná o dobře programově zpracovatelné třídy. příkladem odstraněných detailů jsou například závorky, které v abstraktním syntaktickém stromě nejsou obsaženy.



Obrázek 2.1: Grafická reprezentace tohoto stromu.

2.6 Detekce plagiátů

Se zvyšujícím se počtem zájemců o studium v oboru informačních technologií roste i počet studentů, kteří se při řešení projektů uchýlí k plagiátorstvím zdrojových kódů, nebo jejich částí. Studentské kódy se tedy v tomto smyslu neliší od projektů ostatních typů, a proto s nimi sdílí důvody, kvůli kterým se studenti uchylují k plagiátorství.

Mezi tyto důvody patří

- Neznalost problému – student nezná zadané téma a neví, jak daný problém vyřešit takovým způsobem, aby měl šanci dostat požadovaný výsledek. Místo snahy pochopit problém se student poté snaží nalézt řešení co možná nejpodobnějšího problémů a toto řešení s malými nebo žádnými změnami odevzdat.
- Nedostatek času – student je v časové tísni a nestíhá projekt sám vypracovat.
- Neznalost rozdílu mezi inspirací se existujícím řešením a plagiátorstvím – student část kódu převezme, ale zatají tuto zkutečnost, nebo ji uzná nekorektním způsobem, například odkazem ve zdrojovém kódu.

O dalších důvodech, kdy se studenti uchýlí k plagiátorství pojednává [14]. K detekci plagiátů se používají stejné způsoby jako při detekci duplicit v kódu a to z toho důvodu, že obě metody jsou založeny na stejném principu a to na zjištění podobnosti mezi porovnávanými kódy. Tato podobnost může být vyjádřena procentuální hodnotu. Pro rozhodnutí, zdali je je některý z kódů duplikát nebo plagiát je nutno zavést hranici této podobnosti. Pokud je podobnost pod touto hranicí dvě řešení nejsou tato řešení považována za plagiáty, v opačném případě jsou tato řešení systémem jako plagiáty označena.

Způsoby detekce duplikátů

Duplikáty by bylo možné odhalovat ručně procházením, toto řešení ale není praktické v případě velkých projektů, nebo velkého počtu menších projektů. V tomto případě je tedy nejlepším způsobem provést detekci duplikátů programem. Způsoby detekce zahrnují

- Detekce pomocí vyhledávání podstromů v abstraktních syntaktických stromech.
  Tato verze detekce jde použít na vyhledávání v dokumentech, ze kterých je možné sestavit syntaktické stromy, například programovací jazyky. Ze zdrojového kódu kontrolovaného programu se nejdříve sestaví syntaktické stromy a podstromy tohoto stromu srovnávají se všemi ostatními podstromy. V případě detekce duplikátů mezi studentskými kódy se porovnávají podstromy řešení dvou studentů. Toto řešení má dva problémy. Prvním problémem je detekce podstromů, které jsou si velmi podobné, nejsou shodné, ale i přesto by měly být označeny za duplikáty. Tento problém jde vyřešit porovnáváním podobností podstromů a ne na jejich přesnou shodu. Dalším problémem je problém měřítko, na které je detekce použita. Při velkých měřítkách se hlavně prostudování náročnost velmi zvyšuje a řešení se stává neefektivním. Více o problémech a jejich řešeních v [3].

- Porovnávání řetězců
  Porovnávání řetězců je metoda, ve které se porovnávají části řetězců zvané k-gramy. K-gram je řetězec sousedících znaků o délce k. Při této metodě se dokument rozdělí do těchto k-gramů, kde k je zadáno uživatelem a tyto k-gramy jsou následně srovnávnány za pomocí algoritmu pro vyhledávání textu, často například Rabinův-Karpův algoritmus [9]. Při použití tohoto algoritmu se z každého k-gramu vypočítají otisk (hash), které se poté srovnávají s otisky v jiném souboru. Pokud se tyto otisky shodují je zvýšena podobnost souborů ve kterých se nachází.

- Winnowing
  Tento algoritmus stejně jako předchozí využívá k–gramů, na rozdíl od předchozího ale využívá dvě konstanty a to k – délku a t – garantovaný práh se vztahem $k <= t$ a takzvané okno. Okno představuje předem stanovený počet po sobě následujících otisků. V každém okně je poté vybrána minimální hodnota otisku. Pokud je v okně více otisků s minimální hodnotou je vybrán poslední nalezený. Tyto otisky všech oken poté reprezentují otisk dokumentu. Více o posledních dvou algoritmech na [16].

Problémy s detekcí duplikátů

Při detekci duplikátů se setkáváme s hlavními dvěma problémy. Oba problémy jsou popsány níže.

První problém se týká přesnosti detekovaných duplikátů. Jakákoli detekce má šanci selhat a za duplikát označit řešení, které není duplikátem a naopak. Pokud detekce selže

a duplikáty nebyly odhaleny se nazývá falešné negativum (False-Negative). V opačném případě vznikají falešná pozitiva (False-Positives) tato situace je velmi ovlivněna zadáním, ve kterém se má detekce provádět. Pokud je toto zadání příliš jednoduché je možné, že na něj existuje jeden způsob, který je jednoduchý na implementaci a toto řešení je velkému počtu studentů hned zjevné.

Druhým problémem jsou snahy studentů o ovlivnění detekce. Studenti se při úmyslném plagiátorství mohou snažit vyhnout se její detekci například přidáváním komentářů, změnou jmen identifikátorů a změnou pořadí některých použitých příkazů, pokud tato změna nezmění chování programu. Před samotnou detekcí je tudíž nutné tomuto zamezit. Při použití abstraktních syntaktických stromů je zamezení některých snah jednoduché. V řešení stačí odstranit všechny uzly přestavující komentáře a změnit názvy použitých identifikátorů. Změna pořadí zůstává problémem, který není jednoduché vyřešit. Při použití metod, které srovnávají řetězce jsou tyto operace složitější. Tyto metody jsou často použity v systémech, které podporují větší počet jazyků a kvůli tomu musí být schopné detekovat a odstraňovat nebo přeskakovat větší počet konstrukcí reprezentujících komentáře.

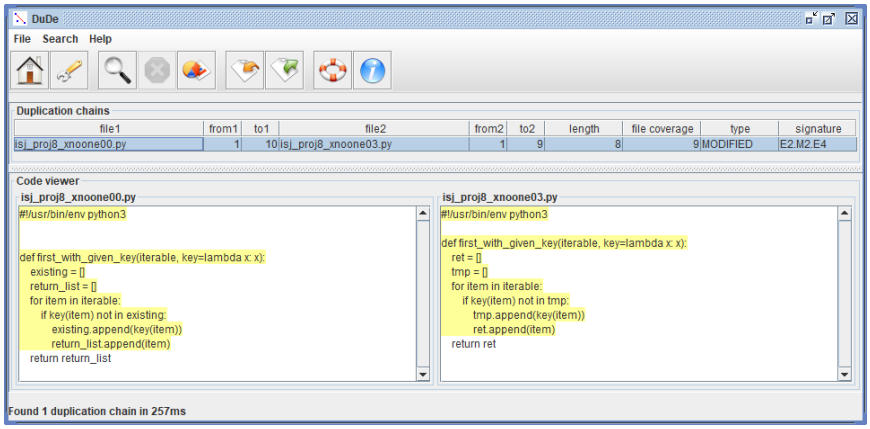
Existující řešení

Pro detekci duplikátů existují populární řešení, která jsou v detekci duplikátů efektivní. V této sekci jsou představeny dvě řešení a to programy Moss a Dude. Porovnání těchto vybraných i dalších řešení možno nalézt v [15].

- Moss – Measure Of Software Similarity[6] je nástroj pro detekci duplikátů vyvíjený Stanfordskou univerzitou od roku 1994. Moss nativně podporuje velké množství programovacích jazyků, mezi tyto jazyky patří například C, C++, Java, C#, Python a Visual Basic. Moss pro svoji funkci využívá výše zmíněnou metodu Winnowing. Při používání nástroje jsou porovnávaná řešení odesílána na server Stanfordské univerzity, kde jsou řešení uložena typicky po dobu 14 dní. Odeslaná data jsou veřejně dostupná všem, kteří mají URL vyhodnocení z důvodu anonymity obsahuje URL náhodné číslo, aby nebylo jednoduché odhadnout další existující řešení. Po odeslání kódů ke kontrole je vygenerována stránka obsahující výpis párů programů s vyznačenými společnými částmi. Systém umožňuje vynechat části, u kterých se předpokládá, že budou sdíleny. Vývojáři prosí uživatele o omezení počtu odeslaných řešení na 100 řešení denně. Kvůli zmíněným omezením může být použití nástroje pro kontrolu velkého počtu řešení nepraktické.

- DuDe – nástroj vyvinutý panem Richardem Wettelem. Jedná se o další nástroj založený na porovnávání řetězců, stejně jako předchozí nástroj. I přes to, že byl původně vyvinut pro detekci duplikátů v rámci jednoho projektu je velmi oblíbený i pro detekci ve studentských řešeních. Jeho výhodou je, že při své činnosti není limitovaný na použitý jazyk. Na rozdíl od Moss je možno DuDe použít pro jakýkoli programovací jazyk. DuDe je distribuovaný jako samostatná aplikace, díky tomuto je možné nástroj používat nezávisle na cizích serverech a není zapotřebí se starat o dostupnost vyhodnocených souborů nebo omezení počtu testovaných souborů. Více o nástroji DuDe [18].

[6] Domovská stránka projektu https://theory.stanford.edu/~aiken/moss/



Obrázek 2.2: Příklad detekovaných duplikátů pomocí nástroje DuDe.

Na obrázku 2.2 je zobrazen příklad detekce duplikátů nástrojem DuDe, Podobné části kódu jsou zvýrazněny žlutou barvou v obou řešeních.

Žádná zmíněná automatická technika však nezaručuje jistou detekci duplikátů. Automatická detekce ale může zmenšit sadu řešení, kterou je třeba kontrolovat ručně. Na samotnou detekci je lepší nahlížet jako na zjišťování podobnosti mezi řešeními. Pokud jsou si řešení příliš podobná je potřeba je ručně porovnat a podle toho rozhodnout, zdali se doopravdy jedná o duplikáty.

2.7 Souhrn

V této sekci byly představeny třídy a příklady existujících řešení systémů použitých ke zvýšení efektivity výuky programovacích jazyků. Tyto přístupy byly také popsány a byla popsána jejich efektivita. Dále se sekce zabývala možným plagiátorstvím ve studentských řešeních, způsoby, kterými je možné automatizovaně možné plagiáty upozorňovat, problémy, které se týkají procesů této detekce stejně jako problémy detekce plagiátů jako takové. Také byla představena řešení detekce plagiátů a limitace související s jejich použitím. Kvůli nutnosti zajistit bezpečnost při spouštění studentských kódů byly představeny metody, které se snaží eliminovat rizika s tím spojená.

Kapitola 3

Návrh

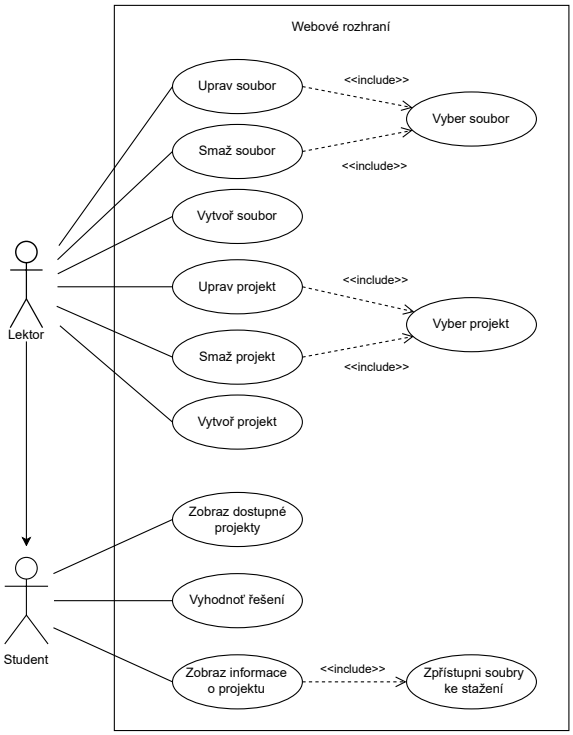
Tato kapitola se zabývá návrhem tohoto systému. Popisuje rozdělení systému do logických celků, které spolu komunikují. Systém podporuje dva způsoby použití. Hlavní způsob, který slouží k poskytování zpětné vazby a vedlejší způsob, který slouží lektorovi k hromadnému vyhodnocení sady projektů studentů. Rozhraní pro použití studenty je díky velkému rozšíření rozšíření internetu a jednoduchosti použití webových stránek navržen jako webová aplikace. Pro vyhodnocení sady studentských řešení a detekce duplikátů je navrženo rozhraní v příkazové řádce. Navržený systém je možné použít k poskytování přístupů Prompting-Answer Strategies a Giving-Answer Strategies popsaných v sekci 2.1.

3.1 Analýza a specifikace požadavků

Jak už bylo zmíněno systém je navržen pro použití dvěma uživateli.

Prvním uživatelem je student, kterého lze v aplikaci považovat za klasického uživatele. Jeho hlavním cílem je systém použít pro získání zpětné vazby a bodového ohodnocení. K splnění tohoto cíle student potřebuje zjistit jaký projekt má řešit například podle jeho dostupnosti, co je náplní tohoto projektu (dále pouze popis) a soubory, které si může pro řešení tohoto projektu stáhnout ze systému. Tyto soubory mohou být například šablony pro řešení, ukázky řešení podobných problémů nebo plný text zadání a další.

Druhým uživatelem je lektor, který má na starost zadávání projektů a jejich vyhodnocení, lze ho tedy z pohledu systému považovat za administrátora se zvýšenými pravomocemi. Systém musí tomuto lektorovi zpřístupnit nástroje, kterými bude možné poskytnout studentovi zdroje popsané výše a upravovat zpětnou vazbu a soubory sloužící k vyhodnocení odevzdaných řešení. Nastavení dostupnosti projektu je možné provést použitím formuláře webového rozhraní. Úpravu popisu, souborů ke stažení, zpětné vazby a testů je možné provést dvěma hlavními způsoby. Prvním způsobem je poskytnout lektorovi možnost soubory přes webové rozhraní nahrávat a mazat, druhým soubory v rozhraní vytvářet, editovat a mazat. První řešení, v případě, že lektor má přístup k souborům aplikace, nijak zlepšuje situaci. Tyto i další akce může se soubory provádět z pohodlí svého oblíbeného editoru. Druhé řešení lektorovi poskytuje možnost použít editor přímo v webovém rozhraní. Nemusí tedy pro malé změny nebo vytváření jednoduchých souborů spoléhat na to, že má přístup ke svému oblíbenému editoru a v případě, že ho má mu při malých změnách může ušetřit čas. Při navrhování jsem se, nejen kvůli těmto skutečnostem, rozhodl použít druhý přístup. Lektor systém také používá k jinému účelu a to k vyhodnocování většího počtu studentských řešení. Typickým případem tohoto použití může být vyhodnocení projektů všech studentů



Obrázek 3.1: Případy užití webového rozhraní.

po termínu odevzdání projektu. Při tomto použití může být pro lektora důležité zjistit zdali si nejsou některá řešení příliš podobná. Při velké podobnosti by tyto řešení mohly být plagiáty. Systém by mohl provádět vyhodnocení automaticky s koncem dostupnosti projektu, pokud by bylo zajištěno, že všichni studenti systém využívají a poslední odeslaný soubor reprezentuje jejich finální řešení. Tato skutečnost při vytváření tohoto systému nebyla zajištěna. Z tohoto důvodu bylo nutné pro toto použití vyvinout rozhraní, ve kterém by bylo možné toto vyhodnocení provést. K tomuto použití poslouží jednoduchá aplikace v příkazové řádce, která využívá některých logických prvků webové aplikace a umožňuje zadání maximální míry podobnosti v případě, že má být provedena detekce duplikátů.

Na obrázku 3.1 jsou vyobrazeny případy užití webového rozhraní odpovídající předem představené analýze.

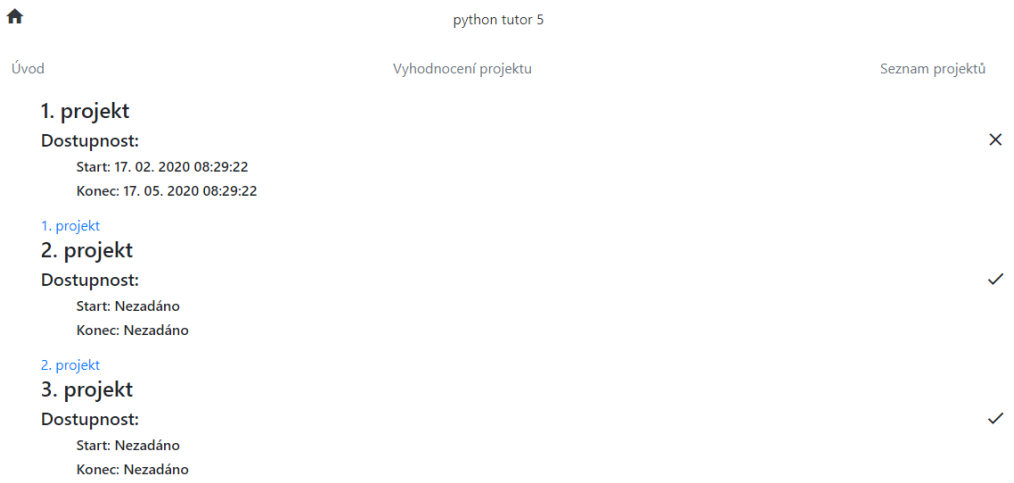
3.2 Grafický návrh webového rozhraní

Tato část se zabývá grafickým návrhem webového rozhraní. Webové rozhraní je rozdělené na administrativní a uživatelskou část a poskytuje funkce popsané v analýze a specifikaci požadavků. Rozhraní je navrženo primárně na požití v desktopových prohlížečích a to proto, že studenti budou systém pravděpodobně využívat v procesu vytváření řešení zadaného projektu a budou tak mít k dispozici prohlížeče tohoto typu.

Seznam projektů

Tato stránka slouží k zobrazení projektů a jejich dostupnosti. Obsahuje také odkaz na stránku s popisem projektu. Jednotlivé projekty jsou zobrazeny jako položky v nečíslovaného se-

znamu. Každá tato položka obsahuje data začátku a konce dostupnosti pojmenována jako start a konec. Dostupnost je pro zjednodušení také vyjádřena pomocí ikon a to ikona zaškrtnutí v případě, že je projekt dostupný nebo křížku v opačném případě.

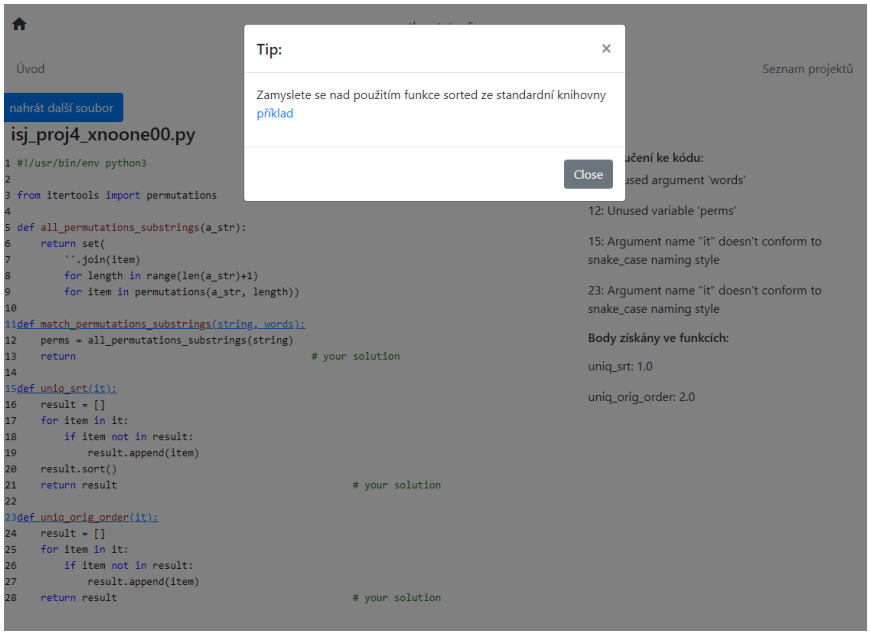


Obrázek 3.2: Způsob zobrazení seznamu projektů v systému.

Na obrazu 3.2 je zobrazen příklad tohoto seznamu. První projekt má nastavenou dostupnost a v době navštívení není dostupný. Druhý a třetí projekt nemají nastavenou dostupnost a jsou proto dostupné bez omezení.

Vyhodnocení projektu

Při navštívení této stránky se studentům zobrazí formulář s výběrem projektu, který chtějí vyhodnotit. Tento formulář se skládá z výběrového pole sloužícího pro výběr projektu a tlačítek pro nahrání souborů a pro odeslání formuláře. Po přiložení všech souborů k vyhodnocení a odeslání tohoto formuláře se zobrazí stránka která obsahuje seznam těchto souborů a hodnocení k nim. Soubory jsou zobrazeny jako nečíslovaný seznam a každý prvek se skládá z nadpisu s názvem souboru, hodnocením za soubor a obsahu souboru, ve kterém jsou vyznačeny části s dostupnou zpětnou vazbou. Po kliknutí se otevře okno s textem zpětné vazby. Toto řešení bylo vybráno z toho důvodu, že zpětná vazba může být delší text, který by při umístění do vedlejšího sloupce byl málo čitelný kvůli častým zalomení a v případě umístění pod zdrojový kód by mohl způsobit ztrátu kontextu a uživatel by se musel mezi těmito elementy často pohybovat. V menším pravém sloupci je možné zobrazit generickou zpětnou vazbu a hodnocení testovaných prvků. Hlášky tohoto typu jsou často kratší a případné zalomení nevytváří až takový problém jako v případě pokročilé zpětné vazby.



Obrázek 3.3: Vyhodnocený soubor.

Na obrázku 3.3 je zobrazen vyhodnocený soubor ve kterém byla nalezena zpětná vazba ve třech místech. Jedno okno se zpětnou vazbou je otevřeno. Z důvodu použití modálního okna je část webové stránky, která nereprezentuje pokročilou zpětnou vazbu zašedlá.

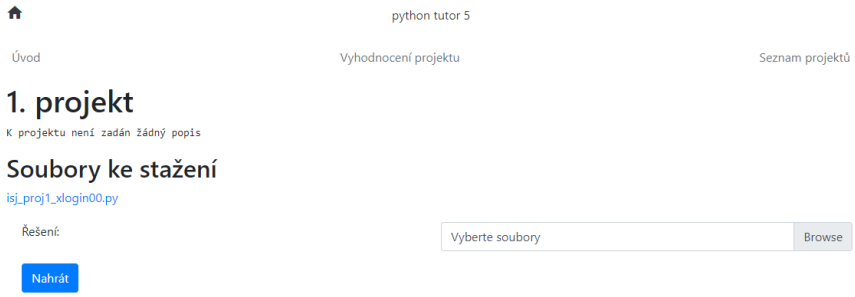
Pokud se uživatel pokusí do systému vložit soubor, který v daném projektu není hodnocen, je špatně pojmenován nebo má nepovolenou příponu je nutné uživatele na jeho nevyhodnocení upozornit a to například chybovou hláškou. Pokud uživatel odevzdá více souborů a některé z nich nejsou systémem přijaty systém zobrazí chybové hlášky pro soubory, které nepřijal u zbytku souborů zobrazí jejich vyhodnocení popsané výše. Jelikož je systém navržen k možnému odevzdání více souborů je nutné zobrazit tyto chybové hlášky výrazným způsobem. V návrhu bylo zvoleno zobrazení pomocí seznamu s těmito hláškami zvýrazněného červenou barvou.

Obrázek 3.4: Vyhodnocený soubor s chybovou hláškou.

Na obrázku 3.4 je možné vidět případ takového odevzdání. Do systému byly odevzdány dva soubory, soubor *isj_proj8_xnoone00.py* byl souborem vyhodnocen a pro soubor *isj_proj5_xnoone03.py* neexistovaly v projektu testy. Tato hláška může být způsobena například tím pokusem o odevzdání souboru jiného projektu jako v tomto případě.

Popis projektu

Tato stránka slouží k zobrazení informací o jednom projektu. Kromě jeho dostupnosti je zde možné najít popis a také stáhnout soubory, které jsou k souboru přiloženy. Pro jednoduchost použití systému tato stránka obsahuje formulář na odevzdání řešení, který je velmi podobný formuláři použitému na stránce pro vyhodnocování. Na rozdíl od něj však nemá výběrové pole pro projekt, protože je v tomto případě zbytečné.

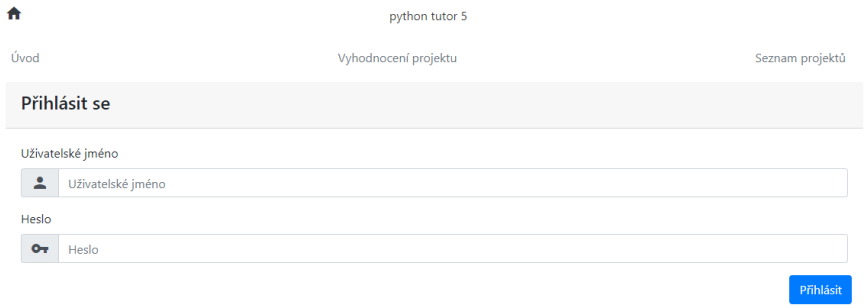


Obrázek 3.5: Detail otevřeného projektu.

Na obrázku 3.5 je zobrazen projekt s jedním souborem s názvem *isj_proj1_xlogin00.py* ke stažení a s nezadaným popisem.

Administrativní část

Pro přístup do administrativní části aplikace je potřeba zadat jméno a heslo. Pro tento účel je vytvořen jednoduchý formulář.



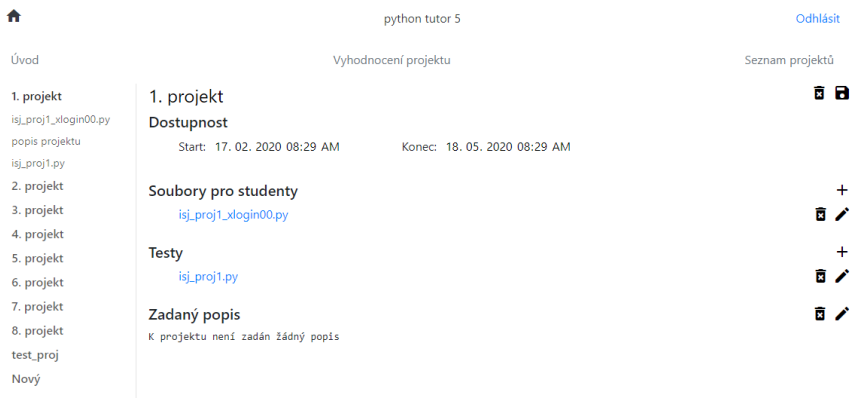
Obrázek 3.6: Přihlašovací formulář.

Formulář se zadaným uživatelským jménem a heslem. Příklad tohoto formuláře v 3.6 s nevyplněnými hodnotami. Pro zjednodušení navigace v administraci je po levé straně

zobrazena navigace s názvy jednotlivých projektů. Po kliknutí na název projektu se zobrazí popis projektu a v levé navigaci se pod položkou vybraného projektu zobrazí názvy souborů, které vybraný projekt obsahuje. Po kliknutí na některou z těchto položek se zobrazí její administrace. Poslední prvek navigace slouží pro vytváření nových projektů. Vybraný projekt je zvýrazněna tučným písmem. Pokud je vybrán některý ze souborů je tučně zvýrazněn i jeho název. Příklad navigace je zobrazen v obrázku 3.7.

Administrace projektu

Na této stránce se nachází název, dostupnost projektu, seznam souborů rozdělených do kategorií podle toho, co představují a je zde zobrazen popis projektu. Dostupnosti je možné nastavovat jako datum a čas. Pro přejmenování projektu stačí kliknout na zobrazený název. Tyto změny je třeba uložit pomocí zobrazeného tlačítka představující disketu. Vedle této diskety je umístěno tlačítko pro odstranění celého projektu a všech jeho součástí reprezentované popelnicí. Vytvoření dalšího souboru pro studenty nebo testu je možné pomocí stisknutí tlačítka pro přidání zobrazeného jako plus v odpovídající sekci. Po kliknutí na tlačítka pro přidání nebo editaci je administrátor přesměrován na administraci souborů. Soubory lze odstraňovat a editovat pomocí tlačítka pro editaci zobrazeného tužkou a odstraňovat tlačítkem stejného vzhledu jako tlačítko pro odstranění projektu. Při odstranění popisu projektu se při pokusu o jeho zobrazení použije text “K projektu není zadán žádný popis”. Tyto tlačítka jsou zobrazena vedle jména souboru. Jeho jméno zároveň funguje jako odkaz na editaci příslušného souboru.



Obrázek 3.7: Přihlašovací formulář.

Na obrázku 3.7 je příklad otevřeného projektu s nezadaným popisem obsahující jeden studentský soubor a jeden test. Datum začátku dostupnosti není nastaveno, konec projektu je nastaven na 18. května 2020 v 8:29.

Administrace souborů

Na této stránce je zobrazen název souboru, jeho obsah a tlačítka pro uložení změn a jeho odstranění. Ovládací prvky se velmi podobají ovládacím prvkům pro projekt. Tlačítka jsou zobrazena za pomocí ikon stejných ikon jako v případě uložení změn projektu a jeho odstranění a přejmenování souboru provedeno stejným způsobem. Obsah souboru je zobrazen

ve webovém editoru. Bez kliknutí na tlačítko uložit nedochází k ukládání změn a je proto možné použít tuto stránku jen pro čtení.

Obrázek 3.8: Přihlašovací formulář.

3.3 Logické prvky systému

Systém je možné rozdělit na logické prvky, které mají každý vlastní funkcí kterou zastávají a komunikují spolu. Tyto logické prvky jsou popsány níže.

Zdroj studentských kódů

Tento logický prvek je zodpovědný za dodání testovatelných souborů ze studentských řešení. Pokud by studentské řeší obsahovalo soubor, který by nebyl povolen je úkolem tohoto logického prvku ho odstranit. Pro pojmenování souborů byla vytvořena notace skládající se z názvu testu odpovídajícímu souboru a uživatelského jména studenta, je ji tedy možné zapsat jako *<název testu>_<uživatelské jméno>*. Při návrhu jsem se rozhodl přidat do systému možnost odeslat soubor anonymně. Při tomto použití je notace pouze *<název testu>*.

Vyhodnocení

Studentské soubory, které jsou testovatelné je potřeba vyhodnotit. Tento prvek se postará o výběr správných testovacích prvků a jejich spuštění. Spolu s generátorem zpětné vazby se tudíž postará o celkové vyhodnocení souborů v sadě.

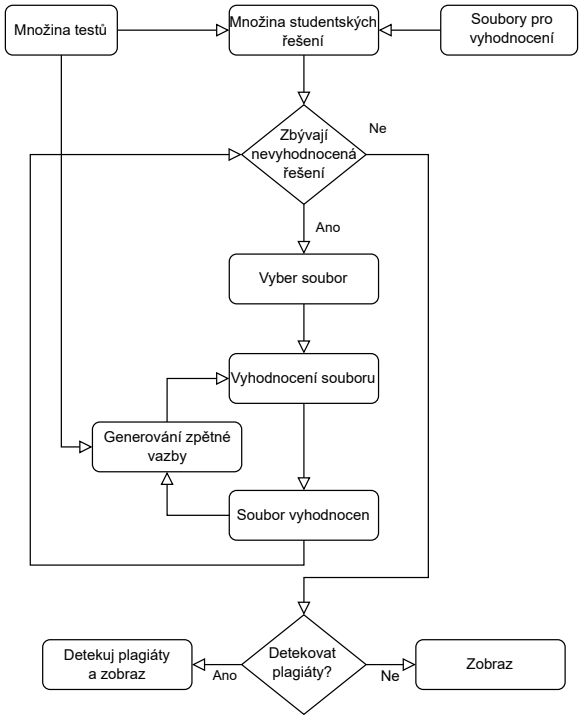
Generátor zpětné vazby

Jelikož systém patřím mezi skupinu systémů založenou na poskytování zpětné vazby je třeba tuto zpětnou vazbu vygenerovat ze studentova řešení. O toto generování se stará právě tento modul. Je možné poskytovat zpětnou vazbu dvou typů. Generickou zpětnou vazbu, kterou je možné použít při každém řešení, například doporučení, která jsou obsažena v PEP8[1] a v dalších doporučeních pro zlepšení kvality kódu v jazyce Python a doporučení specifická pro řešený projekt. Mezi takové doporučení může patřit například doporučení vytvoření množiny místo seznamu, pokud je potřeba zajistit jedinečnost elementů.

[1]Více o doporučení PEP8 https://www.python.org/dev/peps/pep-0008/

Detekce duplikátů

V systému jsem se rozhodl použít detekci duplikátů za pomocí analýzy podstromů abstraktních syntaktických stromů zmíněná v 2.5. Pro jejich detekci je potřeba minimálně dvojice studentských řešení. Pokud se provádí detekce duplikátů není zapotřebí spouštět generickou zpětnou vazbu, jelikož ji není možné použít k hodnocení.



Obrázek 3.9: Architektura navrhovaného systému.

Na obrázku 3.9 je zobrazena architektura navrhovaného systému rozložená do výše popsaných logických prvků. Logické prvky společnou komunikací vytváří celý systém. Při použití webové aplikace popsané v 3.1 není potřeba testovat plagiáty.

3.4 Shrnutí

V této části byl popsán návrh systému, jeho rozdělení do logických prvků, které komunikací spolu vytváří celý systém. U každé logické části byl popsán důvod, proč je tato část potřeba a je popsáno o jakou funkci v systému zastávají. Také bylo popsáno rozdělení systému na webovou část, která je použitá pro všechny operace, které provádí student a většinu operací se systémem prováděných lektorem. Pro lektora je také navržena aplikace v příkazové řádce pro vyhodnocování balíků projektů a detekci duplicit sloužící k detekci plagiátů. Dále bylo popsána struktura webového rozhraní. Webové rozhraní bylo rozděleno na uživatelskou a administrátorskou část, kde uživatelská část obsahuje nástroje, které studentům umožní orientovat se v zadaných projektech a administrátorská část obsahuje nástroje sloužící lektorovi k přizpůsobování částí aplikace jeho požadavkům. U webového rozhraní obou typů byl popsán jejich grafický návrh.

Kapitola 4

Implementace

V této kapitole je popsána vlastní implementace systému. Implementace je založena na rozdělení do systému do logických částí popsaných v 3.3. Systém se skládá ze tří částí. První částí je část zodpovědná za vyhodnocování samotných řešení a generování zpětné vazby. Druhá část za vyhodnocování většího počtu studentských řešení současně a detekci duplikátů mezi nimi. Třetí část systému je administrace, která poskytuje možnost konfigurace aplikace a správy aplikace. Dále jsou popsány postupy při generování zpětné vazby, obsah testovacího souboru a postup, který je možné použít pro nasazení aplikace a omezení, která při použití popsaného způsobu existují. Také je popsán formát souboru s nastavením.

4.1 Použité technologie

Při implementaci byl vybrán jazyk Python a to z důvodu jeho vysoké rozšířenosti a také z toho důvodu, že standardní implementace *CPython* obsahuje ve své standardní knihovně konstrukce pro práci s abstraktními syntaktickými stromy s názvem ast. Systém byl primárně vyvíjen pro verzi 3.6.9 a to z důvodu použití této verze na fakultním serveru *ISJ*[1], na kterém byl systém dostupný studentům. Systém byl při vývoji testován na verzích 3.7.4 a 3.5.3 aby byla zajištěna kompatibilita i s jinými verzemi jazyka. Některé moduly bohužel nejsou kompatibilní s verzí 3.8 a to z důvodu rozsáhlých změn týkajících se knihovny ast. Pro vývoj webového rozraní byl vybrán rámec Flask[2]. Flask je webový rámec (dále používaný jen anglický název framework) vyvíjený v Pythonu, je v dnešní době oblíbený hlavně díky jeho jednoduchosti a rozšiřitelnosti, Flask jako takový se zaměřuje pouze na omezenou škálu funkcí, zbylé funkce je možné ale jednoduše získat použitím rozšiřujících balíčků. Ke své funkci používá šablonovací jazyk jinja[3] a vestavěný webové rozhraní Werkzeug. Jazyk jinja poskytuje možnost vytvářet šablony a usnadňuje tak vyvíjení dynamických webových aplikacích. Vestavěný server není doporučováno využívat pro produkční použití, proto je v této práci nahrazen produkčním server waitress[4]. Mezi výhody tohoto serveru patří jeho dostupnost na většině platforem například Linux, Microsoft Windows i Mac OS. Díky použití frameworku flask je možné vyvíjet aplikaci přímo v jazyce Python a není proto potřeba spoléhat na ostatní programovací jazyky pro tvorbu webových aplikací jako například PHP. Pro tvorbu webového rozhraní byl vybrán framework Bootstrap[5]. Bootstrap

[1] Dostupné na https://isj.fit.vutbr.cz/
[2] Dokumentace frameworku Flask https://flask.palletsprojects.com/en/1.1.x/
[3] Dokumentace šablonovacího jazyka jinja https://jinja.palletsprojects.com/en/2.11.x/
[4] Dostupné z https://docs.pylonsproject.org/projects/waitress/en/latest/
[5] K nalezení zde https://getbootstrap.com/

je volně šiřitelný framework pro vytváření prezenční vrstvy webových aplikací (front-end framework) zaměřený na zjednodušení vývoje responzivních webových aplikací. Bootstrap je oblíbený taky díky svojí kompatibilitě s moderními prohlížeči, ale také schopnosti přizpůsobit se starším prohlížečům. Pro svoji funkci využívá jazyků HTML CSS a javascript. Poskytuje řadu předdefinovaných tříd a přizpůsobuje také vzhled standardních elementů HTML, v projektu je použita verze 4. Pro vytvoření editoru pro administraci byl vybrán Ace editor[6]. Jedná se o zabudovatelný webový editor navržený pro úpravu souborů a je vyvinut především pro použití pro soubory obsahující kód. Poskytuje proto velké množství funkcí s tím spojených, jako například zvýrazňování syntaxe a páru závorek, skládání částí kódu a další. Generování generické zpětné vazby bylo rozšířeno o tipy z nástroje Pylint[7]. Jedná se o nástroj pro statickou analýzu zdrojového kódu v jazyce Python. Kromě detekce chyb poskytuje možnost generovat doporučení ke kódu v případě použití konstrukcí, které mohou být nebezpečné nebo mohou způsobovat vedlejší, potencionálně nechtěné, vedlejší účinky. Také poskytuje možnost kontrolovat doporučení popsaná v PEP8.

4.2 Struktura aplikace

V této sekci je popsána struktura implementované aplikace založená na jejím návrhu v sekci 3.

Třída FileUploader

Tato třída je zodpovědná za ukládání studentských řešení odeslaných přes webové rozhraní na lokální úložiště serveru. Při ukládání je pro studentské řešení vytvořena podložka složky zadané v nastavení aplikace. Tato složka je pojmenována unikátním náhodným řetězcem vygenerovaným pomocí funkce `uuid4`, která slouží k vygenerování náhodného `uuid`[8] a je obsažena ve standardní knihovně jazyka. Posléze jsou seznam souborů studenta procházen a jsou z nich odstraněny soubory, pro které v systému nejsou testy. Odstraněné soubory nejsou ukládány na disk a uživatel je o jejich odstranění informován chybovou hláškou. Díky tomuto není zapotřebí jména souborů kontrolovat. Odkaz na takto připravenou složku je předán dalšímu modulu.

Reprezentace souborů

V systému jsou soubory reprezentovány za pomocí 3 tříd a to *File*, *StudentFile*, *TestFile*. Třída *File* slouží jako bázová třída, při vytváření instance se třída pokusí sestavit ze souboru abstraktní syntaktický strom, který je v případě úspěchu uložen. Dále třída zapouzdřuje původní obsah souboru a základní operace nad souborem. Třída *StudentFile* je potomkem třídy *File* a kromě už zmíněné funkcionality poskytuje možnost ukládání hodnoceni a zpětně vazby. Třída *TestFile* rozšiřuje funkcionalitu třídy *File* o získávání limitů zdrojů, které mají být použity, povolených knihoven, informace, zdali má být v řešení povoleno používat další definované funkce a seznam na funkce poskytující postupnou zpětnou vazbu.

[6] K nalezení zde https://ace.c9.io/
[7] K nalezení zde http://pylint.pycqa.org/en/latest/
[8] Více na https://docs.python.org/3.8/library/uuid.html#uuid.uuid4

Třída TestRunner

Tento modul je zodpovědný za spouštění testů, ukládání hodnocení a spouštění modulu pro zpětnou vazbu, linkování řešení a testů a samotné vyhodnocování.

Modul nejdříve vytvoří ze všech souborů ve složce interní reprezentaci studentského řešení *StudentFile* a z testů reprezentaci *TestFile*. Na začátku vyhodnocování každého souboru je ve vedlejším vlákně spuštěn modul pro generování zpětné vazby. Poté jsou procházeny všechny testovatelné prvky (funkce, metody, třídy) a jsou k nim vyhledávány testy v testovacích souborech. Při nalezení takového souboru je abstraktní syntaktický strom testovatelného prvku a reprezentace testu předána instanci třídy *Linker*, jeho výstup je vyhodnocen za pomocí modulu k tomu určenému a to třídou *Evaluator*. Pokud testovací soubor obsahuje omezení zdrojů, která mají být použita jsou také předána. Výstupem třídy *Evaluator* je bodové ohodnocení testované části a zpětná vazba získávaná za běhu (například rozdíl v očekávaném výstupu a výstupu studentského řešení). Po vyhodnocení všech testovaných prvků je nutno počkat na dokončení činnosti modulu generujícího zpětnou vazbu. Pokud prvek není mezi testovanými prvky je jeho vyhodnocování přeskočeno. Po vyhodnocení všech testovaných prvků je cyklus opakován pro další soubor. V systému byla kvůli detekci vstupního bodu (entry point) zavedena notace odpovídající `*<název testovaného prvku>*`, kde * označuje jakoukoli sekvenci znaků.

Třída Checker

Tato třída je zodpovědná za generování zpětné vazby. K tomuto účelu používá programového rozhraní modulu Pylint pro zpětnou vazbu genetického typu a seznam funkcí pro postupnou zpětnou vazbu ze souboru reprezentující test.

Třída Linker

Tato třída se stará o správné spojení abstraktního syntaktického stromu testovaného prvku, použitých knihoven, pokud jsou povoleny a testovacích prvků. Výstupem je syntaktický strom, který reprezentuje spustitelný modul. Pokud jsou v souboru nalezeno použití nepovolených knihoven *Linker* nepřidá odpovídající uzel abstraktního stromu do výsledného stromu. Díky tomuto je možné vyhodnotit ta řešení, která sice obsahují importování těchto knihoven, ale jsou nepoužity. *Linker* nepodporuje spojování částí kódu, které jsou obsaženy na globální úrovni modulu, kromě importování knihoven. K tomuto rozhodnutí vedla struktura zavedená v dosavadních projektech předmětu ISJ a také fakt, že použití konstrukcí na globální úrovni není v mnoha případech správnou praktikou. Při analýze všech projektů nebyl důvod pro použití příkazů z globální úrovně důvod. Některá řešení obsahovala použití proměnných definovaných na globální úrovni jako implicitní hodnoty funkcí. *Linker* se musí postarat o jejich odstranění. Pokud by jejich použití neodstranil nebylo by možné vyhodnotit části řešení, která by tyto proměnné využívala.

Tento přístup poskytuje velkou možnost kontroly vlastností programů, které se vyhodnocují. Ostatní řešení, jako například použití frameworků k jednotkovému (unit) testování sebou nepřináší takovéto výhody. Jednotkové testy jsou výborným nástrojem při vývoji aplikací pro použití v tomto případě ale postrádají mnohé funkce spojené se spouštením kódu ve kterém mohu být nechtěné nebo nepovolené konstrukce. Pokud by se projekt při nalezení těchto konstrukcí pouze nevyhodnotil bylo by do systému možné odevzdat soubor, který by zadaný problém řešil i bez použití zakázaných konstrukcí, ale systém by ho kvůli jejich nalezení nevyhodnotil. Z tohoto důvodu byl vybrán tento přístup, který se snaží sestavit

spustitelný soubor odpovídající omezením a jeho vyhodnocení selže v případě, že řešení závisí na nepovolených konstrukcích. Pokud například testované řešení bude obsahovat dvě testované funkce a jedna z nich bude používat zakázanou knihovnu bude možné vyhodnotit body získané v této funkci.

Třída Evaluator

Tato třída je zodpovědná za vyhodnocování sestavených syntaktických stromů a omezení zdrojů. Omezení zdrojů je provedeno nastavením limitů vybraných limitů. Tyto limity jsou přístupné z balíku resource[9], který využívá vestavěných funkcí jádra operačního systému a jsou dostupné ve všech operačních systémech s jádrem Linux. Každý limit je reprezentovaný dvojicí hodnot. Tyto hodnoty reprezentují takzvané měkké (soft) a tvrdé (hard) limity. Měkké limity proces nemůže překročit tuto hodnotu, ale může ji zvyšovat maximálně na hodnotu tvrdého limitů. Tvrdý limit může proces snižovat, jeho zvýšení není ale povoleno bez přístupových práv super-uživatele (super-user), typicky root. Tvrdý limit se tedy dá považovat za maximum zdrojů, které jsou povoleny při řešení. Mezi podporované limity patří limit `RLIMIT_AS`, který reprezentuje maximální alokovatelný prostor aplikací a limit `RLIMIT_CPU`, který reprezentuje maximální procesorový čas.

Tyto limity bohužel nejsou dostupné na všech platformách. Na platformách, které limity nepodporují je jejich nastavení přeskočeno, aby byla zachována možnost vyhodnocovat soubory i na těchto platformách.

Generování pokročilé zpětné vazby

Pokročilá zpětná vazba je generována za pomocí tříd rozšiřujících třídu `NodeVisitor` z balíku ast standardní knihovny. Tyto třídy fungují tak, že pomocí zděděné metody visit navštěvují všechny podstromy stromu předaného argumentem. Pro každý podstrom je hledána specializovaná metody podle schématu `visit_` a název třídy, která reprezentuje konstrukci, která má být navštívena. Například metoda `visit_FunctionDef` bude použita pro navštívení uzlu představujícímu definici nové funkce. V projektu je pro upřesnění struktury těchto tříd vytvořena abstraktní třída `AbstraktNodeVisitor`, která garantuje, že výstupem tříd, které implementují tuto abstraktní třídu je slovník referencí na podstromy, které odpovídají vyhledávacímu kritériu. Tento seznam je společně s dalšími informacemi použit při vytváření instancí třídy Tip. Tato třída je použita jako rodičovská v případě, že implementovaná třída provádí obecnou detekci nějakého prvku. Pro detekci konkrétních konstrukcí je možné použít jako rodičovskou abstraktní třídu `AbstractTipVisitor`, která je specializovaná na jeden detekovatelný příklad a obsahuje text zpětné vazby a odkazy na literaturu a je tedy možné je použít přímo pro vytvoření instancí třídy Tip.

Systém také podporuje bodové ohodnocení souboru na základě zpětné vazby a postupnou zpětnou vazbu. Postupná zpětná vazba je možnost poskytovat studentovi zpětnou vazbu po částech, toto může být praktické v případě, že v projektu by mohlo být této zpětné vazby velké množství, což by mohlo studenta odradit. Pokud je tato možnost použita student má po prvním odevzdání přístup k první části, která je označena jako koncová. Pokud ji v dalším odevzdání využil a upravil podle ní její řešení je mu poskytnuta další část. Může se stát, že při pokusu upravit svoje řešení využil konstrukce, před kterými ho varovala předchozí zpětná vazba. V takovémto případě je mu tato vazba znovu zobrazena. Bodové ohodnocení je založeno na stejném principu jako zpětná vazba. Použije se potomek abstraktní třídy

[9] Více na https://docs.python.org/3/library/resource.html

`AbstractNodeVisitor`, který reprezentuje hodnocenou konstrukci a při jeho nalezení nebo nenalezení (podle toho, zdali se má v kódu nacházet) je upraveno ohodnocení souboru.

Třída Tip

Tato třída je použita jako kontejner pro doporučenou zpětnou vazbu a to pro informace o tom kde se nachází, textovém popisu pro studenta a odkazy na případnou doporučenou studijní literaturu. Vlastnosti této třídy jsou po detekci uloženy do instance třídy *Checker*. Po vyhodnocení všech souborů jsou tyto typy kódovány v jazyce JSON a připojeny k odpovědi od serveru. Tyto data jsou posléze použita k vyplnění modálního okna.

Detekce plagiátů

Při detekci jsou použity již předem popsané třídy kromě části třídy *Checker*. Nevyhodnocují se generické tipy nástrojem Pylint a zpětná vazba, která není použita k hodnocení. Před spuštěním vlastní zpětné vazby jsou v kódu nahrazeny názvy identifikátorů, aby byla snížena možnost ovlivnit detekci popsaná v 2.6. Systém provádí detekci pomocí detekce podstromů také popsané v 2.6. Toto řešení bylo vybráno hlavně z důvodu použití abstraktních syntaktických stromů, které jsou aplikací již sestaveny. Pro každý podstrom stromu je z klíčových prvků tohoto podstromu vypočítána hodnota otisku. Všechny odevzdané soubory jsou poté procházeny a porovnávány s ostatními, během tohoto procesu je počítána podobnost těchto dvou řešení. Pokud je tato podobnost vyšší než uživatelem definovaný práh podobnosti je soubor označen jako možný plagiát.

4.3 Popis testovacího souboru

Testovací soubor kromě vlastních testovacích funkcí obsahuje další položky, kterými je možné upravit chování vyhodnocovacího skriptu. Proměnné `cpu_limit` a `max_memory` slouží k nastavování limitů popsaných v 4.2, proměnné `check_imports` a `imports` slouží k omezení knihoven. Pokud proměnná `check_imports` obsahuje hodnotu, která je při konverzi na pravdivostní hodnotu vyhodnocená jako pravdivá jsou importované moduly omezené na moduly uvedené v seznamu `imports`. Seznam těchto knihoven je pro zjednodušení zadávání možné uvést v podobě řetězce. Pokud některá z proměnných není v souboru definována jsou použity implicitní hodnoty limitů a předpokládá se, že není třeba v řešení limitovat použití knihoven. Pro postupnou zpětnou vazbu je možné definovat funkce s jakýmkoli názvem, jediným omezením je skutečnost, že tyto funkce musí jako návratovou hodnotu používat instance třídy Tip. Reference na tyto funkce je poté nutné uložit do proměnné s názvem `feedback_stages` a to v pořadí, ve kterém se mají vyhodnocovat. K hodnocení stylu použitého kódu slouží funkce s názvem `rate_code_style`, její návratovou hodnotu je zisk nebo ztráta bodů. Všem zmíněným funkcím je předán abstraktní syntaktický strom odpovídající celému řešení.

Testovací funkce musí ve svém názvu obsahovat název funkce, pro kterou je určena a jako parametr je jí předána reference na testovaný objekt. Toto pojmenování bylo vybráno z důvodu jednoduchosti vytvoření testovacích funkcí. Dále by bylo možné detekovat použité funkce uvnitř testovací funkce, toto řešení jsem se rozhodl nepoužít a to hlavně z důvodu možných kolizí definovaných funkcí v testovacím a testovaném souboru. Předání reference na testovaný objekt je nutné z důvodu použitého řešení. Při ručním spojování dvou abstraktních stromů docházelo k nenalezení testované funkce. Systém očekává, že ná-

vratovou hodnotou funkce bude buď počet získaných bodů v této testovací funkci nebo dvojici, kde první prvek označuje počet získaných bodů stejně jako v předchozím případě a druhý prvek obsahuje dvojici název hodnocené funkce a text tipu. Tato zanořená dvojice je použita pro vytvoření instance třídy Tip. V současné době v těchto datech není možné předat odkaz na literaturu, ale vzhledem k tomu, že jediným využitím je předání informace o odlišnosti v návratové hodnotě testované funkce nebo odlišnosti dat zapsaných na standardní výstup, není tato limitace závažná. Původně jsem zamýšlel použít pro vyhodnocování testů knihovnu unittest[10], nebo knihovnu jí podobnou, toto řešení nebylo kvůli rozhodnutí vyhodnocovat stromy přímo možné.

4.4 Webové rozhraní

Webové rozhraní je možno rozdělit na dvě části a to na uživatelskou a administrativní část. Obě tyto části jsou reprezentovány vlastním řadičem (controller), který využívá výše popsané aplikace jako modelu, kterému konvertuje vstup od uživatele na příkazy. Pro zobrazení, pohled (view), využívá šablony (template) vytvořené pomocí nástroje jinja2 popsaného v 4.1. Seznam cest (route) systému jsou i s použitou metodou jsou obsaženy v souboru `server.py`, který představuje vlastní webové rozhraní vytvořením všech potřebných řadičů.

Uživatelská část

Uživatelská část je vytvořena pro studenty, kterým umožňuje primárně odeslat soubory pro vyhodnocení systémem ale také seznam dostupných projektů, jejich zadání a případné soubory ke stažení (například šablony pro řešení). Po odevzdání souborů pro vyhodnocení je studentovi zobrazena webová stránka, na které se nachází jeho řešení, ve kterém jsou vyznačeny části s pokročilou zpětnou vazbou. O zvýraznění syntaxe se stará framework highlight.js[11], který je založený na jazyce JavaScript a je ho tudíž možné kombinovat jeho použití s ostatními balíky. Po kliknutí na takto zvýrazněnou část je otevřeno modální okno s textem zpětné vazby. Tato stránka dále zobrazuje zpětnou vazbu, výstup z modulu Pylint a hodnocení funkcí pro které existují testy a bylo v nich získáno bodové ohodnocení. V případě, že nebylo možné z některého ze souborů sestavit abstraktní syntaktický strom výsledek pro tento neobsahuje zpětnou vazbu ani ohodnocení ale pouze výstup interpretu s chybovou hláškou a obsah odevzdaného souboru.

Administrátorská část

Administrátorská část je vytvořena pro vyučujícího předmětu a umožňuje upravování, přidávání a mazání existujících projektů. Pro přístup do administrace je od uživatele vyžadováno přihlášení. Informace o přihlášení jsou uložena do proměnné SESSION, která je obsažena v rozšíření s názvem Flask-session[12]. Přihlašovací kombinace jména a hesla je uložena v souboru s nastavením webového serveru. Mezi možnosti úpravy patří změna jejich dostupnosti, změna jejich zadaného popisu a administrace testů tohoto projektu. Pro upravování souborů k projektu je použit framework ace editor popsaný v 4.1. Zatímco úpravy dostupnosti projektů se v aplikaci projeví okamžitě úprava testů nebo administrace projektů vyžaduje restartování serveru. Za tímto účelem je použitý balíček hupper[13], který

[10] Více na https://docs.python.org/3.8/library/unittest.html
[11] Dostupný na https://highlightjs.org/
[12] Dostupné https://github.com/fengsp/flask-session
[13] Dostupný z https://pypi.org/project/hupper/

umožňuje, při spuštění aplikace s parametrem –restart, automaticky restartovat systém při těchto změnách nebo při změně některé z jeho částí. Popis nebo zadání projektu může být zadáno jak ve formátu HTML nebo pouze jako text. Pokud je popis zadaný jako text formátování popisu odpovídá formátování použitému v souboru.

4.5 Aplikace v příkazové řádce

Jak už bylo zmíněno tato aplikace byla vyvinuta za účelem vyhodnocení řešení více studentů zároveň a detekci duplikátů. Typickým případem použití je vyhodnocení souborů studentů, kteří odevzdali řešený projekt do konce zadaného termínu. Aplikace poskytuje dvě možností výstupu vyhodnocení, obě možnosti jsou formátu csv. Prvním je zobrazení dvojic uživatelské jméno a hodnocení, druhá možnost je vygenerování souboru s hodnocením na základě dodaného souboru ze systému wis. Při detekci duplikátů je možné zadat práh podobnosti a výsledek detekce je zobrazen ve formátu `<název souboru>: <názvy souborů s podobnosti vyšší než práh>`.

4.6 Nasazení aplikace

K nasazení (deployment) aplikace je vytvořen instalační skript `install.sh`, který zajistí instalaci potřebných balíků uvedených v souboru `requirements.txt`, ke své funkci vyžaduje stejně jako systém interpret jazyka python a také balíčky venv pro vytvoření virtuálního prostředí, ve kterém bude systém spouštěn a balíček pip, který je použity k instalaci potřebných balíčků. Oba tyto balíčky jsou obsaženy ve standardní instalaci jazyka python minimální verze. Pro zajištění běhu aplikace je použita služba démona systemd[14]. Tato služba zajišťuje spuštění služby při restartování systému, na kterém je aplikace nasazena, její restartování v případě jejího pádu a jednoduché ruční restartování. Další důležitou vlastností démona systemd je možnost využití démona journald, který je jeho součástí a poskytuje možnost žurnálování (logging). Je tak možné automaticky zapisovat informace ze systému a narozdíl od ukládání těchto dat do souboru poskytuje možnost filtrovat tyto záznamy. Démon systemd je bohužel oficiálně podporovaný pouze na operačních systémech s jádrem linux.

4.7 Konfigurace

Konfigurace systému je uložena v souboru `settings.json` o který se stará třída *SettingsManager*. Jako formát pro tento soubor jsem zvolil velice rozšířený formát JSON a to kvůli jeho rozšířenosti, čitelnosti a jednoduchosti práce s ním. Soubor obsahuje informace o projektech a to jejich názvy a dostupnost, složku, do které mají být ukládána studentská řešení a informace o tom, zdali se mají jimi nahrané soubory po jejich vyhodnocení odstraňovat a odkaz na konfigurační soubor balíku Pylint. Dále jsou je možné nastavit složky, které obsahují testovací soubory a soubory, které si mohou studenti stáhnout.

Konfiguraci jednotlivých projektů obsahují testovací soubory. Každý projekt je reprezentován složkou, ve které jsou umístěny jednotlivé soubory s testy. V každém testovacím souboru je tudíž možné nastavit povolené knihovny a limity pro každý soubor řešení. Pokud testovací soubor neobsahuje jejich nastavení jsou v případě limitů použity implicitní hodnoty a použití knihoven není žádným způsobem omezeno.

[14] Manuálové stránky http://man7.org/linux/man-pages/man1/systemd.1.html

4.8 Shrnutí

V této sekci byly popsány nástroje použité k implementaci systému, implementace logických prvků představených v návrhu a dalších důležitých tříd, které umožňují použití tohoto rozdělení. Také byly popsány důvody, které vedly k výběru vyhodnocování abstraktních syntaktických stromů sestavených ze studentského řešení a testů k nim. Výběr tohoto řešení poskytuje velkou kontrolu nad kódem, který bude spuštěn bez potřeby zásahu do zdrojového kódu souboru. Také byly popsány způsoby použité k nasazení aplikace na velké škále operačních systémů typicky použitých pro hostování webových aplikací a limitace těchto řešení na ostatních platformách. Dále byla popsána struktura souboru s nastavením a struktura testovacích souborů a možnosti, které systém poskytuje.

Kapitola 5

Testování

Systém byl aktivně použit při výuce předmětu Skriptovací jazyky (ISJ) v letním semestru tohoto školního roku. Většina zpětné vazby byla založena na strategii Prompting-Answer, kdy studentům byly poskytnuty podněty, které mohli využít ke zlepšení, například název konstrukce, kterou mají použít a odkaz na dokumentaci, který obsahuje zdůvodnění jejího použití. Strategie Giving-Answer byla použita limitovaně a to spíše pouze v případě, že se výstup od studenta lišil od požadovaného výstupu. V systému většinou nebyla použita možnost postupné zpětné vazby a to z důvody jednoduchosti zpětné vazby v zadaných projektech. Pro některé projekty nebylo možné pokročilou zpětnou vazbu vytvořit vůbec, tyto projekty byly velmi jednoduché jejich úkolem bylo například doplnit jeden řádek řešení a zpětná vazba by tedy řešení přímo odhalila. Studenti měli možnost používat systém jak k otestování jejich řešení tak i k získávání zpětné vazby. Díky tomuto nasazení bylo možné systém ladit na velkém počtu různorodých řešení.

5.1 Automatizované testy

Projekt obsahuje automatizované testy systému. Tyto testy jsou zaměřené na moduly použité k generování zpětné vazby a detekci duplikátů. Tyto testy využívají framework pytest[1] a jsou obsaženy ve složce `SystemTest\Pytest`.

5.2 Ruční testování aplikace

Toto testování bylo zaměřeno na testování zobrazování webového rozhraní v různých prohlížečích a jeho zachování v případě chybových stavů. Kontrola zobrazování byla provedena v prohlížečích Google Chrome, Microsoft Edge, Mozilla Firefox a Opera. Díky použití frameworku bootstrap byla webová stránka kompatibilní se všemi z těchto prohlížeči a v případě prohlížeče Google Chrome i na mobilní platformě, testováno na platformě Android. I přes to, že s použitím aplikace pro na mobilní platformě nebylo počítáno je toto použití možné. V části tohoto testování bylo použití studenty velmi nápomocné. Pokud byla v systému studenty objevena chyba byla jimi rychle ohlášena a opravena.

[1] Dostupný z https://pypi.org/project/pytest/

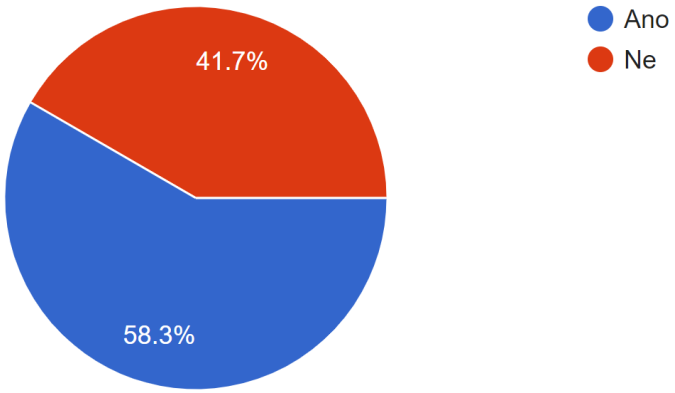
5.3 Analýza odevzdaných řešení

Z analýzy odevzdaných souborů vyplývá, že většina studentů používala systém pouze ke kontrole bodového hodnocení jejich řešení, menší skupina použila alespoň část zpětné vazby. Někteří studenti se rozhodli systém nevyužít vůbec. U těchto řešení byl častý problém s syntaktickou validitou nebo s nedodržením pojmenování souborů, funkcí nebo tříd. Špatné pojmenování souborů se nejčastěji objevovalo v prvním projektu. Všechny tři zmíněné chyby nemohou při použití systému získat žádné bodové ohodnocení, z nevalidních je souborů není možné sestavit abstraktní syntaktický strom, který je použit k vlastnímu vyhodnocení a v případě špatného pojmenování není možné najít odpovídající testy.

5.4 Dotazník

Vzhledem k nasazení systému v období zadání prvního projektu bylo možné požádat na konci semestru studenty o zpětnou vazbu k systému. Na dotazník se rozhodlo odpovědět 12 studentů, což odpovídá 13.3 % ze studentů, kteří odevzdali poslední projekt předmětu. V dotazníku byly obsaženy následující otázky.

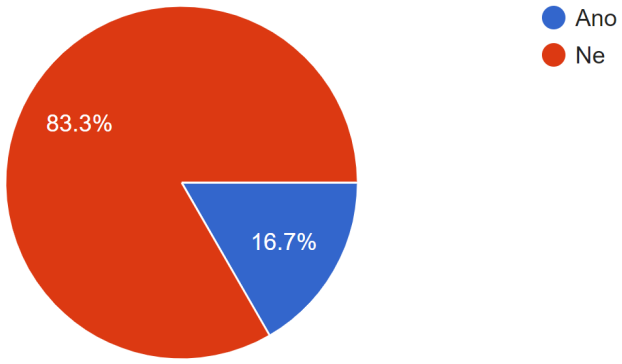
Upravili jste byste svůj kód, když by systém indikoval, že získáte plný počet bodů, ale Vaše řešení není optimální, bylo by možné použít vhodnější datové struktury, případně kód zpřehlednit? Tato otázka má za úkol zjistit, zdali jsou studenti motivováni k úpravě jejich řešení v případě zisku plného počtu bodů.



Obrázek 5.1: Odpovědi studentů na první otázku

Z první otázky dotazníku na obrázku 5.1 vyplývá, že více než polovina studentů by použila zpětnou vazbu systému k vylepšení jejich řešení. Toto naznačuje možnost využití systému pouze pro generování zpětné vazby bez potřeby poskytovat bodové hodnocení.

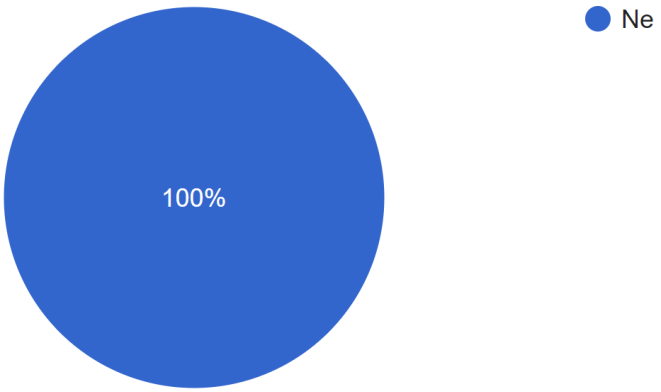
Měl by systém vždy penalizovat použití nevhodných konstrukcí, struktur a podobných? Tato otázka byla položena z důvodu mého rozhodnutí nepenalizovat studenty za styl jejich řešení.



Obrázek 5.2: Odpovědi studentů na druhou otázku

Z otázky dotazníku na obrázku 5.2 vyplývá, že většina studentů (83.3 %) s tímto rozhodnutím souhlasí. Menšina studentů by nebyla proti těmto penalizacím. Na základě výsledků se domnívám, že studenti by byli spíše rádi odměněni za správné řešení například ziskem bonusových bodů.

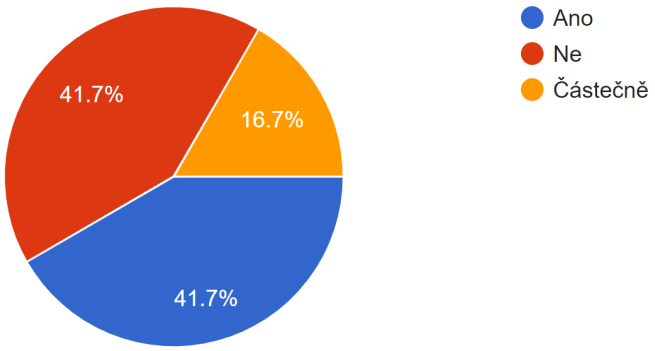
Chyběly vám systému nějaké funkce (pokud ano které)?



Obrázek 5.3: Odpovědi studentů na třetí otázku

Z otázky dotazníku na obrázku 5.3 vyplývá, že studenti jsou s funkcemi, které systém poskytuje spokojeni.

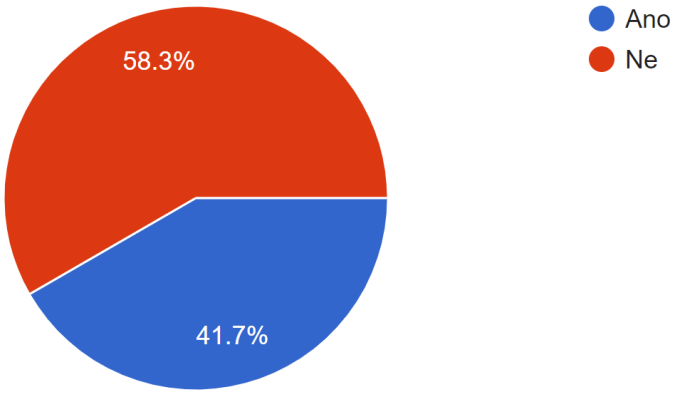
Použil jste pokročilou zpětnou vazbu (vyznačena v kódu podtržením)? Tato otázka byla zaměřena na zkušenosti studentů se samotným systémem.



Obrázek 5.4: Odpovědi studentů na čtvrtou otázku

Z otázky dotazníku na obrázku 5.4 vyplývá, že většina (58.4 %) studentů zpětnou vazbu alespoň částečně využila. Na dotazník zřejmě odpověděli studenti, kteří systém plně využili. Důvodů, proč studenti využili tuto zpětnou vazbu pouze částečně může být velký počet, například nedostatek času.

Měl jste problémy s orientací se v systému? Tato otázka byla zaměřena na zkušenost studentů s grafickým návrhem aplikace.



Obrázek 5.5: Odpovědi studentů na pátou otázku

Z dotazníku, obrázek 5.5, je zřejmé, že většina studentů (58.3 %) s orientací v systému problém nemá, menší část studentů (41.7 %) nebyla s grafickým návrhem spokojena. Některé části systému by bylo možné navrhnout jiným způsobem a to například zpětnou vazbu zobrazovat na oddělené stránce místo modálního okna a nebo ji v systému lépe vyznačit.

Kapitola 6

Závěr

V dnešní době vysokého počtu zájemců o programovací jazyky narůstá potřeba systémů schopných zvýšit efektivitu dosavadního řešení, nebo ji v budoucnu úplně nahradit. Tyto systémy také mohou být velmi žádaným nástrojem v případě krizových situací, které zamezují ve výuce kontaktní formou nebo jiným způsobem omezují výuku. Studenti používající systém mohou na projektech pracovat z domu, individuálním tempem a využívat je při zlepšení svého řešení zpětnou vazbu a doporučení generována systémem.

Navrhovaný a implementovaný systém je studentům schopen poskytovat zpětnou vazbu a zároveň bodově ohodnotit jejich soubory, a je ho tak možné použít k zvýšení efektivity při výuce jazyka Python v případě, že součástí předmětu je tvorba projektů. Systém byl nasazen v průběhu celého letního semestru, kdy, až na kratší nedostupnosti odpovídající systému ve vývoji, byl dostupný studentům pro kontrolu jejich řešení. Studenti se ve většině případů rozhodli využít systému pouze pro informaci o počtu bodů, které jejich řešení může získat. Pokud studenti získali maximální nebo uspokojitelný počet bodů, zpětnou vazbu od systému ignorovali. Systém byl také aktivně použit pro vyhodnocení všech zadaných projektů po jejich odevzdání rozhraním v příkazové řádce.

Finální systém se skládá ze dvou částí, první část je vytvořena pro studenta k vyhodnocení jeho řešení, druhá část slouží lektorovi k administraci. Pro vyhodnocení balíku řešení je možné použít rozhraní příkazové řádky.

Při dalším vývoji aplikace by bylo kromě připomínek studentů zjištěných v dotazníku směřováno na rozšíření možností spojených s virtualizací, a to například přidání možnosti zaměnit funkce použité studentem za funkce definované v testovacím souboru, přidání možnosti vyhodnocovat projekty z webového rozhraní nebo zlepšit zobrazení detekce plagiátů a to například vizuálně vyznačovat podobné části. Další možnou úpravou by bylo zlepšení struktury testovacích souborů.

Literatura

[1] Adu Manu, K., Arthur, J. a Amoako, P. Causes of Failure of Students in Computer Programming Courses: The Teacher Learner Perspective. *International Journal of Computer Applications*. Září 2013, sv. 77, s. 27–32. DOI: 10.5120/13448-1311.

[2] Altadmri, A. a Brown, N. C. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2015, s. 522–527. SIGCSE '15. DOI: 10.1145/2676723.2677258. ISBN 9781450329668. Dostupné z: https://doi.org/10.1145/2676723.2677258.

[3] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. a Bier, L. Clone detection using abstract syntax trees. In: *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 1998, s. 368–377.

[4] Brusilovsky, P., Schwarz, E. a Weber, G. ELM-ART: An intelligent tutoring system on world wide web. In:. Leden 1996, s. 261–269. DOI: 10.1007/3-540-61327-7_123.

[5] Corbett, A. T., Koedinger, K. R. a Anderson, J. R. Chapter 37 - Intelligent Tutoring Systems. In: Helander, M. G., Landauer, T. K. a Prabhu, P. V., ed. *Handbook of Human-Computer Interaction (Second Edition)*. Second Edition. Amsterdam: North-Holland, 1997, s. 849 – 874. DOI: https://doi.org/10.1016/B978-044481862-1.50103-5. ISBN 978-0-444-81862-1. Dostupné z: http://www.sciencedirect.com/science/article/pii/B9780444818621501035.

[6] Ettles, A., Luxton Reilly, A. a Denny, P. Common logic errors made by novice programmers. In:. Leden 2018, s. 83–89. DOI: 10.1145/3160489.3160493.

[7] Gegg Harrison, T. S. Exploiting program schemata in a Prolog tutoring system. In:. 1993.

[8] Jones, J. Abstract Syntax Tree Implementation Idioms. *Pattern Languages of Program Design*. 2003. Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003). Dostupné z: http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf.

[9] Karp, R. M. a Rabin, M. O. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*. 1987, sv. 31, č. 2, s. 249–260.

[10] Lane, H. a Vanlehn, K. Teaching the tacit knowledge of programming to noviceswith natural language tutoring. *Computer Science Education*. Září 2005, sv. 15, s. 183–201. DOI: 10.1080/08993400500224286.

[11] Le, N.-T., Strickroth, S., Gross, S. a Pinkwart, N. A Review of AI-Supported Tutoring Approaches for Learning Programming. Leden 2013, sv. 479, s. 267–279. DOI: 10.1007/978-3-319-00293-4-20.

[12] Mclaren, B., Scheuer, O. a Miksatko, J. Supporting Collaborative Learning and E-Discussions Using Artificial Intelligence Techniques. *I. J. Artificial Intelligence in Education*. Leden 2010, sv. 20, s. 1–46. DOI: 10.3233/JAI-2010-0001.

[13] Resnick, M., Maloney, J., Monroy Hernández, A., Rusk, N., Eastmond, E. et al. Scratch: Programming for All. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. listopad 2009, sv. 52, č. 11, s. 60–67. DOI: 10.1145/1592761.1592779. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/1592761.1592779.

[14] Robinson, V. a Kuin, L. The explanation of practice: Why Chinese students copy assignments. *International Journal of Qualitative Studies in Education*. Listopad 1999, sv. 12, s. 193–210. DOI: 10.1080/095183999236259.

[15] Roy, C. K., Cordy, J. R. a Koschke, R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*. 2009, sv. 74, č. 7, s. 470 – 495. DOI: https://doi.org/10.1016/j.scico.2009.02.007. ISSN 0167-6423. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0167642309000367.

[16] Schleimer, S., Wilkerson, D. a Aiken, A. Winnowing: Local Algorithms for Document Fingerprinting. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Duben 2003, sv. 10. DOI: 10.1145/872757.872770.

[17] Sichman, J. S., Coelho, H. a Rezende, S. O. *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006: 2nd International Joint Conference, 10th Ibero-American Conference on AI, 18th Brazilian AI Symposium, Ribeirao Preto, Brazil, October 23-27, 2006*. Springer, 2006.

[18] Wettel, R. *Automated Detection of Code Duplication Clusters*. Timișoara, RU, 2004. Bakalářská práce. Faculty of Automatics and Computer Science of the "Politehnica"University of Timișoara. Dostupné z: https://wettel.github.io/download/Wettel04a-BachelorThesis.pdf.

[19] Yudelson, M. a Brusilovsky, P. NavEx: Providing Navigation Support for Adaptive Browsing of Annotated Code Examples. In: *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology*. NLD: IOS Press, 2005, s. 710–717. ISBN 1586035304.

Příloha A

Obsah přiloženého paměťového média

- Tento dokument ve formátu pdf.
- Plakát práce ve formátu pdf.
- /Doc dokumentace projektu ve formátech html a LaTeX.
- /Thesis zdrojový kód tohoto dokumentu ve formátu LaTeX.
- /Project zdrojový kód práce.