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

SYSTÉM PRO VYHODNOCOVÁNÍ STUDENTSKÝCH KÓDŮ VE SKRIPTOVACÍCH JAZYCÍCH

SYSTEM FOR EVALUATION OF STUDENT CODE IN SCRIPTING LANGUAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VOJTĚCH BASTL**
AUTHOR

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

BRNO 2017

Zadání bakalářské práce/20087/2016/xbastl03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Bastl Vojtěch**
Obor: Informační technologie
Téma: **Systém pro vyhodnocování studentských kódů ve skriptovacích jazycích**
System for Evaluation of Student Code in Scripting Languages
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s nástroji pro automatické vyhodnocování programového kódu a s přístupy k samoevaluaci ve vzdělávání.
2. Na základě poskytnutých dat zpracujte přehled o problémech kódů odevzdaných studenty v rámci projektů v předmětu ISJ (Skriptovací jazyky).
3. Navrhněte a implementujte systém, který dokáže (polo)automaticky opravovat a vyhodnocovat projekty ISJ a provázat výstupy s Informačním systémem FIT VUT v Brně.
4. Vyhodnoťte vytvořený systém v rámci nasazení v letním semestru akademického roku 2016/2017.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle domluvy s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Funkční prototyp

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá nástroji pro automatické vyhodnocení kódu programátorů - studentů, kteří se teprve seznamují s konkrétním programovacím jazykem. Hlavní důraz je kladen na jazyk Python. Nejprve probírá problémy začínajících programátorů a diskutuje existující řešení. Jádro práce tvoří návrh a implementace nového systému, který je schopen automaticky vyhodnocovat studentská řešení konkrétních programovacích úloh v jazyce Python. Práce ukazuje vnitřní uspořádání a fungování, popisuje jeho vstupy a výstupy a demonstruje možnosti zařazení do výuky. Jako případová studie byl systém nasazen pro podporu výuky předmětu Skriptovací jazyky na FIT VUT v Brně v akademickém roce 2016/2017. Práce shrnuje zkušenosti z tohoto nasazení a hodnotí přednosti i problémy zpracovaného řešení.

Abstract

This work deals with the tools for automatic evaluation of the code of programmers - students, who are in the process to get to know a particular programming language. The main focus is on the Python language. First it discusses the problems of beginning programmers and discusses the existing solutions. The core of the work is to design and implement a new system that is able to automatically evaluate student solutions to specific Python programming tasks. The work shows the internal layout and operation, describes its inputs and outputs and demonstrates the possibilities of inclusion into the teaching. As a case study, the system was deployed to support the teaching of the subject Scripting Languages at the FIT VUT in Brno in the academic year 2016/2017. The work summarizes the experience of this inclusion and evaluates advantages and problems of the solution.

Klíčová slova

Python, automatické vyhodnocení programového kódu, webová aplikace, problémy studentů

Keywords

Python, automatic evaluation of programming code, web application, problems of students

Citace

BASTL, Vojtěch. *Systém pro vyhodnocování studentských kódů ve skriptovacích jazycích.* Brno, 2017. 63 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrž Pavel.

Systém pro vyhodnocování studentských kódů ve skriptovacích jazycích

Prohlášení

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

.....
Vojtěch Bastl
10. května 2017

Poděkování

Chtěl bych poděkovat Doc. RNDr. Pavlu Smržovi, Ph.D. za vedení, ochotu a trpělivost při řešení této práce. Dále bych chtěl poděkovat své rodině za podporu a trpělivost.

Obsah

Kapitola 1

Úvod

Podpora výuky prostřednictvím Internetu, případně její úplné přesunutí, je jedním z trendů 21. století. Vedle široké dostupnosti materiálů, záznamů jednotlivých přednášek i celých kurzů lze do tohoto trendu zařadit i automatické vyhodnocování zadávaných úkolů ve webovém prostředí. V oblasti podpory výuky programovacích jazyků, případně samovzdělávání, vzniklo v uplynulých letech množství nástrojů, dostupných pro široké spektrum jazyků, dosahujících různé úrovně automatizace a aplikovatelných v různých kontextech - od vizualizace exekuce programu a jeho paměti až po systémy schopné sbírat statistiky a hodnotit úkoly.

Tato práce si klade za cíl splnění hned několika úkolů.

- Seznámení čtenáře s nástroji pro vyhodnocení programového kódu ve webovém prostředí,
- diskutování problematiky začínajících programátorů,
- popis řešení, jež vzniklo v rámci této práce,
- popis chování vzniklého řešení v rámci nasazení v předmětu Skriptovací jazyky v akademickém roce 2016/2017,
- shrnutí poznatků.

V kapitole 2 se nejprve práce zaměřuje na popis chyb, které je nutno očekávat, pokud řešení problémů pochází od studentů začínajících s programováním. Dále budou, v kapitole 3 probrány již existující webová řešení, jež se zabývají tímto tématem, a tudíž mají vytyčené podobné cíle jako aplikace vytvořená v rámci bakalářské práce.

Následující kapitoly již popisují samotné řešení zadaného problému. Hlavním důvodem vzniku této kapitoly je cíl, aby čtenář po jejich přečtení byl schopen nejen tuto aplikaci využívat, ale také ji rozšířit či vytvořit vlastní implementaci na základě poznatků získaných z těchto kapitol. Návrh přihlíží k použití v reálném prostředí a má být schopen být snadno rozšiřován v dalších letech. Z těchto důvodů je důkladně popsán třídní návrh a vnitřní fungování aplikace jako celku (viz kapitola 4). Je zde také, v rámci kapitoly 5, popsán průběh a princip jednotlivých kontrol a akcí prováděných nad studentskými projekty.

Kapitola 6, popisující vstup, slouží k tomu, aby bylo možno aplikaci pohodlně použít při opravě projektů. Dále se práce věnuje inicializaci a způsobu spuštění aplikace. A to jak konzolové verze (viz kapitola 8), sloužící k hromadné opravě a bodování, tak webové aplikace, jakožto nástroje, jež je poskytnut studentům, kapitola 9.

V kapitole 7 práce popisuje možné výstupy aplikace, jejich tvorbu a mód aplikace, v němž je možno daný výstup použít.

Správnost výsledku bude demonstrována v kapitole 10, popisem průběhu testování. Zde je práce rozdělena do několika větších celků. Tyto celky jsou odděleny fází implementace, ve kterém se aplikace při testování nacházela. Jako první bude popsáno testování v průběhu samotné implementace základních principů, zde probíhalo testování s projekty vypracovanými studenty v rámci předmětu Skriptovací jazyky (ISJ) v akademickém roce 2015/2016. Poté se práce zaměřila na testování projektů v předmětu ISJ probíhající v akademickém roce 2016/2017, zde již byly základní pilíře aplikace implementovány a probíhalo odhalování chyb a nedostatků. Paralelně s tímto testováním probíhalo srovnání bodování se systémem, jež vznikal ve stejném akademickém roce v rámci diplomové práce: *Výuka pokročilých konstrukcí jazyka Python na základě poskytování zpětné vazby ke studentským kódům*. V rámci testování budou rovněž popsány jednotkové testy pokrývající kritické části aplikace.

Kapitola 2

Problémy začínajících programátorů

Chyby a problémy, kterých se začínající programátoři dopouštějí, jsou úzce spjaty s tématem této práce a s vznikem aplikace. Pro potřeby této práce budou chyby rozděleny do dvou kategorií:

- Omyly a chyby, jež se začínající programátoři dopouštějí nezávisle na programovacím jazyce,
- chyby nalezené v projektech ISJ v akademickém roce 2015/2016 a 2016/2017.

2.1 Chyby nezávislé na jazyku

I když se tento problém může zdát zanedbatelný a odstranitelný lépe navrženými konstrukcemi a klíčovými slovy programovacího jazyka, pánové Soloway a Spohrer ve svém výzkumu dokázali [35], že více než půlka chyb v programu začínajícího programátora jsou chyby, které nejsou závislé na použitém programovacím jazyce.

Jednu z forem chyb, které nejsou závislé na jazyku, popisuje článek: *Language-Independent Conceptual "Bugs" in Novice Programming* [31]. Zde je popsána skupina omylů vycházející z jediné analogie, jež začínající student k programování má, k přirozenému jazyku. Ve výsledném kódu tak vznikají nejednoznačnosti, které při přirozené konverzaci vyřeší cíl konverzace, tedy člověk. Bylo však zjištěno, že studenti popírají, že by v překladači či samotném počítači viděli rozum a doufali, že tyto chyby vyřeší. Vznikají z neschopnosti tyto nejednoznačnosti odhalit. Zmiňovaná chyba se dá dále rozdělit na tři typy:

- Chyba paralelismu [31, s. 27] - zde se programátor domnívá, že několik řádků programu, či celý program, je aktivní najednou. Tedy například návrat k již záporně vyhodnocené podmínce, pokud je její podmínka splněna dále v programu. Tento problém vychází z toho, že podmínka, očima přirozené konverzace, není vyhodnocena pouze jednou, ale je možno ji zpětně splnit po určitý časový interval. O chybě, která je zde označována jako "demon control structure", se zmiňuje i práce Bonar a Soloway [11, s. 138],
- inteční chyba [31, s. 29] - vzniká z domněnky, že kód má nějaký účel. Může pak docházet k chybám, že student přeskočí nepravdivě vyhodnocenou podmínku a udá, že program přesto vykoná věc, ke které byl "stvořen",

- egocetrická chyba [31, s. 30] - vzniká z myšlenky, že program je schopný domyslet některé skutečnosti, které jsou "jasné", či si je student myslel. Tyto chyby nejsou důsledkem domněnky o inteligenci systému, ale neschopností pomyslet na tyto nedostatky. Například malování čtverce příkazem čtyř tahů štětce, ale bez otáčení o devadesát stupňů.

Chybami vzniklými s použitím postupů známých z přirozeného jazyka se zabývá i práce: *Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers* [11]. Popisuje zde jev pojmenovaný "bug generators", nebo-li generátory chyb. Tento jev popisuje případ, kdy začínající programátor při vývoji programu narazí na slepou uličku a vyvine snahu o opravu formou záplat. Práce rozděluje generátory do tří hlavních kategorií [11, s. 144]. přičemž 60% chyb [11, s. 154] nalezených v programu odpovídá kategorie SSK Confounds PK. Kategorie popisuje skupinu generátorů, kde po uvíznutí ve slepé uličce žák použije strategii, jež je mu známá z přirozeného jazyka, specifikaci krok po kroku. Mezi tyto generátory například spadá:

- Programming language as natural language - využití programovací konstrukce z důvodu, že má stejný název jako fráze používaná v přirozeném jazyce. Například se jedná o použití klíčového slova `read` jako deklaraci pro provedení čtení, kdekoli to v algoritmu bude nutné,
- programming language interpreted as natural language - Vykládání konstrukcí programovacího jazyka jako by se jednalo o fráze použité v přirozeném jazyce. Například použití `then` pro vykonání příkazu po skončení smyčky, tak jak je tento problém popsán v přirozeném jazyce,
- multiple roles for a variable - použití jedné proměnné pro více rolí. Vychází z přirozeného jazyka, kde jedno jméno může mít v danou chvíli více významů, které jsou rozlišeny na základě kontextu,
- new programming construct language from natural language - zavedení nových konstrukcí a klíčových slov, na základě přirozeného jazyka.

Další odvětví generátorů chyb již nesouvisí s přirozeným jazykem, ale jedná se o špatnou či nekompletní znalost programování. Článek do této kategorie zařadil například tyto generátory:

- Programming language overgeneralization - jedná se o nadměrné zobecnění algoritmů. Například, nezávisle na potřebě, inicializace všech proměnných,
- tactical similarity - selhání rozlišení konstrukcí sloužící k podobným akcím. Například použití přiřazení při čtení ze vstupu.

Poslední typ generátorů chyb je využívání znalostí z jiných odvětí, než je přirozený jazyk nebo ze znalosti programování. Například sem spadá použití příkazu známého z operačního prostředí, či nezáměrné chyby způsobené nepozorností.

Další druh chyb, nezávislý na použitém programovacím jazyce, je svázán s proměnnými, konkrétně rozdílu výpočetního modelu v matematice oproti modelu v programování [23, s. 55]. Matematický model je primárně založen na substituci a navázání vztahu mezi levou a pravou stranou rovnice. Bez znalosti konkrétních hodnot v proměnných můžeme, podle syntaktických pravidel, zjednodušovat tyto rovnice. Naproti tomu v programovacím modelu

je proměnná spíše reference na konkrétní objekt či hodnotu. Výraz neznačí navázání vztahu mezi levou a pravou stranou, ale jedná se spíše o čistý výpočet pravé strany a předání jeho hodnoty do strany levé. Z tohoto nerozlišení může vzniknout hned několik omylů [23, s. 56]. Například použití substituce nebo líné vyhodnocení výrazů, kdy je výraz pouze "uskladněn"v proměnné a je předpokládáno, že výraz se vyhodnotí v případě potřeby.

2.2 Analýza chyb a problémů v odevzdaných projektech

Před samotným návrhem systému jsem na základě dostupných dat analyzoval nejčastěji se vyskytující chyby v projektech, odevzdaných v rámci předmětu Skriptovací jazyky v akademickém roce 2015/2016. Chyby v projektech lze dále rozdělit do dvou hlavních skupin: na chyby programové a chyby mimo programový soubor.

2.2.1 Chyby v programovém souboru

Zde je možné nalezené chyby rozdělit na chyby kosmetické a chyby, které ovlivňují vykonání programu.

Kosmetické chyby jsou takové, které žádným způsobem neovlivňují chod programu, ale jejich přítomnost může zhoršit přehlednost a čtivost programu. Například to může být porušení zásad definovaných v PEP8 [33]. Tyto zásady se v studentských projektech až na výjimky nedodržují. Nejčastěji jde o použití tabulátoru na odsazení, odsazení, jež není násobkem čtyř a chybějící bílé znaky, tam kde jsou doporučovány. Do této skupiny spadají i dokumentační řetězce [15]. Kromě případu, kdy zcela chybí, se jedná o použití špatného typu komentáře (použití klíčového znaku `#`), či špatné umístění (například nad definicí funkce). Mezi kosmetické chyby spadá i nedodržení jmenných konvencí a nevhodná volba jmen obecně. Kromě špatného schématu víceslovného jména (nové slovo ohlášeno velkým písmenem místo znaku `_` a naopak) se jedná i o příliš obecné pojmenování proměnných. V jednom případě bylo dosaženo opačného extrému a jména proměnných se skládala až ze třinácti slov.

První skupinou chyb, ovlivňující běh programu, jsou výjimky při importu souboru. Jedná se o chyby způsobené úpravou na poslední chvíli (špatné odsazení), používání konstrukcí a jmen dostupných pouze ve verzi Python 2.x, použití nedefinovaného jména a podobně.

Dále jsou zde zastoupeny chyby skryté, jež se prokážou až testováním programu. Například se může jednat o jméno, které není definováno v jedné z větví podmínky, globální proměnná, jež je inicializována pouze při importu a použita ve funkci, kde se očekává její výchozí hodnota, použití proměnlivých typů jako výchozí hodnoty argumentů a tak dále.

Chyby nebo neoptimální řešení vznikají také přechodem studentů z jiných, "nižších", programovacích jazyků, nejčastěji z programovacího jazyka C. Do této skupiny lze zařadit implementaci problémů, jejichž řešení je již součástí standardní knihovny[1]. Tyto snahy o řešení jsou ve většině případů pomalejší, méně spolehlivé a vzniká nepřehledný kód. Příklad může být snaha o implementaci permutace množiny, přičemž permutaci nabízí funkce `itertools.permutations`[2]. Do této kategorie rovněž spadá nevyužití pokročilých konstrukcí v jazyku Python, z důvodu, že jazyk C je nenabízí. Například použití cyklů na úkor generátorů, zvolení neoptimální struktury pro práci s daty (využití seznamu pro

[1] The Python Standard Library https://docs.python.org/3/library/

[2] itertools permutations https://docs.python.org/3/library/itertools.html#itertools.permutations

úschovu a vyhledávání řetězců), globální proměnné místo výchozích argumentů a využití vláken místo procesů.

Poslední skupina chyb vzniká z nepochopení zadání. Tato skupina je nejvíce rozšířená a obsahuje například: špatný typ návratové hodnoty, výtisk hodnoty na standardní výstup místo jeho navrácení, použití výjimky s klíčovým slovem `return` místo `raise`, vyvolání jiné výjimky, než je určena v zadání, jiné pojmenování funkcí, než je definováno v zadání, špatný počet argumentů funkce a podobně.

2.2.2 Chyby mimo programový soubor

V kategorii chyb mimo programový soubor se jednalo hlavně o špatně pojmenovaný soubor. V druhém projektu v rámci akademického roku 2015/2016 se jednalo o 184 z celkového počtu 214 odevzdaných projektů. Zde se jednalo, vzhledem k FIT VUT v Brně, o nekonvenční pojmenování souboru `Polynomial.py`. Projekty byly nejčastěji pojmenovány dle vzoru `login.py`. V rámci prvního a třetího souboru je toto číslo výrazně menší. V prvním projektu se jedná o 60 projektů alespoň s jedním špatně pojmenovaným projektem z 222 odevzdaných s tím, že 17 z nich je spíše chyba archivace, a to z toho důvodu, že složka s projektem obsahuje další zanořenou složku a až v ní se nachází soubory. V třetím projektu jde o pouze 39 špatně pojmenovaných projektů z celkového počtu 197. Z tohoto počtu se sedmkrát jednalo o chybu zanořeného souboru. Tento menší počet chyb může být způsoben dvěma faktory. Může se jednat o konvenci pojmenování, které více drží zvyklostí FIT VUT v Brně nebo o skutečnost, že projekt se skládá z více souborů a proto se student musí více zajímat o pojmenování.

Dalšími chybami, které se netýkají programového souboru, jsou chyby archivace. Kromě chyby se zanořeným souborem se jedná o manuálně přejmenované koncovky archivů nebo o jinak poničené archivy.

Do této kategorie také spadají chyby kódování. K této chybě dochází spíše ojediněle.

Kapitola 3

Rozbor řešené problematiky

Vzhledem k funkci, jakou aplikace popsána v této práci plní, budou v této části probrány a porovnány pouze řešení, které mají obdobné cíle. Tedy aplikace dostupné z webového prohlížeče, které nějakým způsobem podporují výuku programátorů.

3.1 Přednosti automatického vyhodnocení programového kódu

Před představením jednotlivých řešení musí být odpovězeno na otázku, zda automatické vyhodnocení kódu ve webovém prostředí má výhody oproti klasické formě.

Jedním z pozitiv je okamžitá zpětná vazba studentům. V článku *The role of feedback and self-efficacy on web-based learning* [38] je poukázáno na fakt, že okamžitá zpětná vazba použitá v rámci výuky ve webovém prostředí vede k nárůstu vnímání vlastní účinnosti [7]. Poskytnutí zpětné vazby je také jednou z dobrých zásad při výuce [12]. Každé popsané řešení, včetně objektu této práce, poskytuje jednu ze tří forem zpětné vazby [38, s. 1591]:

- Knowledge of result - jednoduché označení odpovědi za správnou, či nesprávnou,
- knowledge of correct response - informuje o obsahu správné odpovědi,
- elaborated feedback - vysvětluje důvod proč je odpověď správně, či špatně.

Při standardní formě projektů v rámci výuky je pro studenta těžké získat okamžitou zpětnou vazbu. Ta je buď nulová, nebo ji musí samostatně vyhledávat (studentské testy k projektům, vytvoření vlastní testové sady). Opožděná zpětná vazba už je více dostupná (konzultace, diskuzní fóra), ale v článku Roberta Dihoffa [16] bylo poukázáno na fakt, že opožděná zpětná vazba je již mnohem méně účinná.

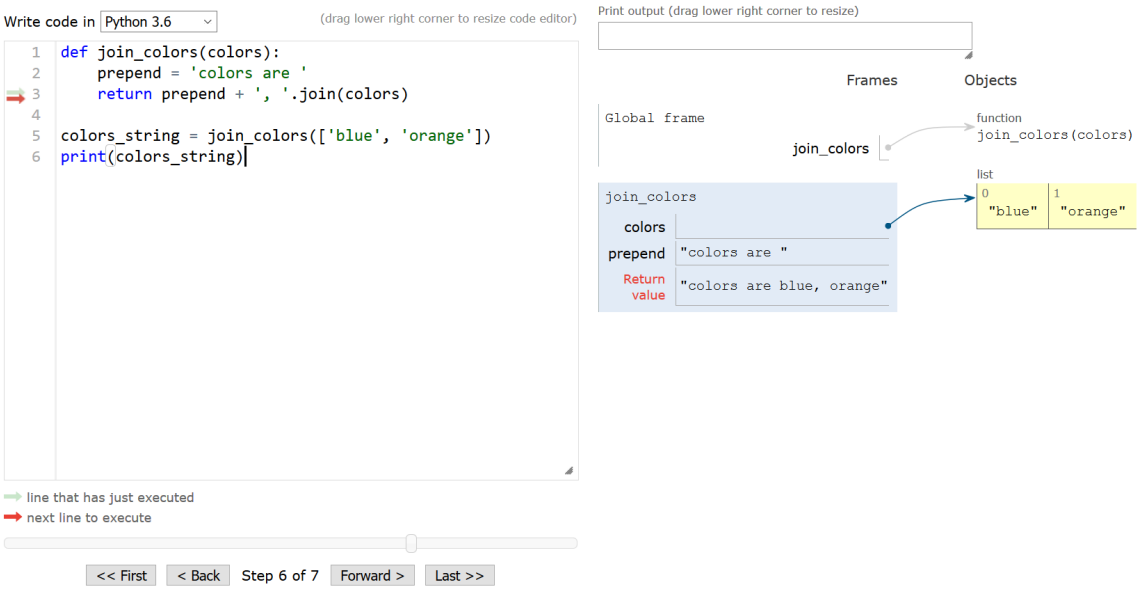
Dalším z důsledků použití těchto nástrojů může být zlepšení studentských výsledků, tomu nasvědčuje studie provedené s nástroji zyBooks [17] a CodeLab [10]. První výzkum se zaměřil na použití interaktivního nástroje oproti statickým textům, zde bylo zdokumentováno zlepšení o 16%. Druhá studie se zabývala nasazením nástroje CodeLab v rámci začínajících studentů. Zde bylo vypozorováno zlepšení výsledků z 2.0 na 2.2 na stupnici do čtyř [10, s. 74].

V článku *The effect of using online tutors on the self-efficacy of learners* [26] je zdokumentován nárůst vnímání vlastní účinnosti při použití těchto nástrojů. Zde se jednalo o měření s pomocí Bloomovi taxonomie [5]. S každým použitým nástrojem byl viditelný nárůst toho okruhu Bloomovi taxonomie, na který nástroj cílil [26, s. 7].

Přínos nemíří pouze směrem ke studentům, nýbrž i k instruktorovi. Použitím některého z těchto řešení vede k snížení zátěže na instruktora jednak, pokud je použita varianta s již přítomnými zadáními, odpadne vymýšlení úkolů, a také přítomnost automatických kontrol, u nichž je zajištěna konzistence hodnocení.

3.2 Python Tutor

Python Tutor[1] je známý podpůrný program pro jazyk python a vizualizaci průběhu jeho vykonání. K jeho běhu není zapotřebí žádných rozšíření [19, s. 1]. Také nabízí možnost integrace do jiných webových stránek. Mezi jeho hlavní přednost patří zobrazení paměti programu, kde jsou ukázány proměnné jednotlivých funkcí a objektů (viz obrázek 3.1). Tyto vlastnosti spolu s možností krokování programu představují silný nástroj pro výuku jazyka python. Tvrzení dokazuje, že se již zařadil do výuky některých univerzit. A je součástí několika webových kurzů [19, s. 2].

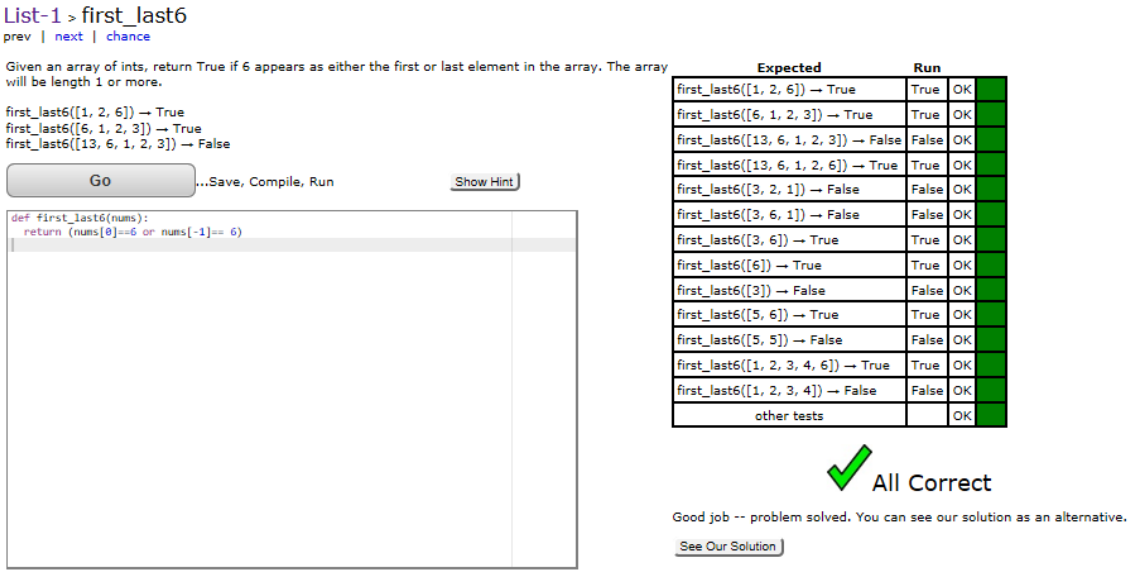


Obrázek 3.1: Příklad vykonání kódu v Python Tutor. Vlevo pracovní plocha. Vpravo vizualizace paměti

3.3 CodingBat

CodingBat [2] nabízí podporu jazyků python a java. Jedná se o soubor testových cvičení různých okruhů. Omezením může být, že všechna cvičení mají řešení v podobě funkce (viz Obrázek 3.2). Toto limituje úroveň znalostí programátorů, jelikož již musejí znát princip funkce. Vymyšlení vlastních problémů je pro začínající programátory velice pracné, proto mu tato existující zadání mohou pomoci procvičit si programování a poukázat na témata ve kterých má nedostatky. Cvičení je možné provést ve webovém prohlížeči bez doplňků. Aplikace zároveň nabízí možnost ukázat správné řešení problému a u některých cvičení i

[1] Python Tutor http://pythontutor.com/
[2] CodingBat http://codingbat.com/python

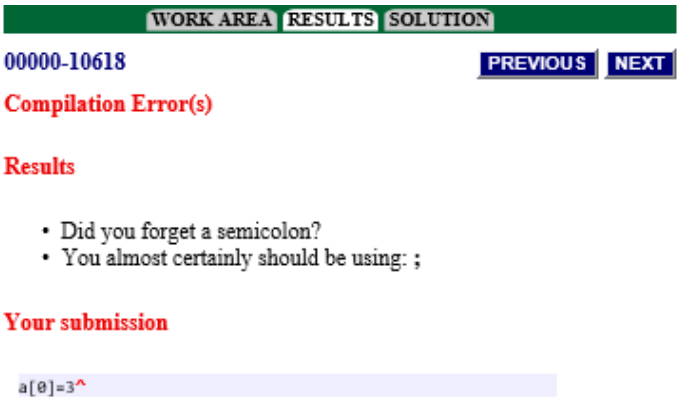


Obrázek 3.2: Ukázka příkladu v prostředí CodingBat. Vlevo je zadání a pracovní plocha, vpravo vyhodnocení testů.

nápovědu. Nabízí také rozhraní pro lektory. Ti si zde mohou definovat vlastní problémy. Student má poté možnost propojit svůj účet s lektorským.

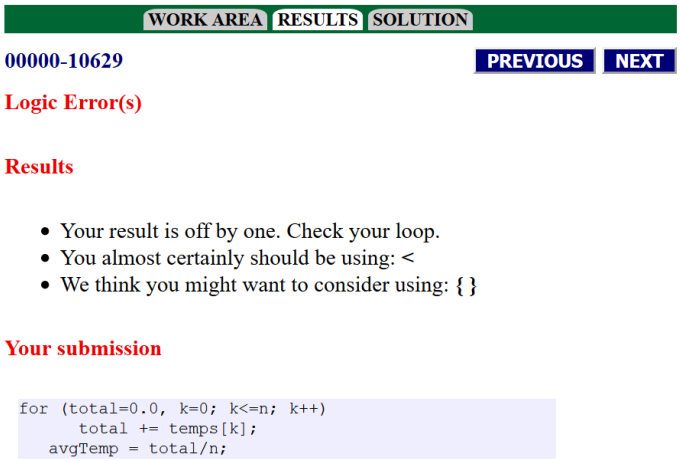
3.4 CodeLab

Jedná se o komerční webové řešení s širokým spektrem podporovaných programovacích jazyků[3]. Nabízí podobný princip jako CodingBat. Jedná se tedy o úkoly specifických témat, které mají stupňující se obtížnost. Kromě správných řešení, jež jsou k dispozici, je aplikace schopná poskytnout rady při špatném řešení (viz obrázek 3.3 a 3.4). Není zde možnost tvorby vlastní úkolů.



Obrázek 3.3: Ukázka syntaktické chyby v prostředí CodeLab.

[3]CodeLab https://www.turingscraft.com/



Obrázek 3.4: Ukázka nesprávného řešení příkladu v prostředí CodeLab.

3.5 Problets

Nabízí podporu výuky jazyků C++, c#, Java a visual basic[4]. Učení probíhá formou již hotových programů na různá témata a řešitelé mají za úkol zjistit chování těchto útržků kódu v určeném čase. Například u cvičení zaměřeného na výrazy je nutno správně označit pořadí operací a také vypočítat mezivýsledky (viz obrázek 3.5). U objektově orientovaného cvičení se pak jedná o zjištění chyb kódu a výstupu(viz obrázek 3.6). Následně je výsledek zkontrolován a příklad důkladně vysvětlen. Díky tomu, že cvičení jsou náhodně generována, lze je opakovat a systém je odolný vůči plagiátorství. Ke svému běhu potřebuje doplněk internetového prohlížeče jazyka java.

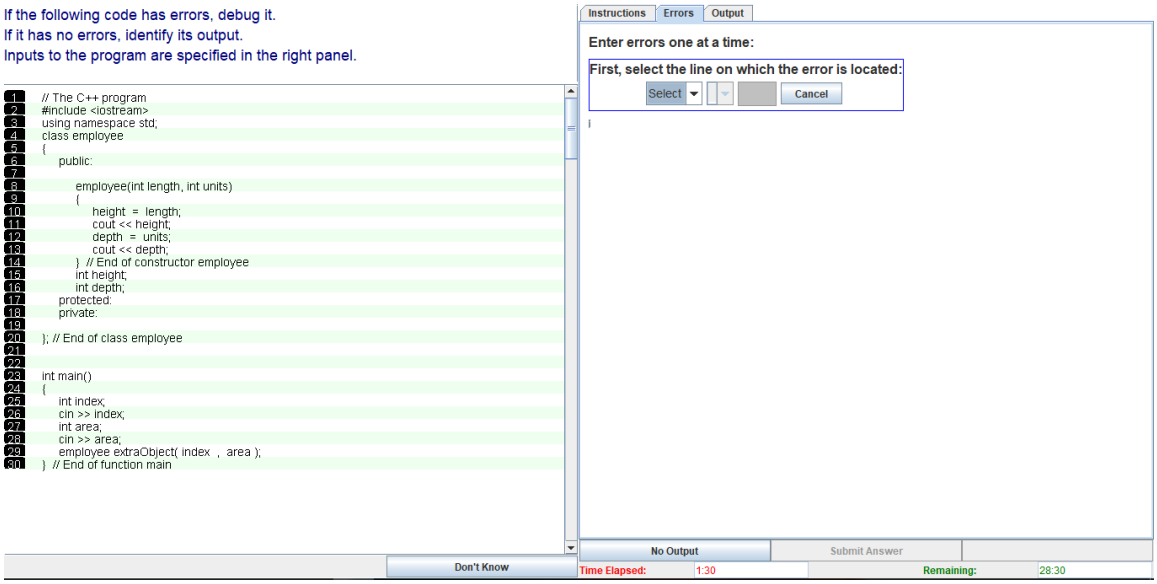
Obrázek 3.5: Problets s příkladem na vyhodnocení výrazu.

3.6 Porovnání řešení

Všechny zde popsané systémy jsou určeny k velice rozdílnému použití. PythonTutor nabízí nástroje, které mohou spět k hlubšímu pochopení jazyka Python. Jeho vizualizace paměti a krokování jsou ideální pro zobrazení rekurzí a jiných pokročilých konstrukcí. Na druhou stranu nenabízí žádná zadání ani žádné hotové příklady. Proto pro použití tohoto systému musí již subjekt mít vlastní zadání nebo dodat hotový kód.

Již vymyšlená zadání jsou schopny dodat aplikace CodingBat, CodeLab a Problets. První dva systémy se vydávají podobnou cestou, nabízejí zadání problému a textové pole, do kterého má být zapsáno řešení. Následně je toto řešení vyhodnoceno. Přítomnost již definovaných problémů může vézt k odhalení témat, ve kterých má student nedostatky.

[4]Problets http://problets.org/



Obrázek 3.6: Problets, příklad na OOP. Vlevo útržek programu, ve kterém mají být nalezeny chyby a zjištěn výstup. Vpravo možnost výběru řádku a problému v něm. Vpravo dole časový limit.

Možnost ukázky správného řešení, jež by se mělo blížit k ideální implementaci daného problému, nabízí seznámení řešitele s odlišným přístupem k problému a také může představit pokročilé vyjadřovací schopnosti jazyka. Druhé řešení, tedy CodeLab, je velice nápomocné při neúspěchu. Aplikace je schopná ukázat pravděpodobné důvody nezdaru.

Problets oproti těmto aplikacím dodá hotový útržek kódu a student s ním nadále pracuje. Tedy nemusí striktně znát klíčová slova a obecně syntax jazyka. Tento velice zdařilý princip brzdí potřeba běhu javy. Zprovoznit doplněk v rámci moderních prohlížečů je velice obtížné až nemožné [3].

Funkce aplikace, vzniklé v rámci této práce, se nachází trochu jinde. Její zaměření je vyhraněnější, i když se funkcí přibližuje druhému typu, tedy k CodingBat a CodeLab. Podobně jako tyto systémy nabízí aplikaci testů na studentský kód. Aplikace však nenabízí ukázku správného řešení a rady při neúspěchu. Místo toho nabízí bodování jednotlivých částí kontroly, kontrola, zda řádek obsahuje shebang [13] a jiné vlastnosti, jež jsou nezbytné pro kontrolu studentských projektů. Oproti představeným řešením, díky možnosti vyžadování dokumentačních řetězců a zásad definovaných v PEP8, podporuje aplikace studenty v psaní více přehledného kódu.

Kapitola 4

Koncepce

Hlavním požadavkem návrhu aplikace je možnost opravy velkého množství projektů. Tyto projekty musí být navzájem odděleny a nijak se neovlivňovat. Důležitá je také potřeba, aby byly zaznamenány chyby a problémy, které se u studentských projektů nalezly, a aby bylo možno rozlišit, kde k těmto problémům došlo. Neméně významný požadavek je na snadnou údržbu a rozšiřitelnost aplikace. Aby aplikace splňovala všechny požadavky a cíle je objektově orientovaná [27, s. 13] a rozdělená na tři hlavní celky. První se stará o samotný studentský projekt, tedy reprezentuje jednotlivé soubory a samotný projekt a provádí nad nimi kontroly. Druhá je pro záznam chyb a jejich srozumitelný výstup. Třetí část je pro generování výstupu. Významnou součástí návrhu je také abstraktní [30, s. 201] bázová třída [27, s. 20] `StateBase`. Od jedné z implementace této třídy dědí [27, s. 17] třída reprezentující projekt i třídy obalující soubory. Její rozhraní [27, s. 15] slouží jako propojovací prvek mezi opravami a záznamy problémů. Slouží také jako komponenta, která je dodána na výstup aplikace.

Výhodou tohoto přístupu je snadná paralelizace oprav projektů. Jedna z hlavních zásad paralelního vykonání je dbát na přístup k sdíleným zdrojům při použití blokující operace [37, s. 3]. Vzhledem k objektovému přístupu, kdy je každý projekt reprezentován vlastním objektem, a tomu, že zápis, jakožto blokující operace, do sdíleného souboru pro výstup, je proveden až po skončení paralelní části, je tato zásada dodržena. Jediný problém nastal při použití knihovny `psutil` k zjištění, zda student zavřel všechny využívané popisovače. Zde aplikace v paralelním chodu uvízla, řešením bylo použití sdíleného zámku[1].

K snadné rozšiřitelnosti přispívají dokumentační řetězce [15] přítomné v celé aplikaci a také `type hints`[32][34], tedy typování, jež komplexnější IDE umí využívat, ale které neovlivňuje typování jazyka python. Důraz je kladen na dodržení konvencí pojmenování proměnných, jmen funkcí, tříd a jiných jmen v programu. Jsou také dodrženy zásady formátování kódu definovaných v dokumentu PEP8.

4.1 Reprezentace projektu

Třídní návrh reprezentace projektů (viz příloha B.1) se snaží co nejvíce přiblížit reálnému obrazu. Z tohoto důvodu obsahuje třídu `Project`, která je abstrakcí studentského projektu a každá instance této třídy reprezentuje jednoho studenta. Tento objekt zároveň sdružuje všechny soubory, které mají být součástí řešení. V zájmu zachování polymorfismu [27, s. 17] jsou všechny třídy reprezentující soubory jednoho bázového typu, a to typu `FileBase`.

[1] Lock Manager.Lock

Tato abstraktní třída obsahuje metody a atributy, které jsou potřeba pro kontrolu souboru a zjištění jeho stavu, ale vlastní implementace těchto metod se liší u jednotlivých typů souborů. V této chvíli jsou konkrétní implementace tříd představující soubory dva a to třída zastupující soubor s kódem v jazyce python s název `FileCode` a třída `File`, jež reprezentuje soubor bez kódu. Toto rozlišení je naprosto klíčové, a to z toho důvodu, že nad `FileCode` probíhají dodatečné kontroly podle navolených možností a aplikace musí být takový soubor schopná importovat[2] a vykonat jeho kód.

4.2 Záznam chyb a problémů

Každý prvek kontroly projektu zaznamenává události do jedné instance z tříd, které dědí od abstraktní třídy `LogsComponentBase`. Tato třída má základní protokol potřebný pro záznam a výpis chyb. Bázová kompozitní třída [30, s. 409] `LogsBase` sdružuje tyto komponenty do větších logických celků, například `ProjectLogs`, jakož jedna z tříd, která je implementací `LogsBase`, má jako své komponenty objekty typu `ArchiveLogs` a `StructureLogs`, představující záznam problému při rozbalování archivů a problémy ve struktuře. Tedy pokud při rozbalování projektu dojde k výskytu chyby, objekt třídy Projekt ji zanese do svého logovacího objektu `ProjectLogs`, konkrétně do komponenty tohoto objektu `ArchiveLogs`. A naopak kód, jež se stará o zobrazení výstupu, může od tohoto objektu získat informace o chybách archivace.

4.3 Výstup aplikace

Výstup se děje přes systém pro tvorbu šablon jinja2[3], jemuž je dodán seznam výsledků oprav projektů, které jsou zabaleny do objektu `ProjectState`, dědice bázové třídy `StateBase`. Výstup je statický a jakmile skončí vykreslení, tak již nadále nekomunikuje s aplikací. Jeho funkčnost je zajištěna scriptem, běžícím na straně klienta v jazyce Javascript[4]. Grafický vzhled je zajištěn pomocí kaskádových stylů[5]. Navíc jsou k dispozici dvě volitelné formy výstupu. Textový výstup, jež má formu textového souboru pro každou provedenou kontrolu a soubor v syntaxi csv [4], pro import do systému WIS, v němž jsou u jednotlivých studentů vyplněné získané body.

4.4 Schéma aplikace

Aplikace je složena z několika balíčků[6], samostatných modulů[7], šablon, souborů obsahující klientské skripty a souborů s definicí vzhledu. Nejdůležitější část aplikace, ve které probíhá samotná kontrola projektu, se nachází v balíčku `Project`. Tento balíček obsahuje tyto moduly:

- `Base` - definice bázových tříd `StateBase`, `FileState`, `FileBase` a `ProjectState`,
- `Project` - obsahuje třídu `Project`,

[2] import https://docs.python.org/3/reference/import.html
[3] jinja2 http://jinja.pocoo.org/
[4] Javascript https://www.javascript.com/
[5] Cascading Style Sheets http://www.w3.org/Style/CSS/
[6] Packages https://docs.python.org/3/tutorial/modules.html#packages
[7] Modules https://docs.python.org/3/tutorial/modules.html

- `File` - modul pro reprezentaci souboru neobsahující kód. Obsahuje třídu `File`,
- `FileCode` - definuje třídu `FileCode`, představující soubor s kódem a třídu `LogsReport`, jež má na starosti PEP8,
- `MyPath` - definuje třídu `MyPath`,
- `ContentOfFile` - kontrola souboru oproti vzoru. Zde se nachází třídy `WrapperRows` a `RowOfContent`,
- `Code` - vykonání studentského kódu a kontrola nevhodných struktur. Třídy `CodeCheck`, `CheckEval` a `CheckBuildingString`,
- `StructureInCode` - kontrola podvrhů v souboru s kódem. Obsahuje tyto třídy: `CheckStructures` a `WhiteListDiffer`,
- `Testing` - provedení testů nad souborem s kódem. Třídy: `Tester`, `HtmlResult` a `HtmlTestRunner`.

Další balíček, starající se o záznam a výstup chyb s název `Logs`, poskytuje tyto moduly:

- `Base` - definuje bázové třídy pro záznam. `PointsHandler`, `LogComponentBase`, `LogsBase`, `StructureBase` a `LogToFile`,
- `Logs` - poskytuje třídy pro záznam problémů. Obsahuje všechny implementace tříd `LogsBase` a `LogComponentBase`,
- `LogToFile` - v tomto modulu jsou definovány třídy pro záznam do různých druhů souborů. třídy: LogToTxt, LogToCsv, LogToHtml.

Konstanty a tokeny použité v aplikaci a sloužící pro definici zadání jsou obsaženy v balíčku `Constants`. Zanořený balíček Logs obsahuje konstanty nalezených chyb ve studentském projektu. Modul `Token` definuje tokeny, které je možno použít při tvorbě zadání. Konstanty v modulu `FlaskConfig` je nutno použít v konfiguračním souboru webové aplikace.

Mimo tyto balíčky obsahuje aplikace i tři samostatné moduly. `YAMLParser` pro zpracování vstupního konfiguračního souboru, `flask_app`, definující webovou aplikaci a `isj_python_tutor`, jež je vstupní bod konzolové aplikace.

Soubory obsahující šablony webové aplikace jsou obsaženy ve složce `templates`. Soubory s kaskádovými styly, obrázky a klientské skripty jsou umístěny ve složce `static`

Kapitola 5

Architektura

V této kapitola se zaměřím na jednotlivé nezbytné části aplikace a jejich vzájemnou integraci. Vzhledem k tomu, že aplikace je cílena na reálné použití studenty,zde důkladně proberu systém kontroly podvrhů a také bodování projektu. Při vzniku aplikace jsem mimo jiné čerpal ze samotná dokumentace jazyka python [8] a také knížky *Learning Python* [28] a *Python 3: výukový kurz* [36] .

5.1 Reprezentace cest

Vzhledem k tomu, jak úzce je aplikace svázaná se souborovým systémem, by bylo nepraktické reprezentovat cesty textovým řetězcem. Dalším důvodem je možnost použití aplikace jak na operačních systémech Windows, uznávající reprezentaci cest pomocí zpětného i normálního lomítka, tak na systémech Unix, kde je cesta zapsána pouze s využitím lomítka. Proto je využívaná třída `MyPath`, potomek třídy `Path`[1] z balíčku `pathlib`[2]. Tato třída poskytuje kromě metod již obsažených v `Path` metodu `python_representation`, která vrátí řetězec reprezentující cestu souborovým systémem, s tím, že jednotlivé úrovně jsou odděleny tečkou, tedy notací, jež je využívána při importu modulů v jazyce python.

5.2 Třída Projekt

Mezi hlavní úkoly instancí této třídy patří kontrola reálné struktury studentského projektu oproti definici, řešení archivace, a také dodání výstupu. První dva úkoly se vykonají již při tvorbě objektu.

5.2.1 Archivace

Prvním činem vykonaným třídou Projekt před samotnou kontrolou struktury, je rozbalení těch částí projektu, které jsou archivovány v některém z druhů podporované komprimace. Kontrola pro přítomnost archivů probíhá jen na úrovni kořenového adresáře studentské složky. Tedy v kořenovém adresáři může být přítomno více komprimovaných souborů, ale archivy uložené v některé z podsložek již nebudou rozbaleny. V této chvíli jsou podporovány formáty zip, rar, tar a tar.gz s tím, že aplikace je schopna rozpoznat, zda se jedná o validní archiv a také zda archiv nemá manuálně přejmenovanou koncovku, a na tuto skutečnost

[1] Path https://docs.python.org/3/library/pathlib.html#pathlib.Path
[2] pathlib https://docs.python.org/3/library/pathlib.html

upozornit. Rozbalení souborů je destruktivní, tedy po skončení kontroly nejsou archivy obnoveny. K samotnému kódu, jež se stará o validaci a extrahaci archivů, je přistoupeno objektově, a to za použití již existujících tříd. K jednotlivých druhům komprese jsou použity odpovídající objekty. Tedy na formát zip je použita třída `ZipFile`[3] z balíčku `zipfile`[4], soubor tar obsluhuje balíček `tarfile`[5] a jeho třída `TarFile`[6] a konečně na kompresy rar je použita knihovna `rarfile`[7] s třídou `RarFile` [24]. Posledně jmenovaná knihovna není součásti standartní knihovny python.

5.2.2 Kontrola struktury

Kontrola probíhá oproti definici souborů, definice popisuje cestu k umístění souborů, jejich název, příznak, zda již byl soubor nalezen, a také obsahuje příznaky jednotlivých souborů.

Nejprve se v rekurzivní [25, s. 2] chráněné metodě [27, s. 69] `_recursive_check_structure` prochází reálný adresář. Soubory a složky jsou v každém kroku porovnány oproti vzoru a jejich nález je zanesen do vzoru. Přičemž jsou zaznamenány nalezené chyby. Po vykonání této metody je zavolána další rekurzivní metoda `_recursive_check_missing_folder`, její funkce je nalézt soubory a složky, které nebyly potvrzeny ve vzoru, a jejich nepřítomnost zaznamenat.

Jeden z problémů, kterým musí tato kontrola čelit, jsou soubory pojmenované dle studentova přihlašovacího jména [1]. Toto je řešeno tak, že se u definice souboru zkontroluje zda je přítomen příznak ohlašující tuto skutečnost, pokud ano, je u jména souboru nahrazen řetězec xlogin00 za jméno projektu, které je shodné s přihlašovacím jménem studenta. Vzhledem k tomu, že se každému objektu třídy `Project` dodává hluboká kopie[8] definice souborů, nejsou touto změnou ovlivněny následně kontrolované projekty.

Pokud se reálná struktura neshoduje s abstraktní definicí, je kontrola projektu ukončena. Jedinou výjimkou jsou soubory a složky, jež studentský adresář obsahuje navíc, toto je nutné, protože součástí studentského řešení mohou být soubory, které nejsou nutně potřebné pro řešení, ale jsou součástí řešení studenta.

5.2.3 Vykonání kontrol studentských souborů

Poslední povinností třídy `Project` je vykonání kontrol všech souborů. Toto se uskuteční zavoláním metody `check_whole_project`, ta pro každý Objekt typu `FileBase` zavolá metodu `check_whole_file`, vzhledem k tomu, že je tato metoda abstraktní, závisí vlastní implementace již na samotném objektu, kterému metoda patří.

Po skončení kontrol je jako návratová hodnota dodán objekt samotný s tím, že pomocí typování je řečeno že je dodán pouze typ `ProjectState` (viz kapitola 5.9.3). Objekty tohoto typu mají omezené spektrum metod a atributů, které jsou ovšem dostačující k zpracování výsledku kontroly projektu.

[3] ZipFile https://docs.python.org/3/library/zipfile.html#zipfile-objects
[4] zipfile https://docs.python.org/3/library/zipfile.html
[5] tarfile https://docs.python.org/3/library/tarfile.html
[6] TarFile https://docs.python.org/3/library/tarfile.html#tarfile-objects
[7] rarfile https://pypi.python.org/pypi/rarfile/
[8] Deep Copy https://docs.python.org/3.6/library/copy.html#copy.deepcopy

5.3 Třída FileCode

Instance této třídy, obalující soubory obsahující kód v jazyce python mají za hlavním úkol rozpoznat jaké kontroly mají být provedeny a následně je provézt s tím, že zjištěné chyby ve studentských kódech se zaznamenávají do jedné z komponent logovacího objektu `FileCodeLogs`. Mezi kontroly, které se provádějí patří:

- Kontrola shebang oproti vzorům,
- kontrola přítomnosti dokumentačních řetězců,
- kontrola doporučení formátování kódu definovaných v PEP8,
- nalezení nevhodných praktik,
- provedení testů (viz kapitola 5.6),
- kontrola abstraktních struktur, které mají být přítomny,
- případně kontrola souboru oproti vzoru.

Kontroly se provádějí podle pevného pořadí a to z důvodu, že některé závisí na úspěchu předchozích. Příkladem mohou být dokumentační komentáře jejichž přítomnost se zjišťuje z abstraktního syntaktického stromu[9] [22], reprezentující studentský kód získaný v průběhu hledání nevhodných praktik.

Algoritmus 5.1 Logika provedení kontrol

```
if je pozadovana kontrola shebang:
        kontrola shebang
abstraktni strom = vytvor abstraktni strom
if abstraktni strom je v~poradku:
        if je pozadovana kontrola dokumentacnich komentaru:
                kontrola dokumentacnich komentaru
        if je pozadovana kontrola PEP8:
                kontrola pep8
        if je pozadovana kontrola struktury kodu:
                kontrola abstraktni strukturu
                if je pozadovana kontrola oproti vzoru:
                        kontrola oproti vzoru
        vykonani abstraktni strom
        if vykonani abstraktniho stromu dopadlo v~poradku:
                if je pozadovano vykonani testu:
                    Vykonej testy
```

5.3.1 Kontrola Shebang

Kontrola přítomnosti a správného tvaru direktivy shebang probíhá v chráněné metodě `_check_shebang`. Zjišťuje, zda se na prvním řádku souboru nachází shebang, správný tvar

[9] Abstract Syntax Trees https://docs.python.org/3.5/library/ast.html#module-ast

a následně přítomnost v povolených tvarech shebang, které jsou definovány v konfiguračním souboru jehož obsah je v notaci yaml. Vzhledem k tomu, že mezera za úvodními znaky '!#' je volitelná [29], aplikace porovnává pouze část řetězce za těmito znaky.

5.3.2 Kontrola dokumentačních řetězců

Jedná se o dokumentační řetězce tak jak je definuje PEP257 [15].

Přítomnost dokumentačních řetězců je zjištěna pomocí abstraktního syntaktického stromu vytvořeného ze studentského kódu. Řetězce mohou být vyžadovány na úrovni modulu, ve funkcích, třídách a jejích metodách, který tento modul definuje. Jednotlivé úrovně hledání je možno potlačit v konfiguračním souboru. K získání požadovaného komentáře je použita metoda `get_docstring`[10] na uzlu, který je jedním z typů v němž jsou vyžadovány dokumentační řetězce.

5.3.3 Kontrola PEP8 doporučení

Kontrola probíhá s pomocí balíčku pep8 [21], jež není součásti standardních balíčků jazyka python. Je využíváno třídy `Checker`[11] a změněn je objekt, kterému instance této třídy předá nalezené chyby a zároveň místo kam se tyto chyby zaznamenají. Tohoto je docíleno při vytváření instance třídy `Checker` naplněním argumentů `reporter` a `log`. `Reporter` očekává objekt typu `BaseReport`[12] nebo některý z jeho podtypů v tomto případě `LogsReport`. `BaseReport` je objekt, jemuž je po kontrole předána množina chyb nalezených v kontrolovaném kódu. `LogsReport` jako jeho potomek plní stejnou funkci s tím, že přepisuje metodu `get_file_report`, která tuto činnost vykonává, aby vyhovovala potřebám aplikace. Log je místo kam se tyto chyby zaznamenávají. Jedná se o jednu z komponent logovacího objektu `FileCodeLogs` konkrétně instance třídy `Pep8Logs`.

5.3.4 Provedení kódu

Spoustu informací o studentském souboru lze získat až jeho vykonáním a také vytvořením abstraktního syntaktického stromu dále ast. Tyto akce se dějí v instancích třídy `CodeCheck`. Nejprve je vytvořen ast v metodě `parse_and_walk_tree`, již zde může dojít k výjimkám v syntaxi a odsazení kódu. Zároveň je zde zkontrolováno, zda jsou ve studentském kódu obaleny funkce `eval` a `exec` konstrukcí `try`. Následně je kód vykonán v metodě `exec_tree`, kód je vykonáván funkcí `exec`, které je dodán testovací `namespace`. Zároveň jsou před vykonáním kódu zablokovány standardní a chybový výstup a také standardní vstup. Vstup je zablokován z důvodu konstrukcí kdy studentský kód čeká na vstup a tím pádem není dále proveden. Je také změněn pracovní adresář na složku, ve které se nachází soubor. Volání funkce `exec` je obalena blokem `try` reagující na všechny standardní výjimky[13] a pokud k výjimce dojde je o ní student srozumitelnou formou informován.

[10] get__docstrings https://docs.python.org/3.6/library/ast.html#ast-helpers
[11] Checker http://pep8.readthedocs.io/en/release-1.7.x/api.html#checker-classes
[12] BaseReport http://pep8.readthedocs.io/en/release-1.7.x/api.html#report-classes
[13] Exception https://docs.python.org/3.6/library/exceptions.html#bltin-exceptions

Kontrola volání funkcí exec a eval

Hledání bloku `try` u těchto funkcí probíhá s objekty třídy `CheckEval`. Je vytvořeno tolik instancí kolik je bloků kódu na úrovni modulu. Kvůli potřebě naleznout volání funkce `eval` a `exec` je třída `CheckEval` potomkem třídy `NodeVisitor`[14].

Nejprve proběhne kontrola, zda je přítomná konstrukce `try` ve stejném bloku jako volání funkce `exec` nebo `eval`. Pokud tomu tak je, vznikne potřeba zjistit, zda je toto použití funkce uvnitř bloku `try`. V případě, že některá tato skutečnost není splněna, zbývá možnost, že použití těchto funkcí je uvnitř definice funkce a blokem `try` je obaleno každé volání funkce. Toto hledání musí být řešeno rekurzivně, protože funkce mohou být libovolně zanořeny.

Kontrola tvorby řetězce z pole

Konkrétně se jedná o hledání struktury `for`, v jejímž těle je obsažen jediný příkaz a to konkatenace řetězce. Jazyk python nabízí elegantnější řešení v podobě využití funkce `join`.

Kontrola probíhá v objektu třídy `CheckBuildingString`, potomka třídy `NodeVisitor`. Zde proběhne návštěva každého uzlu `for`. Následně je ověřeno, zda cyklus obsahuje jediný příkaz. Pokud ano je nadále zkontrolován. Příkaz musí splňovat tyto podmínky: Musí se jednat o přiřazení či rozšířené přiřazení. Pokud se jedná o prosté přiřazení, musí být jméno na levé straně výrazu přítomné i na pravé straně. Tímto je zajištěno, že potencionálně dochází k postupné tvorbě řetězce. Nadále je zkontrolováno jestli proměnná na levé straně výrazu je typu `string`. V posledním kroku dochází k ověření, zda je proměnná cyklu použita na pravé straně výrazu.

Pokud jsou splněny všechny tyto kroky, pak dochází k tvorbě řetězce. O kontrolu, či je proměnná cyklu v každém kroku typu `string` se postará překladač, který nedovolí operaci nad jinými než stejnými typy.

5.4 Kontrola podvrhů

Podvrhem se v tomto případě myslí případ, kdy si student uvědomí logiku kontroly a snaží se jí nějakým způsobem obejít. Například se může jednat o zjištění testovacích vstupů a správnou reakci pomocí jednoduché konstrukce `if/else` s vynecháním požadovaných konstrukcí. Další příklad může být již připravený kód, ve kterém má student upravit pouze některé řádky a snaha o podvrh smazáním některých rozhodovacích konstrukcí nebo jejich zjednodušení. Tomuto problému se do určité míry může předejít rozšířením testovacích případů, ale vymýšlení testovacích příkladů je nákladné na čas. Proto aplikace podporuje dva způsoby zamezení a odhalení podvrhů. Jedním je definice abstraktních struktur, které musí kód obsahovat. Druhým je kontrola oproti vzorovému souboru s tím, že jsou definovány upravitelné řádky.

5.4.1 Kontrola oproti vzoru

Pomocí konfigurační souboru YAML je řečeno, kde se nachází vzorový soubor a zároveň jaké řádky student může upravit (viz kapitola 6.5.1). Pro samotné porovnání je využívána třída `WhileListDiffer`,dědící od třídy `Differ`[15], jež je součásti balíčku `differ`[16] standartní

[14] NodeVisitor https://docs.python.org/3.6/library/ast.html#ast.NodeVisitor
[15] Differ https://docs.python.org/3.6/library/difflib.html#differ-objects
[16] differ https://docs.python.org/3.6/library/difflib.html

knihovny python. Implementace jednotlivých metod je upravená jen do té míry, aby byl respektován seznam řádků, které nemají být kontrolovány a zároveň aby byly nalezené neshody logovány do objektu místo na standartní výstup. Tímto objektem je instance třídy `DifferLogs`. `DifferLogs` je součástí logovací komponenty `StructureLogs`.

5.4.2 Kontrola abstraktní struktury

Definice kontroly je zadána přes konfigurační soubor (viz kapitola 6.5.1). V aplikaci je reprezentovaná jako stromová [20, s. 79] struktura s neomezeným větvením objektů třídy `StructureDefinition`. Těchto stromů se zpětnými ukazateli může být několik, kde každý strom označuje syntaktickou strukturu, která má být nalezena. Uzly na stejné úrovni, se stejným předkem, znamenají konstrukce, jež se v kódu musejí nacházet ve stejném bloku kódu. Toto znamená, že rodičovský uzel musí být schopen zneplatnit svoji větev, pokud se již kontrola dostala na konec bloku, který značí, a nebyla nalezena celá větev.

Třída StructureDefinition

Každá Instance této třídy představuje jednu konstrukci, kterou musí kód obsahovat. Vzhledem k logice kontroly, kdy jsou potvrzovány listy stromu, musí být uzel schopen odkázat na svého rodiče, aby mohl tento rodič zkontrolovat, zda již byli nalezeny všichni potomci.

Třída CheckStructures

Samotné průchod studentským kódem a hledání konstrukcí se děje s použitím instance potomku třídy `NodeVisitor` s název `CheckStructures`. `NodeVisitor` poskytuje základní funkčnost pro návštěvu uzlů ve stromu. Pokud si přejeme uzel dále prozkoumat je nutnost definovat metodu s názvem `visit_node`[17], kde `node` je název jednoho z typu uzlů, které jsou reprezentovány třídami, jež dědí od bázového typu `AST`[18]. S tímto principem počítá hledání struktur.

Třída `CheckStructures` definuje statické metody s parametry, odpovídající instančním metodám, jež slouží k podrobnějšímu zkoumání uzlů abstraktního syntaktického stromu. Tyto metody jsou připojeny k instanci po její tvorbě pomocí zavolání třídní metody `determine_finding`. Rozhodnutí, které metody budou instanci přiděleny, se děje podle poskytnutých objektů `StructureDefinition`. Přidělení těchto metod instanci způsobí, že uzel typem odpovídající názvu metody, je do této metody zaslán. Tělo těchto metod využívá jednu z funkcí `check_node_with_body` nebo `check_node_without_body`.

První metoda slouží pro kontrolu uzlů, které mohou obsahovat další větve abstraktního syntaktického stromu. Je jí jako parametr dodán seznam objektů `StructureDefinition`. Metodě jsou předány jen objekty stejného typu jako je navštívený uzel. U každého `StructureDefinition` je zjištěno, zda se jedná o list definice, pokud ano, je, v případě potřeby, zkontrolováno jméno uzlu a nastaven nález struktury. V případě, že objekt `StructureDefinition` obsahuje potomky, musí být tito potomci rovněž nalezeni v studentském kódu. Z tohoto důvodu je vytvořena nová instance `CheckStructures`, jež je nastavena podle těchto potomků a s její pomocí je navštívena každá větev abstraktního syntaktického stromu obsažená v uzlu, jež je zrovna zkoumán.

Metoda `check_node_with_body` pouze nastaví každý obdržený objekt `StructureDefinition` jako nalezený.

[17] visit https://docs.python.org/3.6/library/ast.html#ast.NodeVisitor.visit
[18] ast.AST https://docs.python.org/3.6/library/ast.html#node-classes

5.5 Třída File

Objekty reprezentují jakýkoli soubor, kromě těch obsahujících kód. Jejich úlohou je porovnání souboru oproti vzoru a vykonání testů (viz kapitola 5.6). Porovnání oproti vzoru je inicializováno pomocí příznaků v konfiguračním souboru. Vzor musí být reálný soubor a řečena je pouze cesta, na které se nachází. Výstup z kontroly v HTML souboru může být buď viditelný s přehledem chybných řádků, nebo je pouze řečeno zda je obsah správný či nesprávný a je možnost poskytnutí nápovědy.

5.5.1 Inicializace kontroly

Instancializace objektu, který provádí vlastní porovnání probíhá v chráněné metodě `_load_pattern_from_file`. Zde vznikne objekt třídy `WrapperRows`, jež slouží k porovnání vzoru a studentského souboru.

Nahrání obsahu studentského souboru je provedeno v chráněné metodě `_load_and_check`, výchozí kódování je utf-8, pokud toto čtení selže je použita knihovna `chardet`[19] pro rozpoznání použitého kódování.

5.5.2 Třída WrapperRows

Každý objekt této třídy odpovídá jednomu vzorovému souboru. Z důvodu toho, že tvorba objektu ze vzorového souboru může být časově nákladná, je tento objekt sdílený mezi jednotlivými opravami projektů. Kvůli paralelismu aplikace tohoto nemůže být docíleno použitím dekorátoru [9]. Místo toho je učiněn název třídy chráněným `_WrapperRows` a je definovaná funkce s jménem, které odpovídá jménu třídy a s atributy stejnými jako konstruktor. Tato funkce spolu s globálním slovníkem `inst_of_wrapper` se stará o konstrukci objektů a uchováním těchto objektů v slovníku `inst_of_wrapper`. Pomocí typování je pak řečeno, že je vrácen objekt třídy `_WrapperRows`.

Pokud první projekt požádá o vytvoření instance třídy `_WrapperRows`, dekorátor tuto instanci vytvoří ze vzorového souboru a uloží ji do slovníku `inst_of_wrapper`, následujícímu projektu již není vytvořená nová instance, ale je mu znovu inicializovaná instance již existující. Aby bylo rozpoznáno, zda je již vzorový soubor zpracován a instance tedy existuje, je klíč v tomto slovníku cesta adresářovou strukturou ke vzoru.

Inicializování objektu pro nový projekt spočívá v změně atributu s názvem studenta, pro soubory obsahující texty s názvy studentů, zneplatnění příznaku nalezení řádku a změně objektu, do kterého je proveden záznam.

Kvůli možnosti, že stejný objekt bude využívat několik procesů zároveň, je slovník `inst_of_wrapper` jedinečný pro každý proces, tedy jsou instance sdíleny jen v rámci jednotlivých procesů.

Řádky vzoru jsou uloženy v atributu `list_of_instances`. Jedná se o pole objektů `RowOfContent`. Tyto objekty reprezentují jednotlivé řádky vzoru. Jsou zde uchovány informace o obsahu řádku, číslu na kterém se nacházel ve vzoru, zda už byl nalezen a jiné pomocné hodnoty.

- `__init__(pattern_location, project_name, case_sensitive, line_sensitive, login_sensitive, log)` – konstruktor objektu. Za argumenty má lokaci vzoru, zda má být porovnávání citlivé na velikost písmen, jestli mohou být prohozeny řádky

[19] chardet https://pypi.python.org/pypi/chardet

a mají-li být řetězce xlogin00 hledány jako studentovo přihlašovací jméno. Poslední argument slouží pro objekt do kterého budou zaznamenány nalezené nesrovnalosti,

- `re_init(project_name, log)` – metoda pro nachystání kontroly nového souboru,
- `compare_file_to_pattern(file_location)` – základní metoda pro použití třídy. Provede porovnání vzoru oproti souboru který je přítomen na cestě `file_location`. Návratová hodnota typu boolean udává, zda se kontrola provedla bez nalezené chyby,
- `check_if_identical_row(line_content, line_number)` - kontrola, zda řádek ve studentském souboru je totožný s řádkem na stejné pozici ve vzoru.

  Vrací dvojici hodnot boolean a řádek ze souboru, který je upravený vzhledem k nastavení kontroly (citlivost na velikost písmen) pro rychlejší pozdější zpracování,
- `check_if_other_row(line_content, line_number)` – metoda volána při neúspěchu předchozí. Její chování je různé vzhledem k nastavení objektu. Pokud je možnost, že řádky mohou být prohozeny, je nalezen první stejný řádek ve vzoru a kontrola je úspěšná. V opačném případě je snaha najít stejný řádek a toto špatné umístění oznámit studentovi.

  V každém případě pokud toto hledání selže je zjištěno, zda je řádek mimo rozsah vzorového souboru. Pro řádky v rozsahu, je nalezena nejbližší shoda pomocí knihovny `difflib` a její funkce `get_close_matches`, o této podobnosti je student spolu s vzorovým obsahem řádku informován. O řádcích mimo rozsah vzorového souboru je student informován jako o redundantních,
- `check_if_all()` – metoda volána po projití celého studentského souboru. Zanese do záznamu vzorové řádky, které nebyli nalezeny.

5.5.3 Provedení kontroly

Kontrola je provedena v metodě `_check_content_lines`, vzhledem k složitějšímu nahrání obsahu souboru není použita výchozí metoda objektu `WrapperRows`, sloužící ke kontrole. Místo toho jsou použity tři dostupné metody pro kontrolu oproti vzoru: `check_if_identical_row`, `check_if_other_row` a `check_if_all`. Všechny nalezené neshody jsou zaneseny do komponenty `ContentLogs` v logovacím objektu třídy `FileLogs`.

5.6 Provedení testů nad souborem

Testy se provádějí jak nad soubory obsahující kód, třída FileCode, tak i soubory, které kód neobsahují,třída File. Jsou syntaxí totožné se standardním testováním v jazyce python. Jedná se tedy o testování s použitím balíčku `unittest`[20]. Tento balíček nabízí objektový přístup k testovacím příkladům a široké spektrum nástrojů, které podporují psaní a přehlednost testů.

5.6.1 Nalezení a inicializace

O nalezení studentských testů se starají potomci bázové třídy `Tester`. Třída `FileCode` využívá potomka `CodeTester` a třída `File` využívá třídu `FileTester`. Rozdíl těchto nadtříd

[20] Unit testing https://docs.python.org/3/library/unittest.html

lpí v importování, respektive neimportování studentského souboru. Pomocí konfiguračního souboru YAML je řečeno, v jaké složce jsou moduly obsahující testovací třídy (viz kapitola 6.7). Zároveň je v tomto souboru řečeno, jaké testové třídy mají které metody, tento podrobný zápis je z důvodu možnosti bodování jednotlivých metod a nemá žádný vliv nato, jaké testy budou vykonány . Tyto soubory s testy se musí automaticky upravit pro každého studenta, aby byly provedeny nad správným studentským souborem. Toto se děje vytvořením dočasného souboru obsahujícího všechny řádky originálního souboru, ke kterým je přidán, v případě nadtřídy `CodeTester`, řádek obsahující import všech jmen definovaných ve studentském kódu. Zároveň je přidán řádek definující proměnou `student` obsahující přihlašovací jméno studenta a v případě třídy `FileTester` je přidána proměnná `file_location`, jež obsahuje cestu k souboru. Po vykonání testů je tento dočasný soubor smazán. Testovací třídy samotné musí být potomky třídy `TestCase`[21] z balíčku `unittest`. K načtení testů ze souboru je použita třída `TestLoader`[22], jejíž již vytvořenou instanci v atributu `defaultTestLoader`[23] nabízí modul samotný.

5.6.2 Vykonání testů

K vykonání testů slouží instance třídy `HtmlTestRunner`, potomek třídy `TextTestRunner`[24], Z důvodu zapisování do objektu místo do textového proudu bylo nutné přepsat všechny dostupné metody a je tedy nadtřídou jen z důvodu zachování rozhraní. Jedním z argumentů konstruktoru této třídy je objekt, který se stará o záznam problémů. Tímto objektem je `HtmlResult`, jedná se o instanci podtřídy třídy `TestResult`[25]. Třídě `TestResult` je přepsán konstruktor a zároveň mu jsou přidány metody `log_errors` a `log_error_list` pro záznam výsledků testů do logovacího objektu.

Algoritmus 5.2 Logika provedení testů spolu s importem studentského souboru

```
if cesta k~testum is soubor:
        testove soubory = cesta k~testum
else:
        testove soubory = vsechny soubory ve slozce s~priponou 'py'
for testovy soubor in testove soubory:
    with open(docasny soubor):
        with open(testovy soubor):
            docasny soubor = obsah testoveho soubor
        docasny soubor += 'import studentskeho souboru'
        docasny soubor += 'prihlasovaci jmeno studenta'
        testy += testy z~docasneho souboru
        smazani docasneho souboru
for test in testy:
        vykonani testu
```

[21] TestCase https://docs.python.org/3/library/unittest.html#unittest.TestCase
[22] Testloader https://docs.python.org/3/library/unittest.html#unittest.TestLoader
[23] Defaulttestloader https://docs.python.org/3/library/unittest.html#unittest.defaultTestLoader
[24] TextTestRunner https://docs.python.org/3/library/unittest.html#unittest.TextTestRunner
[25] TestResult ttps://docs.python.org/3/library/unittest.html#unittest.TestResult

Algoritmus 5.3 Logika provedení testů bez importu souboru

```
if cesta k~testum is soubor:
        testove soubory = cesta k~testum
else:
        testove soubory = vsechny soubory ve slozce s~priponou 'py'
for testovy soubor in testove soubory:
    with open(docasny soubor):
        with open(testovy soubor):
            docasny soubor = obsah testoveho soubor
        docasny soubor += 'cesta k~souboru'
        docasny soubor += 'prihlasovaci jmeno studenta'
        testy += testy z~docasneho souboru
        smazani docasneho souboru
for test in testy:
        vykonani testu
```

5.7 Záznam problémů

Záznam problémů se skládá ze dvou bázových tříd. Třídy `LogsComponentBase`, jejíž dědicové slouží pro konkrétní logování, a `LogsBase`, která instance potomků třídy `LogsComponentBase` sdružuje do větších celků.

5.7.1 Abstraktní třída LogsComponentBase

Poskytuje jednotné rozhraní a základní metody pro záznam chyb a upozornění.

Metody a atributy třídy LogsComponentBase

- Abstraktní metoda `log(self, *, err_code, **kwargs)` – metoda, kterou dědic musí implementovat. Slouží pro záznam. V jejím těle může být rozhodnuto, jestli jsou obdržené informace uloženy do atributu sdružující významné chyby `_error` nebo do `_warning`, zde se nacházejí upozornění, tedy chyby které nejsou tak vážné. Rozhodnutí je učiněno podle parametru `err_code`. Jedná se o jednu z konstant definovaných v balíčku `Constants.Logs`,
- abstraktní metoda `get_error_html()` – touto metodou je poskytnut výstup. Základní typ, který se předpokládá na výstupu je seznam řetězců v notaci HTML. Pokud se očekává mnoho záznamů, je možno jako návratovou hodnotu poskytnout pouze generátor těchto řetězců,
- abstraktní metoda `get_error_text()` – jedná se o alternativu k metodě `get_error_html()`, skrze ni lze získat textový výstup logu,
- `check_if_ok()` – kontrola, zda objekt obsahuje záznam chyby. Tato metoda nezohledňuje upozornění,
- `get_number_of_error()` – vrátí počet nalezených chyb,
- `get_number_of_warning()` – vrátí počet nalezených upozornění.

5.7.2 Abstraktní třída LogsBase

Jedná se o abstraktní třídu s čistě abstraktními metodami a bez atributů, jde tedy o formu rozhraní[27, s. 72]. Její jednotlivé implementace slouží jako obal objektů typu `LogsComponentBase` k tomuto účelu nabízí také metody.

Abstraktní metody třídy LogsBase

- `check_if_ok()` – kontrola, zda ani jedna z komponent objektu nezaznamenala chybu,
- `check_if_empty()` – rozšiřuje tuto kontrolu i na upozornění,
- `get_number_of_error()` – vrátí součet nalezených chyb v jednotlivých komponentách,
- `get_number_of_warning()` – návratová hodnota této metody je počet upozornění nalezených v komponentách,
- `get_number_of_points()` – vrátí `namedtuple`[26] typu `Points` reprezentující počet získaných bodů.

Jednotlivé implementace třídy LogsBase

- `FileCodeLogs` – instance této třídy využívá třída `FileCode` a sdružuje tyto implementace třídy `LogsComponentBase`: `Pep8Logs`, `ShebangLogs`, `DocLogs`, `ExceptionLogs`, `CodePracticeLogs`, `UnitTestLogs` a `StructureDefinitionLogs` (viz příloha B.3),
- `FileLogs` – objekt využívá třída `FileLogs` a skládá se z komponent `ContentLogs` a `UnitTestLogs`,
- `ProjectLogs` – reprezentuje záznam chyb pro třídu `Project`. Obsahuje tyto dědice třídy `LogsComponentBase`: `StructureLogs` a `ArchiveLogs` (viz příloha B.2),
- `YamlParserLogs` – tato implementace je rozdílná od prvních tří, slouží pouze pro záznam problémů nalezených při inicializaci aplikace. Neskládá se z komponent a logování probíhá rovnou do tohoto objektu.

5.8 Bodování

Definice bodování jednotlivých kontrol je zapsáno v souboru YAML (viz kapitola 6.5.2). Logika bodování vychází z toho, že pokud nebyl nalezen žádný problém, za který by měly být strženy body, je dán jejich maximální počet. Tato logika je uplatněna až na grafickém výstupu. Po celou dobu provádění oprav je naopak uchováno, kolik bodů by mělo být strženo. Tento údaj se mnohem lépe uchovává a jsou na něm lehčeji prováděny operace. Každá komponenta, která chce mít možnost bodování musí dědit od abstraktní třídy `PointHandler`.

[26] namedtuple https://docs.python.org/3/library/collections.html#collections.namedtuple

5.8.1 Abstraktní třída PointHandler

Tato třída má za úkol zachování bodů a poskytnutí operace nad nimi. I když z ní nemohou být vytvořeny instance každá z jejich metod je implementovaná. Při volání konstruktoru musí být řečeno, jaký je maximální počet bodů, který může být stržen a také optimální informace, jež udává krok s jakou se bude k maximálnímu počtu blížit.

Metody a atributy

- `points_to_subtract` – proprieta, která uchovává počet bodů k strhnutí,
- `max_sub_points` – proprieta pro uschování maximálního počtu bodů, které se mohou za komponentu odečíst,
- `subtract_points()`– zavolání této metody vede k zvýšení bodů, jež mají být strženy a to o jeden krok. Pokud krok není definován, je odečten maximální počet bodů,
- `get_number_of_points()`– vrátí `namedtuple Points`, obsahující prvky `subtract` a `maximum`, tedy počet již odebraných bodů a maximální počet bodů ke stržení.

5.8.2 Výjimečné stavy

Jakýkoli stav, kdy neproběhnou všechny kontroly do konce, je stav výjimečný a bodování se ním musí vyrovnat. Jedním z takovýchto stavů je studentský soubor vyvolávající výjimku na globální úrovni modulu, tedy při importu. Toto je řešené komponentou `ExceptionLogs`, již je oznámeno, že maximální počet bodů k stržení je součet všech kontrol, které se provádějí až po importu kódu, stejný počet bodů dostane i komponenta pro kontrolu struktury souboru. Pokud studentský projekt nesplňuje adresářovou strukturu, je mu odečten počet bodů, které lze získat za všechny soubory. Tedy bodový zisk se rovná nule.

5.9 Provázání s výstupem

Po skončení metody `check_projects` je jako návratová hodnota nastaven slovník, kde klíče jsou název opraveného projektu a hodnota je objekt `Project`, respektive pomocí typování je řečeno, že je vrácen pouze jeho předek `ProjectState`. Tímto je dosaženo toho, že je na výstup poskytnuto pouze rozhraní, které přímo souvisí s dokončením opravy. V rámci zachování logiky je slovník vrácen i pokud dojde k opravě pouze jednoho projektu. Tato třída dědí od abstraktní třídy `BaseState`.

5.9.1 Abstraktní třída StateBase

bázový typ pro zachování stavu kontroly.

Metody a atributy

- `log` – hlavní položka každého objektu typu `StateBase` a jeho podtypů. Jedná se o jednoho z potomků třídy `LogsBase`,
- abstraktní metoda `check_if_ok()`– metoda pro získání stavu,
- abstraktní metoda `get_points()` – získání `namedtuple` typu `Points`.

5.9.2 Abstraktní třída FileState

Abstraktní třída pro zachování stavu kontroly souborů. Jedná se o potomka třídy `StateBase`. Implementuje metody, deklarující třídou `StateBase` a zároveň přidává atribut `flags`. Tedy pole příznaků, které kontroly byli nad souborem provedeny.

5.9.3 Abstraktní třída ProjectState

Obsahuje výsledek kontroly studentského projektu. Dědí od třídy `StateBase`. Navíc definuje abstraktní metodu `get_files_state`, tato metoda vrací slovník, kde klíče jsou název souboru a hodnoty objekty `FileState`. Metodu implementuje třída `Project`.

Kapitola 6

Vstup aplikace

Pro konfiguraci celé aplikace slouží soubor se syntaxí YAML. S jeho pomocí je možno říct množinu projektů, které budou inicializovány. Jaká struktura bude po nich požadována a průběh jejich bodování. Pokud je vyžadována direktiva shebang, je v pro ní určené sekci seznam řetězců shebang, jež jsou povoleny. Tokeny popisující tuto definici, jsou definované v modulu `Constants.Token`

6.1 Výběr projektů

Tato část konfiguračního souboru označené tokenem `PROJECTS_TO_CHECK` je rozdělena na dvě části.

První sekce popisuje cestu k hledaným projektů. Je onačena konstantou PATH a hodnota musí být validní cesta směřující od pracovní složky aplikace k projektům či cesta absolutní. Pokud je řetězec `PATH` nepřítomný, je výchozí umístění projektů pracovní adresář.

Druhá část je regulární výraz v syntaxi knihovny re[1], který je podobný výrazům v jazyce Perl [6]. Tato část je označena tokenem `PATTERN_TO_MATCH` a pokud tento token není přítomný, je výchozí hodnota regulární výraz odpovídající standardní konvenci přihlašovacích jmen na FIT VUT [1]. Oproti tomuto regulárnímu výrazu jsou pak porovnávány názvy složek na cestě zadané první sekci.

Výčet 6.1 Popis hledaných projektů: projekty se nacházejí ve složce `example/projects` a jsou nalezeny projekty, které mají platný login.

```
PROJECTS_TO_CHECK:
    PATH: example/projects
    PATTERN_TO_MATCH: '(x[a-zA-Z]{4,5}\d{2}(?:\..+$|$))'
```

6.2 Popis Struktury

Popis struktury je v sekci konfiguračního souboru označené konstantou `STRUCTURE`. Samotná struktura je popsaná pomocí dvou entit: složek a souborů. Soubor se od složky liší tím, že

[1] re https://docs.python.org/3/library/re.html

již v sobě nemá zanořenou další entitu, ale naopak obsahuje seznam příznaků, deklarujících kontroly, jež budou provedeny.

Výčet 6.2 Popis sruktury: projekt obsahuje soubor `xnovak00.py` a složku s názvem `example`, která obsahuje soubor `example.txt`. Nad soubory není prováděna žádná kontrola.

```
STRUCTURE:
    xnovak00.py:
    example:
        example.txt:
```

6.3 Seznam dovolených direktiv shebang

Pro účely definice povolených shebang je vyhrazena konstanta `SHEBANG`. Z důvodu, že platných direktiv může být hned několik, jsou zadány pomocí pole jehož jednotlivé prvky jsou dovolené direktivy. Je uznáván i samotný řetězec neobalený polem. Pokud je definice vynechána výchozí hodnota je `#!/usr/bin/env python3`

Výčet 6.3 Deklarace povolených direktiv shebang.

```
SHEBANG: ['#!/usr/bin/env python3', '#!/usr/bin/env python']
```

6.4 Globální definice bodů

Jedná se o sekci označenou tokenem `POINTS`. Tato sekce slouží spíše pro kontrolu, zda součet jednotlivých zadaných bodování není větší nebo menší než počet definovaný zde. V této chvíli má smysl pouze jediný argument této sekce `MAX_POINTS`, číslo které ho následuje je maximální počet bodů jaký může student za projekt získat.

Výčet 6.4 Globální definice bodů

```
POINTS:
    MAX_POINTS: 5
```

6.5 Nastavení souboru obsahující kód

Nastavení souboru se skládá ze seznamu příznaků. Nejdůležitější příznak je `CODE`. Tento příznak oznámí aplikaci, že se jedná o soubor obsahující kód a je s ním v tomto směru zacházeno. Další možné příznaky jsou:

- `DOCSTRING` – po souboru budou požadované dokumentační řetězce, jako hodnotu příznaku je možno nastavit slovník, kde klíč `CONFIG` udává, na jaké úrovni mají být dokumentační řetězce vyžadovány,

Hodnota prvku `CONFIG` je seznam, ve kterém mohou být prvky: `MODULE`, `CLASS`, `FUNCTION`, `METHOD`. Pokud toto není řečeno, budou vyžadovány všechny čtyři typy řetězců,

- `SHEBANG` – bude zkontrolován první řádek souboru oproti definici shebang,
- `PEP8` – kontrola doporučení psaní kódu, jak je definovaná v PEP8,
- `LOGIN` – řetězec xlogin00 v názvu souboru bude nahrazen názvem projektu, tedy přihlašovacím jménem studenta,
- `STRUCTURE` – definice abstraktních struktur, které se mají nacházet v souboru. Také zde může bát definovaná kontrola oproti vzoru,
- `TESTS` – definice testování souboru (viz kapitola 6.7),
- příznak `LOGIN`, řetězec `xlogin00` je nahrazen přihlašovacím jménem studenta.

6.5.1 Definice hledaných struktur

Začátek sekce je vymezen konstantou `STRUCTURE`, jedná se o stejný řetězec, jaký je použitý při ohlášení sekce definice struktury projektu. Rozdíl lpí v místě použití konstanty. Pokud je použita jako příznak souboru, je na ní nahlíženo jako definice struktur souboru. Uvnitř této sekce je struktury popisující studentský kód. Jedná se o pole, jehož jednotlivé prvky mohou být dvojího typu: kontrola oproti vzoru a abstraktní struktura.

Kontrola oproti vzoru

Tento typ může být definován pouze jednou u každého souboru. Jako parametr vyžaduje cestu ke vzorovému souboru, s nímž musí být studentský soubor totožný. Dále obsahuje parametr s názvem `WHILELIST`, který určuje jaké řádky mohou být rozdílné oproti vzoru.

Abstraktní kontrola struktury

Může být přítomno vícekrát, kdy každý výskyt značí samostatný kořen. Vzájemně zanořené slovníky představují abstraktní syntaktický strom. Každý uzel má definován typ, potomky a pokud je to zapotřebí tak i jméno. Dostupné typy jsou v souboru `Constant.Token` v části popisu typů struktur.

6.5.2 Bodování komponent

U tokenů `SHEBANG`, `DOCSTRING`, `CODE`, `CONTENT`, `PEP8`. Pokud se za nějakým z těchto klíčů objeví číselná hodnota, je brána za maximální počet bodů, ke strhnutí. Je zde také možno uvést slovník. Jehož prvky jsou počet bodů a krok, tento krok značí, jaká část bodů bude při každé chybě odečtena. Pokud tento krok není přítomen, již za první chybu je stržen maximální počet bodů.

Výčet 6.5 Definice hledaných struktur: soubor `example.py` bude kontrolován oproti obsahu souboru `pattern.py` s tím, že řádky 9 a 10 budou z této kontroly vynechány. Zároveň tento soubor musí obsahovat třídu s názvem `Example` s metodami `__init__` a `__str__`. Tento soubor musí také obsahovat metodu `__main__`.

```
STRUCTURE:
    example.py:
        - CODE
        - STRUCTURE: [
          {TYPE: DIFFER, WHITELIST: [9, 10], PATH: 'pattern.py'},
          {TYPE: CLASS, NAME: Example, CHILDS: [
              {TYPE: FUNCTION,  NAME: '__init__'},
              {TYPE: FUNCTION, NAME: '__str__'}
          ]}
          {TYPE: FUNCTION, NAME: '__main__'}
        ]
```

Výčet 6.6 Bodování bez kroku: za jakýkoli chybějící dokumentační komentář v souboru `example.py` bude odečteno 2 body.

```
STRUCTURE:
    example.py:
        - CODE
        - DOCSTRING: 2
```

Výčet 6.7 Bodování s krokem: za chybějící výskyt dokumentárních komentářů bude strženo maximálně 2 body s krokem 0,5 bodů.

```
STRUCTURE:
    example.py:
        - CODE
        - DOCSTRING:{MAX_SUB_POINTS: 2, STEP_SUB: 0.5}
```

6.6 Nastavení souboru neobsahující kód

Soubor, jež nebude importován a nebude na něj nahlíženo jako na soubor obsahující kód, je ohlášen nepřítomností příznaku `CODE`. Lze zcela vynechat příznaky a poté je požadovaná pouze jeho přítomnost, jež se dá kombinovat s obdržením bodů za přítomnost souboru. Chování souboru určuje několik argumentů a příznaků:

- Argument `CONTENT_IN_FILE` ohlašující porovnání souboru oproti vzoru. Následuje ho cesta ke vzorovému souboru,
- příznaky `CASE_SENSITIVE`, `LINE_INSENSITIVE`, `LOGIN_SENSITIVE` a `SECRET` určující průběh porovnání souboru se vzorem. Popořadě znamenají: citlivost porovnání na velká a malá písmena, ignorování prohození řádků, nahrazení řetězce `xlogin00` přihlašovacím jménem studenta a zamlčení výstupu z porovnání souboru oproti vzoru s tím, že může následovat řetězec, jež bude poskytnut jako nápověda,

- argument `POINTS`, jehož hodnota, jež následuje, definuje body získané za porovnání souboru nebo existenci souboru. Je možno uvést slovník s určením kroku, s nímž se odečítají body,
- sekce `TESTS` definující testování souboru (viz kapitola 6.7),
- příznak `LOGIN` ohlašuje případ, kdy řetězec `xlogin00` v názvu souboru je nahrazen přihlašovacím jménem studenta.

Výčet 6.8 Definice souboru neobsahující kód: nechť je nad souborem `example.txt` provedena kontrola oproti vzoru s názvem `example_pattern.txt`. Kontrola je citlivá na velká a malá písmena. Za každou nalezenou chybu nechť je strženo půl bodu do maximálního počtu dvou bodů. Studentovi nejsou ukázány jeho chyby, ale je pouze poskytnuta nápověda s textem: Soubor musí obsahovat ukázkový text.

```
STRUCTURE:
    example.txt:
        - CONTENT_IN_FILE: 'example_pattern.txt'
        - CASE_SENSITIVE
        - POINTS:{MAX_SUB_POINTS: 2, STEP_SUB: 0.5}
        - SECRET: 'Soubor musí obsahovat ukázkový text'
```

6.7 Definice testování

Sekce testování je oznámena tokenem `TESTS`. Sekce je přípustná u jakékoli definice souboru.

V této sekci je konstanta `PATH`, hodnotou u této konstanty je řečena cesta na které se testy nachází. Pokud se jedná o složku, testy jsou hledány ve všech souborech s koncovkou py. V opačném případě, kdy je cesta přímo soubor, jsou testy hledány pouze v tomto souboru.

Další položky v této sekci jsou jednotlivé testovací třídy a u každé třídy seznam testovacích metod. Vzhledem k tomu, že jsou provedeny všechny testy v souborech, které jsou řečeny první částí je toto oznámení testovacích metod jen z důvodu definice bodování. Body jsou definovány číselnou hodnotou u jména metody.

Výčet 6.9 Definice tesování: nechť jsou nad souborem `example.py` provedeny všechny testy v souboru `example/test.py`. Za testovou metodu `test_first_task` je strženo maximálně jeden bod za testovou metodu `test_second_task` body dva.

```
STRUCTURE:
    example.py:
        - CODE
        - TESTS:
            PATH: 'example/tests.py'
            TestExample:
                test_first_task: 1
                test_second_task: 2
```

6.8 Modul pro zpracování vstupu

Zpracování konfiguračního souboru se syntaxí YAML je provedeno v souboru `YamlParser.py`, tento modul, který je obdobou návrhového vzoru jedináček [18], využívá k zpracování konfiguračního souboru balíček `PyYAML` [2].

6.8.1 Rozhraní modulu YamlParser

- `parse_file(task_path)` – tato funkce má za úkol zpracování konfiguračního souboru yaml, jehož umístění bylo poskytnuto skrz parametr `task_path`.

  Během svého běhu vytvoří všechny potřebné skutečnosti pro následný vznik samotného objektu třídy `Project`,

- `check_projects(names_of_project, project_name)` – podle parametru `names_of_project` vytvoří požadované instance třídy `Project` a poté provede jejich kontroly. Parametr je seznam jmen projektů. Výchozí hodnota je `None`, v tomto případě jsou zkontrolovány všechny dostupné projekty.

  Parametr `project_name` slouží pro webovou aplikaci, díky němu je zjištěno přihlašovací jméno studenta. Pokud je v případě konzolové aplikace vynechán, jsou jména nastaveny podle jmen složek projektů.

  Vrací seznam instancí `ProjectState`. Kde každý prvek tohoto seznamu odpovídá kontrole jednoho projektu,

- `check_projects_para(names_of_project)` – vykonává totožnou činnost jako předchozí funkce s tím rozdílem, že je oprava projektů vykonána paralelně. K tomuto účelu je použit balíček `multiprocessing`[3] a jeho třída `Pool`[4]. Instance této třídy je využita jako kontextový manažer[5]. Jako parametr konstruktoru je mu dodán počet procesů, který je schopen procesor, na kterém aplikace běží, dodat. Toto číslo je zjištěno pomocí funkce `cpu_count`[6]. V nitru kontextového manažera si každý proces vezme jeden z množiny projektů a vykoná opravy, tomuto projektu je navíc předán sdílený zámek, který je využit v aplikaci. Jakmile ukončí opravu projektu, vezme si nezávisle na stavu ostatních procesů projekt další.

[2] PyYAML http://pyyaml.org/
[3] multiprocessing https://docs.python.org/3.6/library/multiprocessing.html
[4] Pool https://docs.python.org/3.6/library/multiprocessing.html#multiprocessing.pool.Pool
[5] Context Manager https://docs.python.org/3/reference/datamodel.html#context-managers
[6] cpu_count https://docs.python.org/3.6/library/os.html#os.cpu_count

Kapitola 7

Výstup aplikace

Výstupy jsou definované v modulu `Logs.LogToFile`. Aplikace umožňuje dodat tři odlišné druhy výstupu. Každý z nich je popsán samostatnou třídou dědící od abstraktní třídy `LogToFile` (viz příloha B.4). Jedná se o HTML soubor, soubor ve formátu CSV a textové soubory. S tím, že druhý a třetí typ je k dispozici pouze pro konzolovou aplikaci.

7.1 HTML výstup

Tento výstup je tvořen pomocí systému pro tvorbu šablon jinja2. Vzniklý HTML soubor je plně responzibilní a nepožaduje žádné doplňky. Hlavní část této responzibility je dosažena pomocí flexbox pozicování elementů [14] definovaných v kaskádových stylech. Tato vlastnost mírně omezuje kompatibilitu webových prohlížečů [2], ale propůjčuje vlastnosti, jež již nemusí být řešeny pomocí skriptovacího jazyku.

Pro vznik HTML souboru použije aplikace jednu ze svých dvou základních šablon. O tom, která verze je aplikována, rozhoduje počet opravených projektů a také to, zda byla aplikace použita jako konzolový script nebo webová aplikace.

Pokud je opraven pouze jeden projekt nebo došlo k opravě přes webovou aplikace je použita šablona definovaná v souboru `log_output_one.html` (viz příloha C.2). V případě opravy více než jednoho projektu je šablona v souboru `log_output_all.html` (viz příloha C.3). Hlavní rozdíl těchto šablon spočívá v přítomnosti navigace skrz projekty a také v dalších přibalených souborech obsahující definici vhledu a chování, které toto menu vyžaduje. K přihlédnutí k faktu, že může být opravováno velké množství projektů, jsou jednotlivé výsledky kontrol zabaleny do písmen české abecedy. O tom do jakého písmene výsledek spadá je rozhodnuto podle druhého písmene, toto je kvůli podoby přihlašovacího jména na VUT BRNO.

Vzhledem k tomu, že primární cíl těchto šablon je stejný, a to zobrazení výsledku kontroly projektu, je tato část separována do dalšího souboru `projects.html`. Soubor se již sám stará o samotné vykreslení jedné kontroly projektu. Jednotlivé své opakující se části dále separuje do maker, tak jak to umožňuje jinja2.

7.2 Soubor ve formátu CSV

Aplikace nabízí dva způsoby, jak se stará o výstupní soubor csv. Může aktualizovat již vytvořený soubor csv nebo vytvořit nový. Obě tyto akce probíhají ve stejné třídě `LogToCsv`. Rozhodnutí akce je učiněno na základě poskytnutých parametrů při spouštění konzolové

aplikace. Výstupní soubor slouží pro import bodů do systému WIS, které studenti obdrželi za řešení projektu. Aby byl možný import, musí soubor csv obsahovat hlavičku s těmito sloupci:

- Prázdný sloupec,
- Jméno – jméno studenta,
- Login – login studenta,
- Body – body obdržené za projekt,
- Celk – celkové body v rámci předmětu,
- Datum – datum obdržení bodů,
- Kdo – hodnotitel, je možno zadat pomocí parametrů příkazové řádky.

Soubor je vytvářen a aktualizován pomocí objektů a metod, které nabízí knihovna csv. Důležité je zmínit, že dialekt csv souborů, jež generuje a podporuje systém WIS není této knihovně známý. Proto musel být zjištěn z vygenerovaného souboru ze systému WIS pomocí instance třídy `Sniffer`[1], po zjištění již mohla být tato detekce smazána a dialekt je při startu aplikace přidán pomocí funkce `register_dialect`[2] do známých dialektů knihovny `csv`[3].

7.2.1 Aktualizace existujícího souboru

V módu aktualizace souboru csv jsou nejprve pomocí objektu `reader`[4] procházeny řádky originálního csv souboru a každý řádek je spolu s vyplněným sloupcem, jež odpovídá bodovému zisku studenta uložen do lokálního pole. Následně je tento soubor přepsán verzí s vyplněnými body. Tohoto je docíleno pomocí objektu `writer`[5], který každý prvek dočasného pole zapíše jako nový řádek do prázdného csv souboru.

Výhoda aktualizace již existujícího souboru je v tom, že budou přítomny všechny hodnoty sloupců, jež definuje hlavička, také skutečnost, že nebude přítomný záznam studenta, který není součástí kurzu a lehká kontrola studentů, jež nebyli opraveni.

7.2.2 Tvorba nového souboru

Tvorba probíhá podle obdobného principu jako aktualizace. Je zde také využito objektu `writer` a generován je soubor se stejným dialektem. Hlavička souboru je zachována, ale vyplněny jsou pouze sloupce: Login, Body, Datum, Kdo. Výhoda tohoto přístupu spočívá v tom, že nemusí být exportován existující soubor csv ze systému wis a také možnost opakovaného spuštění kontroly se zápisem bodů.

[1] Sniffer_dialect https://docs.python.org/3.6/library/csv.htm\T1\l#csv.Sniffer
[2] register_dialect https://docs.python.org/3.6/library/csv.htm\T1\l#csv.register_dialect
[3] csv https://docs.python.org/3.6/library/csv.html
[4] reader https://docs.python.org/3.6/library/csv.html#csv.reader
[5] writer https://docs.python.org/3.6/library/csv.html#csv.writer

7.3 Textový výstup

Vzniká ve třídě `LogToTxt`. Jedná se o soubor textových dokumentů, v nichž jsou základní informace o průběhu kontroly daného projektu. Tyto informace jsou zjištěny z logovacích objektů. Zároveň vznikne i soubor `general.txt`, jež obsahuje seznam oprav a počet udělených bodů. Tyto krátké logy mohou sloužit pro stručné informování studentů.

Kapitola 8

Konzolová aplikace

Konzolová aplikace slouží jako nástroj, jež by měl primárně používat lektor. Z tohoto také vychází její zaměření, mezi které patří oprava pouze podle jednoho zadání, ale většího množství projektů, paralelní zpracování, zápis do souboru pro import do systému WIS a potencionálně také výpis chyb v konfiguračním souboru zadání.

8.1 Spuštění konzolové aplikace

Konzolovou aplikaci obsahuje modul `isj_python_tutor.py`, pro svůj běh používá jazyk python 3.6.0. Nižší verze nejsou podporovány hlavně z důvodu použití typování a oznámení instančních proměnných. Kromě standardních knihoven aplikace potřebuje pro svůj běh tyto balíčky: `Flask`, `Flask-Session`, `Jinja2`, `PyYAML`, `Twisted`, `Werkzeug`, `chardet`, `pep8`, `psutil` a `rarfile`. K jednoduchému spuštění se doporučuje již připravené virtuální prostředí[1] na serveru athena3 se jménem `venv`. Parametry spuštění tohoto scriptu jsou:

- -t, - -t – povinný argument, cesta k konfiguračnímu souboru opravy,
- -c, - -check – názvy projektů, které mají být zkontrolovány. Přípustný je jeden a více názvů,
- -p, - -paralel – příznak paralelního zpracování,
- -q, - -quit – potlačení výpisu na standardní výstup,
- -o, - -output – cesta k výstupnímu souboru, pokud neexistuje bude vytvořen a s ním i cesta k němu. Musí mít koncovku html. Při absenci je nastaveno na `result.html` v kořenovém adresáři aplikace,
- -s, - -student_text_log – následovaný cestou ke složce slouží k zapnutí výstupu ve formě krátkých textových souborů, kde každý soubor náleží jedné provedené opravě,
- -w, - -wis – export bodů do souboru csv. Přípustná hodnota je cesta k csv souboru. Tento soubor a ani cesta k němu nemusí existovat. V případě že soubor již existuje, bude přepsán,
- -u, - -wis_update – aktualizace již existujícího souboru csv. Parametr musí být cesta k již existujícímu souboru csv,

[1] Virtual env https://virtualenv.pypa.io/en/stable/

- -l, - -lektor – argument spjatý s parametrem -w, hodnotu, která následuje tento argument zapíše do souboru csv jako název opravujícího, výchozí hodnota je ‚smrz‘.

Kapitola 9

Webová aplikace

Webová aplikace má striktně vyhrazené použití. Slouží jako nástroj pro studenty. Z toho se odvíjí i její požadavky:

- Musí být konkurenceschopná,
- nesmí docházet k zápisu do souboru, do kterého by mohl zapisovat jiný,
- v jednu chvíli musí být schopna oprav několika různých zadání,
- musí být schopná, kvůli hledání chyb, ukládat jednotlivá použití.

Toto všechno splňuje aplikace obsažená v souboru `flask_app.py`. Tato aplikace je vytvořená za použití frameworku `Flask`[1]. Webová aplikace je poté napojena a spuštěna na webovém serveru `Twisted Web`[2], díky kterému je tato aplikace konkurenceschopná. Vzhledem k minimálním nárokům je tento framework dostačující. Aplikace běží na portu 8088, pokud je vyžadován jiný port, je nutno tuto hodnotu změnit přímo v modulu.

9.1 Konfigurace webové aplikace

Vzhledem k požadavkům musí být aplikace schopná opravy několika různých zadání. Zadávání této skutečnosti při spouštění aplikace z konzolové řádky by bylo pracné a nepřehledné, z tohoto důvodu je předsazen konfigurační soubor v syntaxi YAML. Tento konfigurační soubor se skládá z seznamu jednotlivých zadání oprav. V každé takovéto části jsou parametry nezbytné pro inicializace opravy a to:

- Seznam URL, na kterých bude oprava dostupná – doporučuje se, aby jedné opravě bylo přiřazeno prázdné URL, tedy '/',
- nadpis,
- seznam dovolených přípon při nahrávání souboru,
- cesta ke konfiguračnímu souboru zadání v syntaxi YAML.

[1] Flask http://flask.pocoo.org/
[2] Twisted Web http://twistedmatrix.com/trac/wiki/TwistedWeb

Výčet 9.1 Konfigurační soubor webové aplikace

```
proj1:
    URL: ['', 'proj1']
    ALLOWED_EXTENSIONS: ['py', 'zip']
    TITLE: 'První projekt'
    TASK: 'task1.yaml'
proj2:
    URL: 'proj2'
    ALLOWED_EXTENSIONS: 'py'
    TITLE: 'Druhý projekt'
    TASK: 'task2.yaml'
```

9.2 Spuštění webové aplikace

Webová aplikace se spouští z konzolové řádky se stejnými požadavky na překladač, jako je tomu u aplikace konzolové. Liší se v spouštěném scriptu a také v argumentech tohoto scriptu. Script s názvem `flask_app.py` má tyto parametry:

- -t, - -task – povinný parametr, jeho hodnota představuje cestu k konfiguračnímu souboru webové aplikace,
- -l, - -logging – příznak logování, logován je studenty odevzdaný soubor a také výsledek kontroly.

9.3 Průběh webové kontroly

Průběh webové kontroly začíná stránkou, sloužící jako vstup aplikace. Stránka je přístupový bod pro studenty, zde mohou zkušebně odevzdat své řešení a zjistit své ohodnocení. Nachází se na jedné z url oprav a její šablona je definována v souboru `upload.html`. Nabízí grafické rozhraní, do kterého je nutno zadat přihlašovací jméno a nahrát soubor, po němž je požadovaná oprava (viz příloha C.1). Po splnění těchto dvou akcí proběhne, na straně klienta, kontrola správnosti přihlašovacího jména a zda má nahraný soubor dovolenou příponu. Před provedení opravy je stejná kontrola i na straně serveru, pro případ že by měl klient zakázaný javascript.

Jakmile je kontrola hotova, je student přesměrován na url `'/result'` a zároveň, pro účely možnosti aktualizace stránky bez ztráty výsledku kontroly, je výsledek opravy uložen do relace klienta. Tohoto je docíleno pomocí balíčku Flask-Session[3], externího balíčku je použito kvůli omezení maximální velikosti relace poskytované frameworkem Flask, ukládající relace do cookies klienta[4]. Kromě velikosti je zde také omezení v možnosti zakázání cookies na straně klienta.

[3] Flask-Session https://pythonhosted.org/Flask-Session/
[4] Sessions http://flask.pocoo.org/docs/0.12/quickstart/#sessions

Kapitola 10

Testování

Testování je nedílnou součástí životního cyklu každé aplikace. Vzhledem k zaměření tato potřeba pouze nabyla na důležitosti. Aplikaci bude využívána řádově stovkami programátorů s různou úrovní znalostí, od začátečníků až po velmi zkušené jedince. Tito uživatelé mají rozmanité způsoby programování a bude zde docházet k velmi různorodým chybám. Aplikace se s těmito stavy musí vyrovnat. Testování proběhlo ve dvou hlavních fázích, a to s projekty, které byly odevzdány v rámci předmětu Skriptovací jazyky v akademickém roce 2015/2016 (ISJ2016), fáze druhá obsahovala projekty v předmětu Skriptovací jazyky v akademickém roce 2016/2017 (ISJ2017), tedy již byla spuštěna webová aplikace a testováno probíhalo za běhu předmětu.

10.1 Testování s projekty z akademického roku 2015/2016

V raných fázích práce s projekty z předmětu ISJ2016 proběhlo spíše zpřesnění specifikací aplikace a také implementace a otestování základních principů. V rámci tohoto předmětu byli zadány tři projekty. Hlavní testování proběhlo s prvním a druhým projektem. A to z toho důvodu, že byly tyto projekty velmi rozdílné. První projekt obsahoval více souborů, textové soubory a samotný soubor s kódem nebyl zásadní. Naproti tomu druhý projekt obsahoval jediný soubor. Byla v něm obsažena třída a zásadní je testování této třídy.

10.1.1 První projekt

Testování s tímto projektem bylo zaměřeno na opravu jednoho projektu. Vzhledem k struktuře projektu byl ideální na testování kontroly struktury projektu, pojmenování souboru podle přihlašovacího jména studenta a kontrolu souboru oproti vzoru. Jedna z hlavních chyb, která byla při testování odhalena, je skutečnost, že pokud student řádně neuzavře soubor, do kterého zapisuje a následně je tento soubor porovnáván oproti vzoru nastane situace, že v souboru ještě není zapsán veškerý obsah. Toto je vyřešeno seřazením souborů podle typu. Tedy soubory, které neobsahují kód jsou kontrolovány první a nemohou být ovlivněny jejich neuzavřením. Tento přístup zároveň řeší další problém vyskytující se při zapisování do souborů a to je, že pokud soubor již existuje, student zapisuje nový obsah na jeho konec a tudíž je obsah duplikován.

Vzhledem k skutečnosti, že řešení projektu obsahuje několik souborů, bylo toto zadání využito k testování rozbalení archivovaných souborů. Bylo zde zjištěno, že aplikace musí počítat i s velmi neobvyklými pokusy o archivaci, jakož je manuální přejmenování přípony.

V průběhu vývoje aplikace bylo s tímto zadáním testováno ještě opakovaně. Například bodování několika souborů a součet těchto bodů a nahrání archivovaného souboru do webové aplikace.

10.1.2 Druhý projekt

Vzhledem k zadání, kdy je požadován pouze jediný soubor, v němž je obsažena třída, se tento stává ideálním k testování kontrol nad souborem obsahující kód. S těmito projekty byla testována kontrola dokumentačních řetězců, hledání abstraktních struktur v kódu, bodování jednotlivých komponent a další.

Velice významné bylo u tohoto zadání provedení testů nad implementovanou třídou a proto bylo s touto třídou testováno provedení testů nad studentským souborem. Nejprve bylo implementováno vlastní řešení provedení testů. Aby byly testy vykonány nad jednotlivými soubory, byla nejprve změněna fyzická podoba souboru s testy. Jeho první řádek postupně obsahoval import jednotlivých studentských souborů. Následně byl proveden import souboru s testy pomocí knihovny `importlib`. Zde byl dlouho řešen problém s opakovaným importem souboru s testy. Kde první řádek zůstával zastaralý. I pomocí funkce `invalidate_caches` dostupné v této knihovně stále zůstával nahrán původní soubor s neměnným prvním řádkem. Tento problém byl vyřešen dočasnými názvy souborů. Kvůli těmto problémům byl přístup s knihovnou `importlib` opuštěn a povedení testů bylo vykonáno pomocí abstraktního syntaktického stromu. Zde byl první řádek měněn již v tomto stromu a byl bezproblémový. Nakonec bylo i toto řešení nedostatečné a provedení testů dostalo svojí nynější podobu s podporou knihovny `unittest`.

Proběhlo zde také testování hromadných oprav. Zde byla nalezena zásadní chyba vzájemné interference projektů, kdy předchozí kontrola ovlivňovala kontrolu budoucí. Toto bylo vyřešeno tím, že kontrole projektů jsou dodány hluboké kopie všech struktur.

10.2 Testování v rámci akademického roku 2015/2016

Další fáze testování, kdy spíše než k vývoji docházelo k opravě nedostatků a zahrnutí stavů se kterými nebylo počítáno, proběhla v rámci předmětu ISJ2017. Vzhledem k tomu, že návrhy zadání již počítaly s automatickou opravou, tudíž je v nich kladen důraz na mírně odlišné skutečnosti než v roce 2016, projektů je více a jsou méně komplexní, zadání je co nejjednoznačnější a hlavní částí bodování je průběh testů. Po dokončení hromadné opravy probíhalo porovnání bodování se systémem, jehož vývoj probíhal souběžně.

10.2.1 První projekt

Povinnost studentů v rámci prvního projektu bylo doplnit řešení několika řádků do již připraveného kódu tak, aby vložené příkazy `assert` nevyvolaly výjimku. Kód byl rozdělen do tří funkcí a volání těchto funkcí bylo součástí kódu. Toto spojení výrazu `assert` a volání bloku kódu v němž je obsažen bylo hlavní chybou, která byla objevena v rámci prvního projektu. U prvních oprav projektů, které nebyly zcela správně, ukončil tento `assert` opravu a vzhledem k tomu, že odchycení této výjimky nebylo součástí záznamu výjimek a skutečnosti, že se body strhávají za chybu, udělen plný počet bodů. Ani po opravě zachycení této výjimky nebyl výsledek oprav uspokojivý. Aplikace hodnotila projekty nulou nebo plným počtem bodů. A to z důvodu, že pokud nebyl správně minimálně jeden úkol, tak tento kód vyvolal `assert`.

U výsledného hodnocení tedy musely být nalezeny projekty s nulovým ziskem bodů a manuálně zjištěno zda není hotový některý z úkolů.

Zároveň u tohoto druhu zadání vyvstala potřeba porovnávání souboru s kódem oproti vzoru.

Tabulka 10.1 ukazuje rozdíl v bodovém ohodnocení dvou systémů. Rozdíly je možno rozdělit do dvou skupin. První méně početná skupina jsou studenti, jejichž projekt, které systém vyvinutý v rámci bakalářské práce špatně vyhodnotil, jako zcela správné. Konkrétně se jednalo o první úkol, v němž měl student vyjmout z pole ty prvky, jež porušují podmínku. Osm řešení obsahovalo prázdné pole, které sice podmínku splnilo, ale bylo v rozporu se zadáním. Druhá skupina je drobná chyba druhého systému, jež ohodnotil nulovým počtem bodů studenty, kteří projekt neodevzdali.

Počet projektů	Systém1	Systém2
8	5b	4b
15	nehodnoceno	0b

Tabulka 10.1: Porovnání bodování prvního projektu. Systém1 je systém vyvinutý v rámci bakalářské práce.

10.2.2 Druhý projekt

Druhý projekt se již vyvaroval chyb zadání a byly zde opraveny pouze drobné chyby jako byla studentem vyvolaná výjimka. Takto vyvolaná výjimka je odlišná v tom, že nemusí mít tělo.

U tohoto projektu bylo zjištěno, že webová aplikace bude muset být schopná obsluhovat několik druhů zadání najednou a to kvůli tomu, že datum konce odevzdávání prvního projektu bylo ve stejný den jako zahájení.

Porovnání hodnocení dvou systému odhalilo pouze jedno řešení, ve kterém se neshodují. Manuální kontrolou bylo zjištěno, že měl pravdu systém popsaný v technické zprávě.

10.2.3 Třetí projekt

Zde docházelo k drobným opravám bodování. Byl zde také opraven pád aplikace v případě, že student dodal soubor v jiném kódování, než je utf-8.

Při porovnání hodnocení bylo nalezeno, že jeden z úkolů není pro automatické opravování vhodný. Jednalo se o úpravu objektů seznam tak, aby byla možnost jejich použití jako klíč slovníku. 50 studentů použilo řešení, které bylo nekonzistentní a zhruba v 1 z 20 pokusů selhalo. V těchto řešení byla použita metoda `__str__` nebo `__repr__` samotného seznamu k získání jednoznačného řetězce k použití jako hashe. Vzhledem k tomuto faktu a také skutečnosti, že kromě těchto projektů bylo bodování totožné, nemělo další porovnávání smysl.

10.2.4 Čtvrtý projekt

Zde bylo zadání již odlišné. Student musí vytvořit celý obsah souboru. Vyvstala zde potřeba kontroly pouze nějakého typu dokumentačních řetězců a to z důvodu, že modul obsahoval tři odlišné funkce a tudíž modul samotný neplnil žádnou důležitou funkci, tudíž jeho dokumentační řetězec nedával velký smysl.

Porovnání hodnocení systémů (viz tabulka 10.2) odhalilo velké rozdíly v hodnocení, toto bylo očekávané, vzhledem k různým testovým sadám. Rozdíl bodování dvou největších skupin s celkovým počtem 30 studentů byl způsobem v pojetí testování prvního úkolu. Systém popsaný v práci měl pouze testy, jež byly již popsány v zadání, naproti tomu druhý systém rozdělil první úkol mezi dvě testové metody, kde každá byla ohodnocena jedním bodem. Druhá testová metoda obsahovala náročnější testy a proto část projektů na této metodě selhalo. Dále je v tabulce viděna skupina o pěti projektech. Zde byl naopak méně benevolentní první systém. Předposlední skupina rozdílů je způsobena odlišnou množinou testů. Poslední řádek, kde došlo k nárůstu z nuly na bodový zisk, byl v jednom případu způsoben zablokováním vstupu, který projekt očekával a ve druhém odhalením syntaktické chyby, vzniklé úpravou souboru na poslední chvíli.

Počet projektů	Systém1	Systém2
20	2b	1b
10	4b	3b
5	2b	5b
2	2b	3b
2	2b	0b

Tabulka 10.2: Porovnání bodování čtvrtého projektu. Systém1 je systém vyvinutý v rámci bakalářské práce.

10.2.5 Pátý projekt

Vzhledem ke skutečnosti, že zadání bylo obdobné druhému projektu minulého roku, s kterým probíhal vývoj aplikace, nenastaly zde žádné problémy se samotnou aplikací. Jediná obtíž nastala se skutečností, že testovací případy byly v těle metody, která byla vyžadována v rámci zdrojového souboru. S tímto přístupem vzniklo hned několik problémů. Výsledný kód byl hodnocen pouze plným počtem nebo nulou. Další problém byl se snahou o podvod ze strany studentů, vzhledem k tomu, že aplikace nenabízí žádný nástroj, jak zkontrolovat zda tělo funkce je nezměněné, smazali problémové testy a byli ohodnoceni plným počtem. Přesunem testů na stranu serveru bylo dosaženo lepší distribuce bodů a také zabráněno podvodům.

10.2.6 Šestý projekt

Projekt se skládal ze dvou úkolů. Jediný problém byl se vznikem testů k úkolu druhému. Úkol vycházel z logické hádanky, kdy s pomocí kombinace čtyř čísel a aritmetických operací má vyjít číslo páté[1]. Problém vznikl z toho, že každé jedno z řešení, jež je validní, musí být uznáno s tím, že studentův projekt může vracet jakoukoli podmnožinu všech řešení. Jediná podmínka na počet řešení nastane, pokud se výsledku lze dopočítat více různými výrazy. V řešení poté musí být přítomen alespoň jeden výraz z každé možné kombinace. Pomocí zpětné vazby v podobě fóra se tyto testy odladily a při opravovování již nenastal žádný problém.

[1] Combine4 http://blog.plover.com/math/17-puzzle.html

10.2.7 Sedmý projekt

Průběh sedmého projektu byl bezproblémový. Drobný nedostatek byl v testování kontextového manažera, kde byl striktně vyžadován znak nového řádku podle systému Unix, tedy `LF` a jeden projekt použil znak nového řádku dle operačního systému. Vzhledem k tomu, že hromadná oprava proběhla na systému Windows, tak mu test neprošel. Kód musel být manuálně upraven.

10.2.8 Osmý projekt

První z projektů v akademickém roce 2016/2017 s více soubory. Zadání bylo totožné s třetím projektem v akademickém roce 2015/2016. Vyvstala zde potřeba provedení unit testů nad souborem, který nebude importován, tedy kombinace, jež nebyla možná. Důvod byl úkol, jehož řešení spělo v přechodu z `treading`[2] na jiný způsob paralelismu. Při importu tohoto druhu souboru musí být paralelní část v bloku `main`[3], jinak vznikne zacyklení aplikace. Pro velké množství řešení z akademického roku 2015/2016, které paralelní část nemělo v bloku `main`, byla implementována kombinace provedení unit testů bez importu souboru.

10.3 Unit testy

V rámci vývoje a ověřování správnosti aplikace vznikly pro některé části kódu unit testy. Testů je přes 80 a pokrývají:

- Zpracování konfiguračního souboru,
- rozbalení archivů,
- kontrolu struktury projektu,
- kontrolu obsahu souboru bez kódu,
- kontrolu struktury souboru s kódem,
- hledání funkcí `exec`/`eval` bez konstrukce `try`,
- hledání zpracování řetězců, které by mohli být zapsány funkcí `join`,
- udělení bodů za jednotlivý soubor i za celý projekt,
- kontrolu shebang,
- kontrolu dokumentačních řetězců.

[2] threading https://docs.python.org/3/library/threading.html
[3] main https://docs.python.org/3/library/__main__.html

Kapitola 11

Závěr

V rámci této práce jsem nejprve prostudoval chyby, kterých se dopouštějí začínající programátoři. Nejprve jsem provedl výzkum chyb nezávislých na konkrétním programovacím jazyce. Poté jsem provedl rozbor projektů odevzdaných studenty v předmětu Skriptovací jazyky (ISJ) v akademických rocích 2015/2016 a 2016/2017. Mohu konstatovat, že některé chyby, například chybné pojmenování souboru, se podařilo odstranit pomocí webové aplikace. Naopak chyby dokumentačních řetězců stále přetrvávaly i v akademickém roce 2016/2017. Důvod může být nepenalizování chybějících dokumentačních řetězců.

Poté jsem prostudoval již existující řešení v oblasti automatických vyhodnocování kódu a podpůrných systémů obecně. Zaměřil jsem se pouze na systémy, jež jsou schopny běžet ve webovém prostředí. Byli rozebrány jejich přednosti a naopak oblasti, ve kterých zaostávají. Smyslem tohoto zkoumání nebyla snaha o nalezení vyhovující aplikace, která by mohla být nasazena v předmětu ISJ, ale spíš inspirace při vývoje systému, o němž tato práce pojednává.

Následně jsem se již zaměřil na popis aplikace. Při popisu jsem důkladně probral třídní návrh a rozhraní jednotlivých abstraktních tříd. Toto bylo z důvodu pozdější snahy o rozšíření systému, které nemusí vzejít pouze z mé strany. V rámci popisu jsem probral i celkové fungování aplikace, definici vstupu, její výstupy a také její možné formy (webová a konzolová).

Poté jsem popsal průběh testování s projekty z předmětu ISJ. V první sadě testů jsem se zaměřil na práci s projekty v akademickém roce 2015/2016, na základě těchto testů vznikaly samotné specifikace aplikace a byli použity při dokazování takové míry správnosti, aby jsem mohl přejít k druhé fázi testování. To probíhalo s projekty v akademickém roce 2016/2017 a již se jednalo o ostré nasazení, k webové aplikaci již měli přístup studenti a ohodnocení projektů ovlivňovalo výsledné hodnocení projektů. Hodnocení projektů jsem porovnával s druhým řešením ‚jehož implementace pobíhala souběžně s mým, zde mohu konstatovat, že výsledky byly podobné a jejich rozdíly byly způsobeny lidským faktorem (rozdílné sady testů, drobná oprava studentského kódu atd.). Zde již docházelo k chybám výjimečně. Na základě testování v akademickém roce 2016/2017 mohu konstatovat, že nejdůležitější pro správné použití a přehledný výstup aplikace jsou testovací třídy a testy v nich obsažené. V několika případech byli testové případy nevhodně navrženy:

- Maximální počet bodů, či nulový zisk,
- nevhodná distribuce bodů,

- nevhodné použití `assert`, který měl za následek neprovedení kontroly, pokud nebylo řešení zcela v pořádku,
- chyba v testování v rané fázi spuštění opravy. Tento problém obzvlášť vynikne v kombinaci s přístupem studentů k webové aplikaci a k odevzdávání obecně. K valné většině zkoušení ve webové aplikaci docházelo v den konce odevzdávání, tudíž mohly být problémy v testování odhaleny až v tento den.

Tyto problémy, vzhledem k tomu, že testování vychází ze standardních unit testů, mohou být způsobeny mnou, jakožto nezkušeným instruktorem. Vzhledem k minimu chyb, jež v pozdějších fázích testování nastaly a k přihlédnutí k výsledkům které aplikace dodala, mohu konstatovat, že systém je možno použít v reálném prostředí.

Aplikaci by dále bylo možno rozšířit tak, aby brala v potaz plagiátorství. Tímto tématem, vzhledem ke svému rozsahu, jsem se v rámci bakalářské práce nezabýval. Dále je zde možnost přidání hledání dalším nevhodných praktik nebo rozšířit možnou definici abstraktní struktury hledané ve studentském kódu. Vzhledem ke vstupu, který je pouze textový v podobě konfiguračního souboru, se nabízí možnost tomuto souboru předsadit grafické rozhraní, například v podobě webové stránky. K přihlédnutí k faktu, že s tímto souborem pracuje pouze instruktor, jsem se tímto rozšířením nezabýval.

Literatura

[1] *Účty a bezpečnost dat : Studentské účty* [online]. Božetěchova 1/2 612 66 Brno, Czech Republic: Fakulta informačních technologií VUT v Brně [cit. 2017-04-18]. Dostupné z: <http://www.fit.vutbr.cz/CVT/net/ucty.php.cs>.

[2] *Flexible Box Layout Module* [online]. Alexis Deveria, 2013 [cit. 2017-04-18]. Dostupné z: <http://caniuse.com/#feat=flexbox>.

[3] *Java plug-in does not work in Firefox after installing Java* [online]. Oracle, 2017 [cit. 2017-04-18]. Dostupné z: <https://www.java.com/en/download/help/firefox_java.xml>.

[4] *Comma-separated values* [online]. San Francisco (CA): Wikimedia Foundation, 3 March 2004, 2 April 2017 [cit. 2017-04-18]. Dostupné z: <https://en.wikipedia.org/wiki/Comma-separated_values>.

[5] *Bloom's taxonomy* [online]. San Francisco (CA): Wikimedia Foundation, 4 July 2003, 15 April 2017 [cit. 2017-04-18]. Dostupné z: <https://en.wikipedia.org/wiki/Bloom%27s_taxonomy>.

[6] *Regular expression : Perl* [online]. San Francisco (CA): Wikimedia Foundation, 4 September 2001, 13 April 2017 [cit. 2017-04-18]. Dostupné z: <https://en.wikipedia.org/wiki/Regular_expression#Perl>.

[7] *Self-efficacy* [online]. San Francisco (CA): Wikimedia Foundation, 7 September 2005, 23 March 2017 [cit. 2017-04-14]. Dostupné z: <https://en.wikipedia.org/wiki/Self-efficacy>.

[8] *Python 3.6.1 documentation* [online]. Python Software Foundation, c2001-2017 [cit. 2017-04-18]. Dostupné z: <https://docs.python.org/3/>.

[9] *Python decorator with multiprocessing fails* [online]. Stack Exchange, Feb 17 '12 at 22:59, Feb 17 '12 at 23:15 [cit. 2017-04-18]. Dostupné z: <http://stackoverflow.com/questions/9336646/python-decorator-with-multiprocessing-fails>.

[10] Barr, V.; Trytten, D. : Using turing's craft codelab to support CS1 students as they learn to program. *ACM Inroads*, roč. 7, č. 2, 16.5.2016: s 67–75, ISSN 21532184, 10.1145/2903724. Dostupné z: <http://dl.acm.org/citation.cfm?doid=2938622.2903724>

[11] Bonar, J.; Soloway, E. : Preprogramming Knowledge. *Human–Computer Interaction*, roč. 1, č. 2, 1985: s 133–161, ISSN 0737-0024, 10.1207/s15327051hci0102_3. Dostupné z: <http://www.tandfonline.com/doi/abs/10.1207/s15327051hci0102_3>

[12] Chickering, A. W.; Gamson, Z. F. : Seven principles for good practice in undergraduate education. *Biochemical Education*, roč. vol. 17, č. issue 3, 1989: s 140–141, ISSN 03074412, 10.1016/0307-4412(89)90094-0. Dostupné z: <http://linkinghub.elsevier.com/retrieve/pii/0307441289900940>

[13] Cooper, M. *Advanced Bash-Scripting Guide : Chapter 2. Starting Off With a Sha-Bang* [online]. The Linux Documentation Project, 05 Apr 2012, 10 Mar 2014 [cit. 2017-04-18]. Dostupné z: <http://tldp.org/LDP/abs/html/sha-bang.html>.

[14] Coyier, C. *A Complete Guide to Flexbox* [online]. CSS-TRICKS, 2013, 2017-3-21 [cit. 2017-04-18]. Dostupné z: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.

[15] David Goodger, G. v. R. *PEP 257 : Docstring Conventions* [online]. Beaverton, OR 97008: Python Software Foundation, 29-May-2001, 13-Jun-2001 [cit. 2017-04-18]. Dostupné z: <https://www.python.org/dev/peps/pep-0257/>.

[16] Dihoff, R. E.; Brosvic, G. M.; Epstein, M. L.; aj. : Provision of Feedback during Preparation for Academic Testing. *Psychological Record*, roč. 54, č. 2, 2004: s 207–231, ISSN 0033-2933.

[17] Edgcomb, A. D.; Vahid, F. Effectiveness of Online Textbooks vs. Interactive Web-Native Content. *2014 ASEE Annual Conference & Exposition*. 2014. Dostupné z: <https://peer.asee.org/20351>. ISSN 2153-5965.

[18] Freeman, E.; Robson, E.; Sierra, K.; aj. *Head First design patterns*. Sebastopol, CA: O'Reilly, c2004. ISBN 05-960-0712-4.

[19] Guo, P. J. Online python tutor. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. 2013-03-06, s. 579-584. Dostupné z: <http://dl.acm.org/citation.cfm?doid=2445196.2445368>. 10.1145/2445196.2445368.

[20] Honzík, J. M. *Algoritmy : Studijní opora*. 14 n. VUT v Brně, 2014. Dostupné z: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIAL-IT%2Flectures%2FOpora-IAL-2014-verze-14-N.pdf&cid=9958>.

[21] Johann C. Rocholl, I. L., Florent Xicluna. *Pep8's documentation* [online]. Read the Docs, March 17, 2015, Feb. 24, 2017 [cit. 2017-04-18]. Dostupné z: <http://pep8.readthedocs.io/en/release-1.7.x/>.

[22] Kluyver, T. *Green Tree Snakes : the missing Python AST docs* [online]. Read the Docs, June 11, 2015, Feb. 17, 2017 [cit. 2017-04-18]. Dostupné z: <https://greentreesnakes.readthedocs.io/en/latest/>.

[23] Kohn, T. *Teaching Python Programming to Novices : Addressing Misconceptions and Creating a Development Environment* Zürich, 2017.

[24] Kreen, M. *Rarfile - RAR archive reader for Python* [online]. Read the Docs, April 29, 2015, April 4, 2017 [cit. 2017-04-18]. Dostupné z: <https://rarfile.readthedocs.io/en/latest/>.

[25] Kreslíková, J. *Rekurzivní metody v programování*. Brno, c2015. Dostupné z: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIZP-IT%2Ftexts%2F2016%2F12t_rekurze.pdf>.

[26] Kumar, A. N. The effect of using online tutors on the self-efficacy of learners. *2015 IEEE Frontiers in Education Conference (FIE)*. 2015, s. 1-7. Dostupné z: <http://ieeexplore.ieee.org/document/7344117/>. 10.1109/FIE.2015.7344117.

[27] Křívka, Z.; Kolář, D. *Principy programovacích jazyků a objektově orientovaného programování*. 1.1. VUT v Brně, 2008. Dostupné z: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIPP-IT%2Ftexts%2FIPP-II-ESF-1_1_printable.pdf&cid=9999>.

[28] Lutz, M. *Learning Python*. 5th ed. Beijing: O'Reilly Media, 2013. ISBN 9781449355739.

[29] Mascheck, S. *The #! magic, details about the shebang/hash-bang mechanism on various Unix flavours : Selected issues* [online]. 2001-08-13, 2017-03-28 [cit. 2017-04-19]. Dostupné z: <https://www.in-ulm.de/~mascheck/various/shebang/#details>.

[30] McLaughlin, B.; Pollice, G.; West, D. *Head first object-oriented analysis and design*. 1st ed. Sebastopol: O'Reilly, c2007. ISBN 05-960-0867-8.

[31] Pea, R. D. : Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of Educational Computing Research*, roč. 2, č. 1, 1.2.1986: s 25–36, 10.2190/689T-1R2A-X4W4-29J2. Dostupné z: <http://journals.sagepub.com/doi/10.2190/689T-1R2A-X4W4-29J2>

[32] Guido van Rossum, L. L., Jukka Lehtosalo. *PEP 484 : Type Hints* [online]. Beaverton, OR 97008: Python Software Foundation, 29-Sep-2014, 22-May-2015 [cit. 2017-04-19]. Dostupné z: <https://www.python.org/dev/peps/pep-0484/>.

[33] Guido van Rossum, N. C., Barry Warsaw. *PEP 8 : Style Guide for Python Code* [online]. Beaverton, OR 97008: Python Software Foundation, 05-Jul-2001, 01-Aug-2013 [cit. 2017-04-18]. Dostupné z: <https://www.python.org/dev/peps/pep-0008/>.

[34] Ryan Gonzalez, I. L. L. R. G. v. R., Philip House. *PEP 526 : Syntax for Variable Annotations* [online]. Beaverton, OR 97008: Python Software Foundation, 09-Aug-2016, 02-Sep-2016 [cit. 2017-04-19]. Dostupné z: <https://www.python.org/dev/peps/pep-0526/>.

[35] Spohrer, J. C.; Soloway, E. : Novice mistakes. *Communications of the ACM*, roč. 29, č. 7, 1986: s 624–632, ISSN 00010782, 10.1145/6138.6145. Dostupné z: <http://portal.acm.org/citation.cfm?doid=6138.6145>

[36] Summerfield, M.; Krejčí, L. *Python 3 : výukový kurz*. Vyd. 1. Brno: Computer Press, 2010. ISBN 9788025127377.

[37] Vojnar, T. *Synchronizace procesů*. VUT v Brně, 2017. Dostupné z: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIOS-IT%2Flectures%2Fios-prednaska-06.pdf&cid=9378>.

[38] Wang, S.-L.; Wu, P.-Y. : The role of feedback and self-efficacy on web-based learning. *Computers & Education*, roč. vol. 51, č. issue 4, 2008: s 1589–1598, ISSN 03601315, 10.1016/j.compedu.2008.03.004. Dostupné z: <http://linkinghub.elsevier.com/retrieve/pii/S036013150800050X>

Přílohy

Příloha A

Seznam možných konstant při vytváření abstraktní struktury

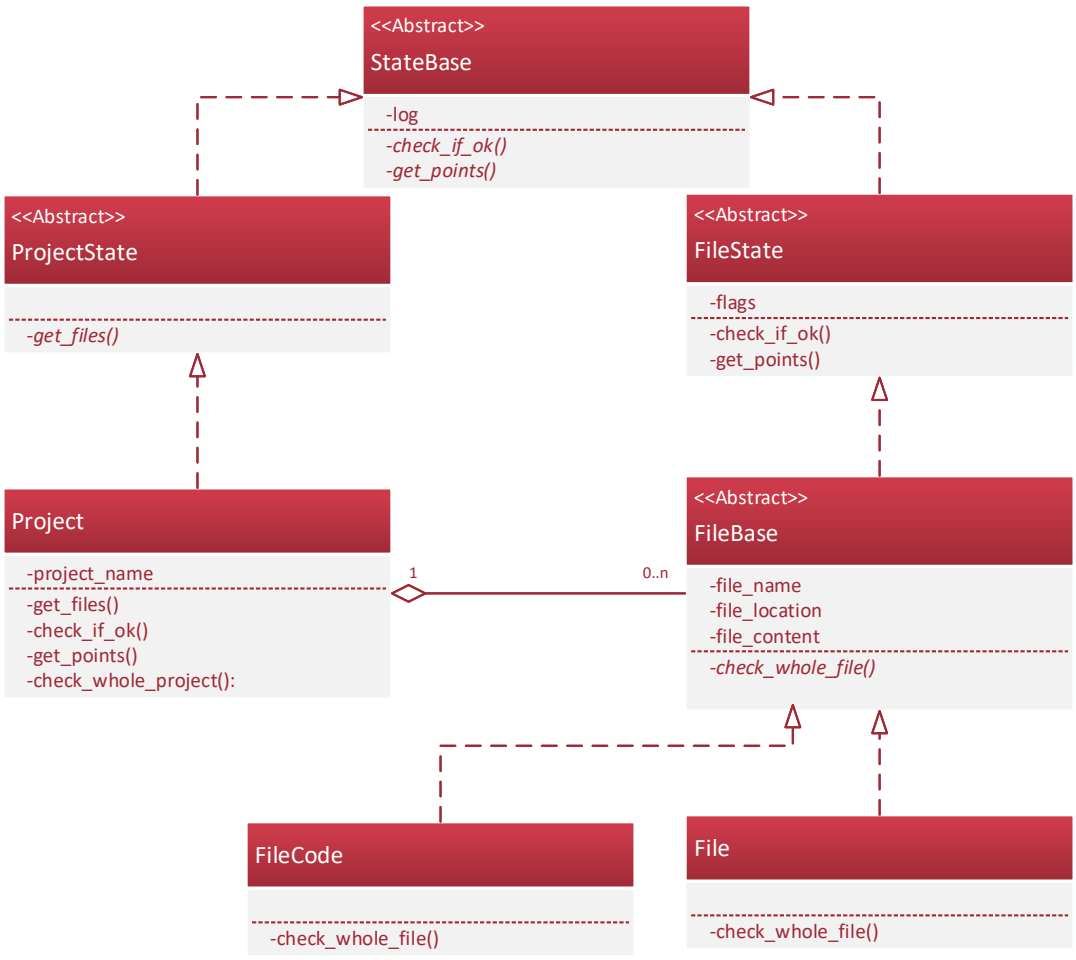
V této příloze budou popsány konstanty, které je možno použít při definici abstraktního stromu v souboru YAML.

- `FUNCTION` - popisuje funkci, u definice se může nacházet i konstanta `NAME`,
- `CLASS` - popisuje třídu, u definice se může nacházet i konstanta `NAME`,
- `WHILE` - popisuje cyklus `while`,
- `LIST_COMP` - popisuje generování seznamu,
- `SET_COMP` - popisuje generování množiny,
- `GEN_EXP` - popisuje generátor,
- `DICT_COMP` - popisuje generování slovníku,
- `CALL` - popisuje volání funkce, u definice se může nacházet i konstanta `NAME`, která označuje název volané funkce,
- `TRY` - popisuje blok `try`,
- `IF` - popisuje podmínku `if`,
- `FOR` - popisuje cyklus `for`,
- `WITH` - popisuje blok `with`.

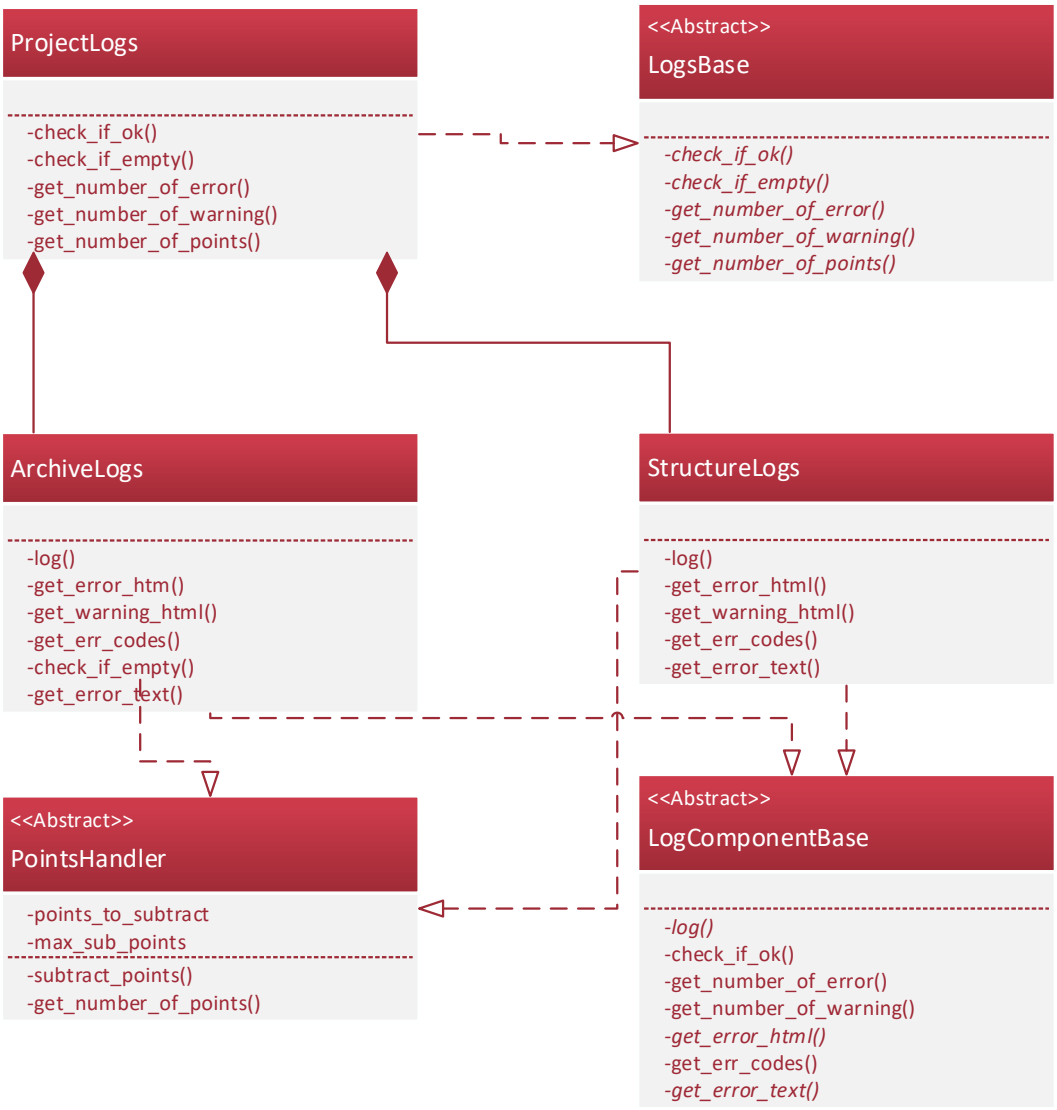
Příloha B

Třídní návrh

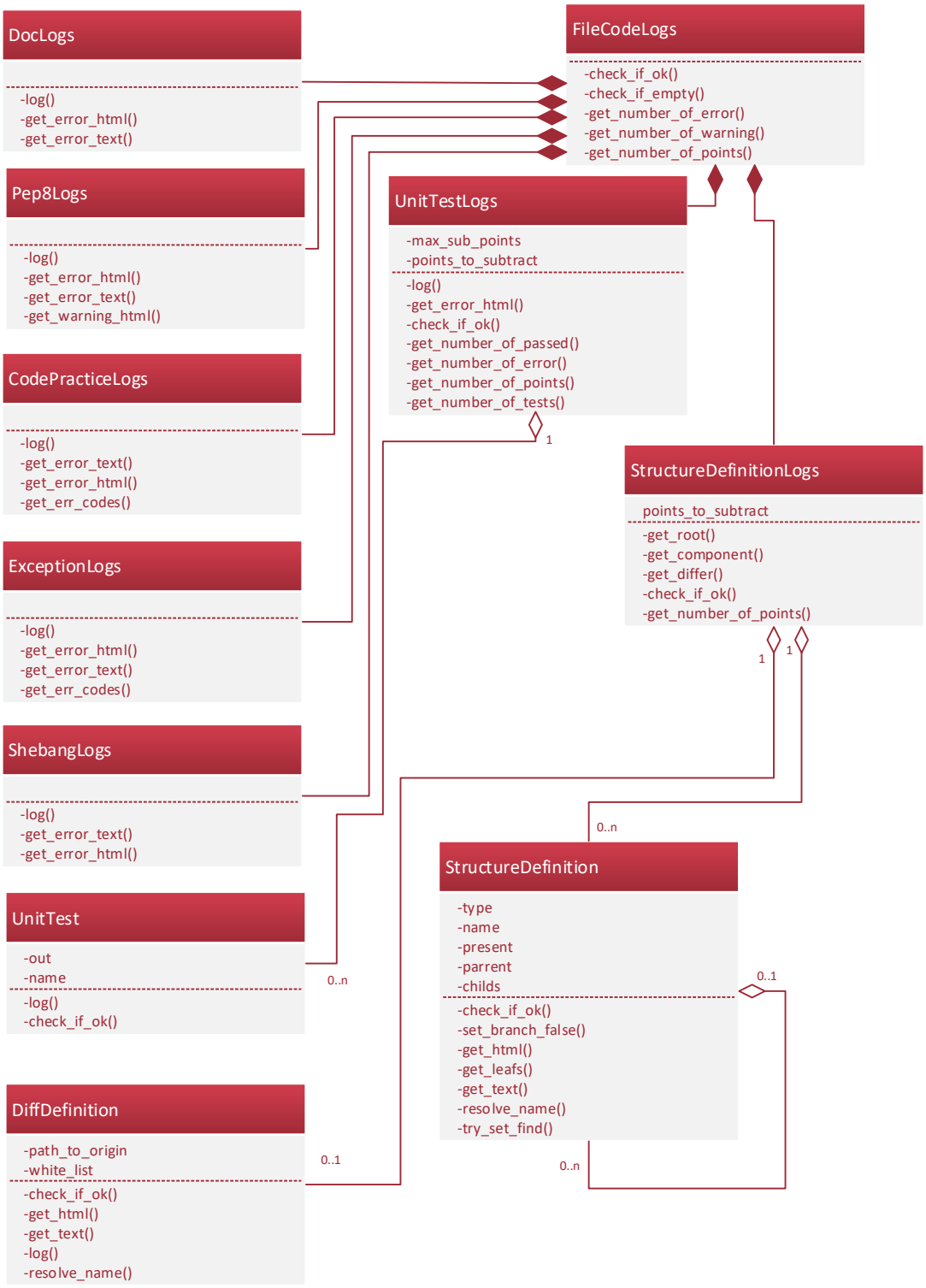
Součástí této přílohy budou jednotlivé detaily třídního návrhu aplikace.



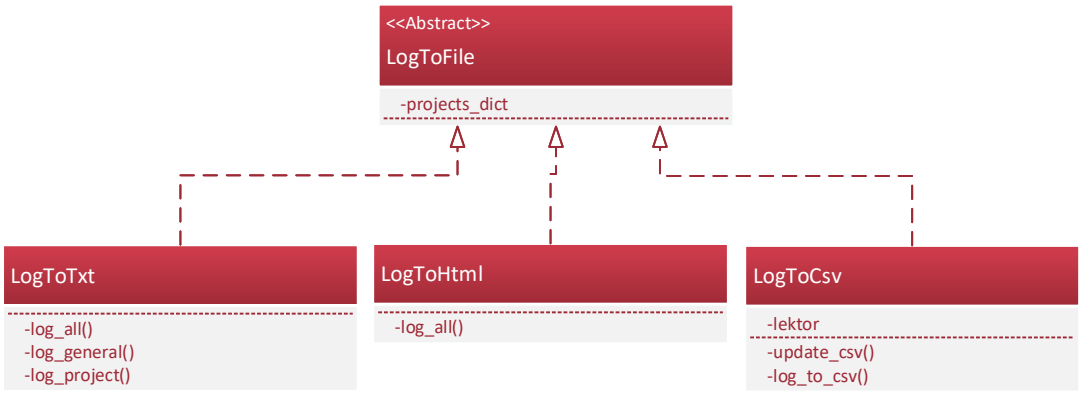
Obrázek B.1: Třídní návrh třídy Project



Obrázek B.2: Třídní návrh záznamu pro třídu Project



Obrázek B.3: Třídní návrh záznamu pro třídu FileCode, kvůli přehlednosti nejsou uvedeny abstraktní třídy.

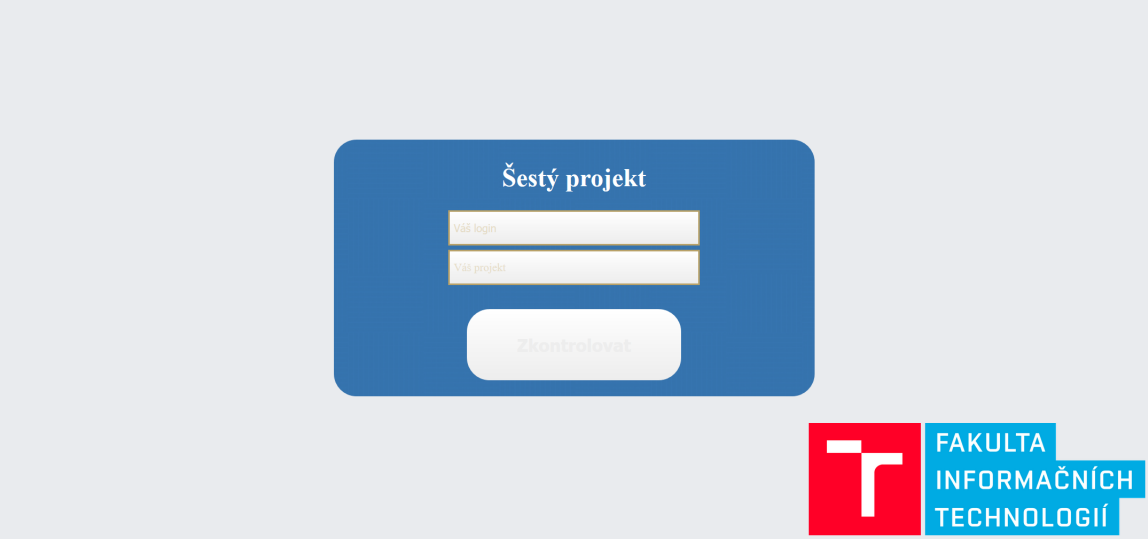


Obrázek B.4: Třídní návrh záznamu do souboru

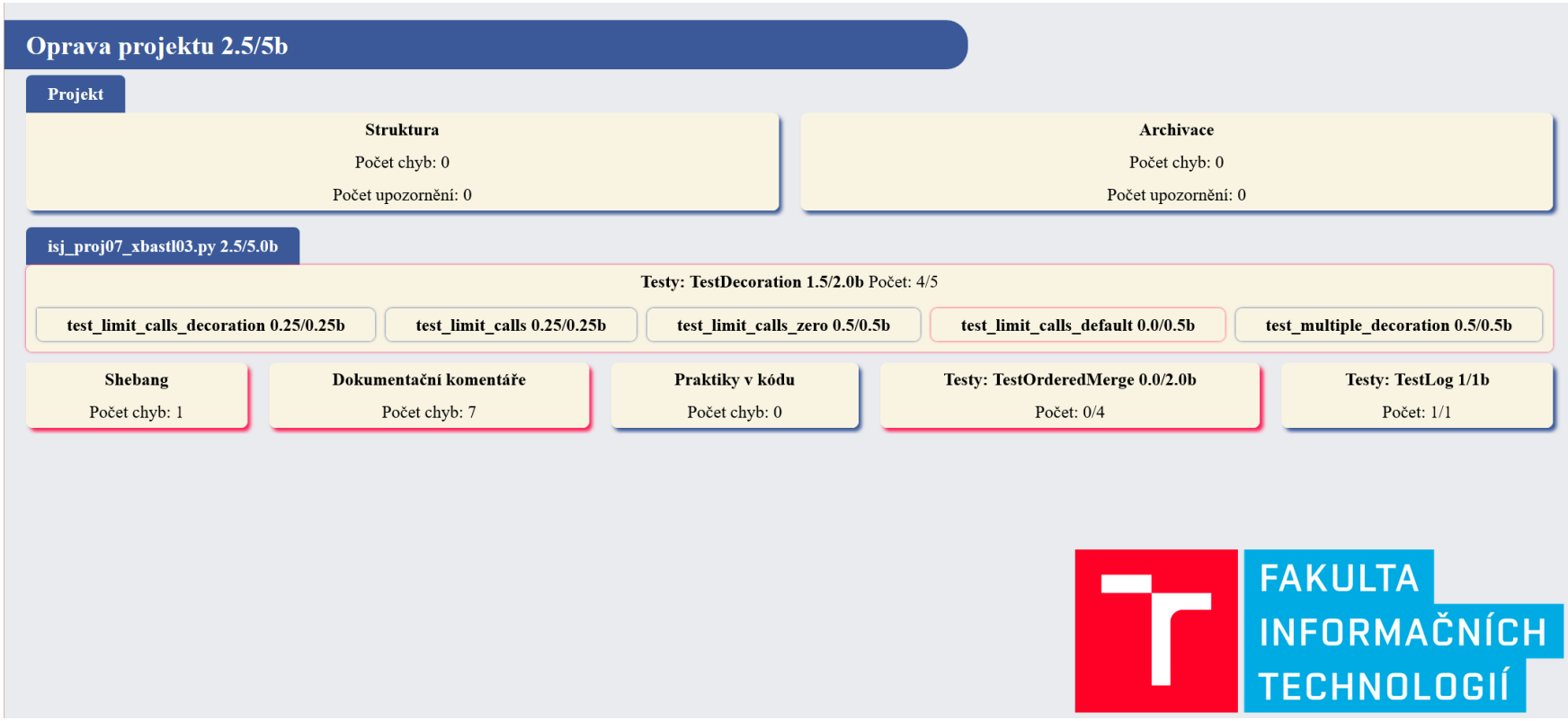
Příloha C

Grafické rozhraní

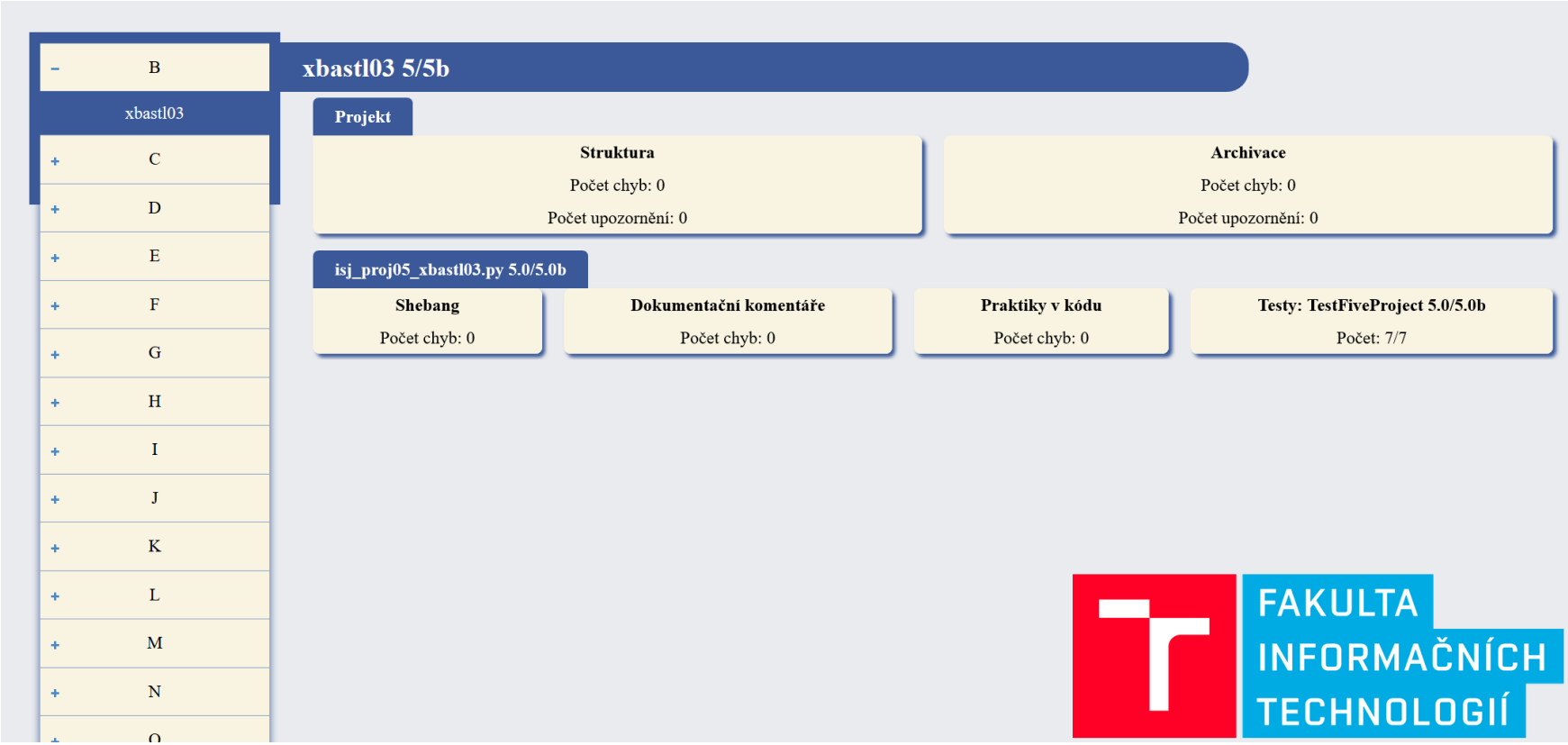
Obsah této přílohy je zaměřen na grafický vstup a výstup aplikace



Obrázek C.1: Webové rozhraní aplikace



Obrázek C.2: Výsledek opravy sedmého projektu v ISJ2017, za pomocí šablony `log_output_one.html`



Obrázek C.3: Výsledek hromadné opravy pátého projektu v ISJ2017, za pomocí šablony `log_output_all.html`