Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА СИСТЕМИ ДЛЯ ЕВРИСТИЧНОГО РОЗПОДІЛУ ПРОЦЕСУ ПЕРЕВІРКИ РОБІТ ЗА РЕЙТИНГОВИМ ПОКАЗНИКОМ

Текстова частина до курсової роботи

за спеціальністю „Інженерія Програмного Забезпечення”

Керівник курсової роботи
к.т.н., доц. _____
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.

Виконав студент _____
_____
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра Мережних технолошій факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доц., к.ф-м.н.
_____Малашонок Г.І.
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Петренко М.В. факультету інформатики 4-го курсу

ТЕМА Розробка системи для евристичного розподілу процесу перевірки робіт за рейтинговим показником

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Анотація

Вступ

Теорія про предметну область

Опис використаних технологій

Аналіз реалізації застосунку

Висновки

Список використаної літератури

Додатки

Дата видачі „___” _____ 2020 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання курсової роботи

Тема: Розробка системи для евристичного розподілу процесу перевірки робіт за рейтинговим показником

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту	Термін виконання етапу	Примітка
1.	Запис на курсову роботу	14.11.2019	
2.	Отримав вимоги до виконання від наукового керівника	18.12.2019	
3.	Розпочав практичну частино курсової (код)	21.02.2020	
4.	Перегляд структури джерела інформації та документації ресурсів для використання	01.03.2020	
5.	Розбиття на під задачі	02.03.2020	
6.	Тестування чорнового варіанту програми на під структурі	04.03.2020	
7.	Поетапне нарощування функціоналу на скелет програми	05.03.2020 - 20.03.2020	
8.	Рефакторинг, оформлення виводу програми	21.03.2020	
9.	Закінчив написання практичної частини	23.04.2020	
10.	Розпочав написання текстової частини	9.04.2020	
11.	Закінчив написання текстової частини	19.04.2020	

Студент Петренко М. В.

Керівник Франчук О. В.

“_____”_____

Зміст

Анотація

У роботі розглянуто методи оптимізації централізованої перевірки виконаної роботи; принципи роботи систем з розподілення навантаження; специфіка роботи протоколів HTTP, FTP; фреймворку для веб застосунків на мові Python Django; API сервісів для перевірки на плагіат, та правопису.

Розроблено програму (веб застосунок) для автоматичного збору файлів з виконаною роботою (домашнє завдання, код для перевірки, тощо), для частки з них – подальшого розподілу між самими авторами в залежності від рейтингів авторів та рецензентів, перевірки робіт на плагіат та помилки системою та додаткової верифікації перевірених авторами робіт від експертів (викладачі, керівники).

Для стимуляції та заохочення авторів до рецензування відкалібрований після перевірки роботи рейтинг може бути використаний для покращення оцінки роботи самого рецензента (яку рецензент здав, а не перевірив), підвищення оцінки у випадку студентів, та преміального заохочення у випадку розробників.

ВСТУП

Актуальність теми:

На сьогоднішній день питання збору та подальшого оцінювання робіт виконаних студентами, учнями, розробниками, програмами, алгоритмами машинного навчання, та взагалі, будь-якими одиницями або системами, результат роботи котрих доводиться в подальшому перевіряти (надалі – “автори”) є дуже актуальним.

Оцінювання робіт: програм, текстів, наукових робіт зараз виконується здебільшого централізовано: викладачами, технічними лідерами та комісіями (надалі – “експерти”). При цьому виникає проблема, де із зростанням розміру оцінюваної вибірки людей (групи, класу) лінійно зростає навантаження на перевіряючий вузол, що призводить до зниження якості перевірки в сталих часових проміжках, або до потреби розширення часового проміжку, потрібного для перевірки.

Зазвичай для зниження навантаження на системи де наявна присутність людей використовують автоматизацію на тих, чи інших етапах роботи цих систем, але в нашому випадку ще не існує алгоритмів, які б повністю усували потребу в суб’єктивному експертному аналізі людини людиною.

Також одним з варіантів можна назвати алгоритми машинного навчання, які можуть навчатися імітувати методики різних експертів для перевірки робіт, але в цьому випадку цей алгоритм ризикує стати упередженим та необ’єктивним, як і сама людина у котрої алгоритм буде навчитися.

У цій роботі я спробував поєднати сильні сторони алгоритмів та людей: швидкість, неупередженість, масштабованість та модульність машин зі здатністю людей без наявних чітких категорій надавати абстрактну оцінку речам з їх предметної області.

Мета і завдання дослідження.

Метою і завданням дослідження є створення програми задачею якої буде:

1 Автоматичний збір робіт

2 Попередній автоматизований аналіз робіт (на плагіат, граматичні помилки та стиль написання)

3 Розподіл зазначеної частки (N) робіт між самими авторами випадковим, або зазначеним чином

4 Повторний автоматизований аналіз робіт перевірених авторами один в одного

5 Розподіл зазначеної частки (M) робіт перевірених авторами (роботи до та після перевірки іншим автором та автоматизований аналіз обох робіт) для повторної перевірки експертами

Умовні позначення груп робіт

A – всі роботи надані для аналізу

Ā– всі роботи, що попередньо аналізовані автоматично (п.2)

n – частка робіт обраних для перевірки іншими авторами

N = (Ā*n) – роботи перевірені іншими авторами

Ñ – всі роботи перевірені іншими авторами, проаналізовані автоматично (п.4)

e = (1-n) – частка робіт обраних для перевірки експертами

E = (1-N) – роботи обрані для перевірки експертами

m – частка перевірених робіт, що буди обрані для верифікації експертами

M = (Ñ*m) – роботи обрані для повторної перевірки експертами

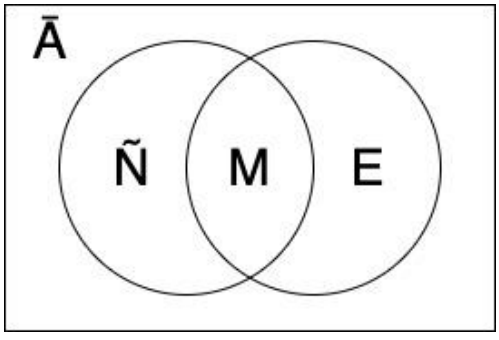

Рисунок 0.1 Множини робіт за типом перевірки

Об’єкт дослідження.

Системи розподілення

Документація інструментів для побудови системи

Результати роботи системи

Методи дослідження.

Джерела дослідження.

Інтернет ресурси:

https://www.softwaretestinghelp.com/code-review-tools/

https://habr.com/ru/post/207120/

Документація:

Наукова новизна одержаних результатів.

Результатом цієї роботи є веб-застосунок (клієнт для зручного користування системою)

Практичне значення одержаних результатів.

Ця система, завдяки її модульності та здатності масштабуватися може бути використана у вищих навчальних закладах, в сфері IT, школах, аудиті та інших галузях пов'язаних з перевіркою роботи в форматі задовільному для аналізу системою.

РОЗДІЛ 1: ТЕОРІЯ ПРО ПРЕДМЕТНУ ОБЛАСТЬ

1.1 ПРИНЦИП РОБОТИ СИСТЕМИ

1.1 Модулі **2** та **5** одразу перевіряють те, що надійшло на плагіат та помилки (додаткові модулі можуть бути додані в подальшому)

1. Роботи розподіляються на N та E групи в залежності від рейтингу автора[1]
2. Група N – роботи які перевіряють автори в залежності від рейтингу[2]

3.3 Група E – роботи які одразу перевіряють експерти

2 Початкові та перевірені роботи з групи N порівнюються системою: визначає кількість виправлень, їх природу та пропонує свою оцінку.

2.1Групу N перевіряють на помилки

2.2Група M – частина групи N, що проходить додаткову верифікацію у експертів також враховуючи помилки авторів, що перевіряли

Рисунок 1.0.1.1 Життєвий цикл роботи

Вирішальний етап – калібрування рейтингів, який відбувається вкінці.

На початковому етапі рейтинги призначаються всім однакові та/або з огляду на інші рейтингові системи, що були застосовані до впровадження цієї системи.

Надалі рейтинги авторів [та експертів, за потреби колегіальної перевірки автора[-ів]] калібруються після закінчення циклу перевірок, зображеного згори.

Калібрація базується на якості зданої роботи (визн. експертом та системою), якості перевірки роботи іншого (визн. експертом та системою) а також на попередньому рейтингу.

[1] Автори з нижчим рейтингом з більшою вирогідністю потраплять до групи N (перевірених іншими авторами з високим рейтингом)

[2] Роботи перевіряють випадково обрані автори з якомога більш високим рейтингом

Рейтинг *автора* збільшується, або зменшується в залежності від:

1 Оцінки перевіряючого та/або експерта, та виявлених системою помилок 1.1
2 Косинусна відстань між ***початковою*** та *експертною* роботою (чим ближче до експертної – тим більший рейтинг)
3 [Опціонально] Відсоток плагіату

Рейтинг *перевіряючого* збільшується, або зменшується в залежності від:

1. Косинусна відстань між ***перевіреною*** та *експертною* роботою (чим ближче до експертної – тим більший рейтинг)
2. Оцінка системи виявлених помилок [1.1]

Рейтингова система базується на принципі Proof-of-stake.
Proof-of-stake (PoS) («підтвердження часткою») — метод захисту в криптовалютах, заснований на необхідності доказу зберігання певної кількості коштів на рахунку. При використанні цього методу алгоритм криптовалюти з більшою ймовірністю обере для підтвердження чергового блоку в ланцюжку обліковий запис з великою кількістю коштів на рахунку. [3]

Отже замість коштів в нашому випадку рейтинг авторів [та, опціонально, експертів], який калібрується впродовж часу за допомогою власних та перевірених робіт.

[3] Habr Proof of Stake

1.2 АЛЬТЕРНАТИВНІ ПІДХОДИ ТА КОНКУРЕНТИ

Головними конкурентами запропонованій мною евристичній системі є застосування для автоматичної взаємо-перевірки коду (Code review).
Code review – (огляд коду, іноді його називають експертною оцінкою) - це діяльність із забезпечення якості програмного забезпечення, в якій один або кілька людей перевіряють програму в основному, переглядаючи та читаючи частини її вихідного коду, і вони роблять це після впровадження або як переривання реалізації. Принаймні одна з осіб не повинна бути автором коду. Особи, які здійснюють перевірку, крім автора, називаються "рецензентами".[4]

Приклад застосування для автоматичного Code review – Collaborator[5]
Функціонал наявний в даному продукті:

- Перегляд змін коду, визначення дефектів та коментарі до конкретних рядків коду.
- Спеціальні шаблони оглядів. Встановлення спеціальних полів, контрольних списків та груп учасників.
- Інтеграція з 11 різними SCM, та IDE, такими як Eclipse & Visual Studio.
- Створення спеціальних звітів.

Як стверджують розробники Collaborator – огляд можливо проводити не лише коду програм, а й звичайних документів, чим мій застосунок дуже схожий, але акцент вони роблять саме на огляди програмних застосунків, про що свідчать інтеграції цього застосунку із середовищами розробки (Eclipse & Visual Studio).

Головною відмінною рисою Collaborator та мого застосунку є відсутність у першого будь-якої рейтингової системи, та систематизованого, автоматичного призначення рецензента до роботи в залежності від рейтингу автора цієї роботи.

[4] Wikipedia – Code Review
[5] Collaborator

У конкурента рецензент призначається вручну, а отже процес стає схожий на відправлення e-mail листа з правками коду автора (що є одною з методик огляду коду).

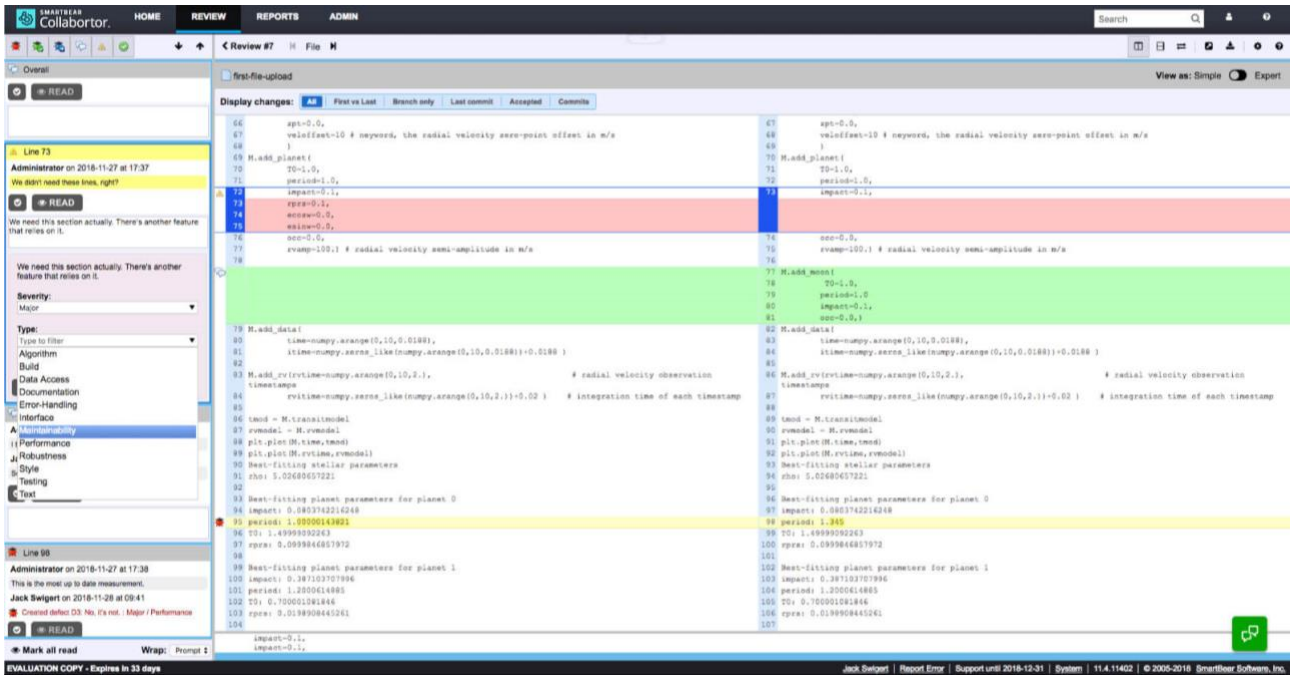

Рисунок 0.2 Collaborator, приклад перегляду змін коду до/після огляду

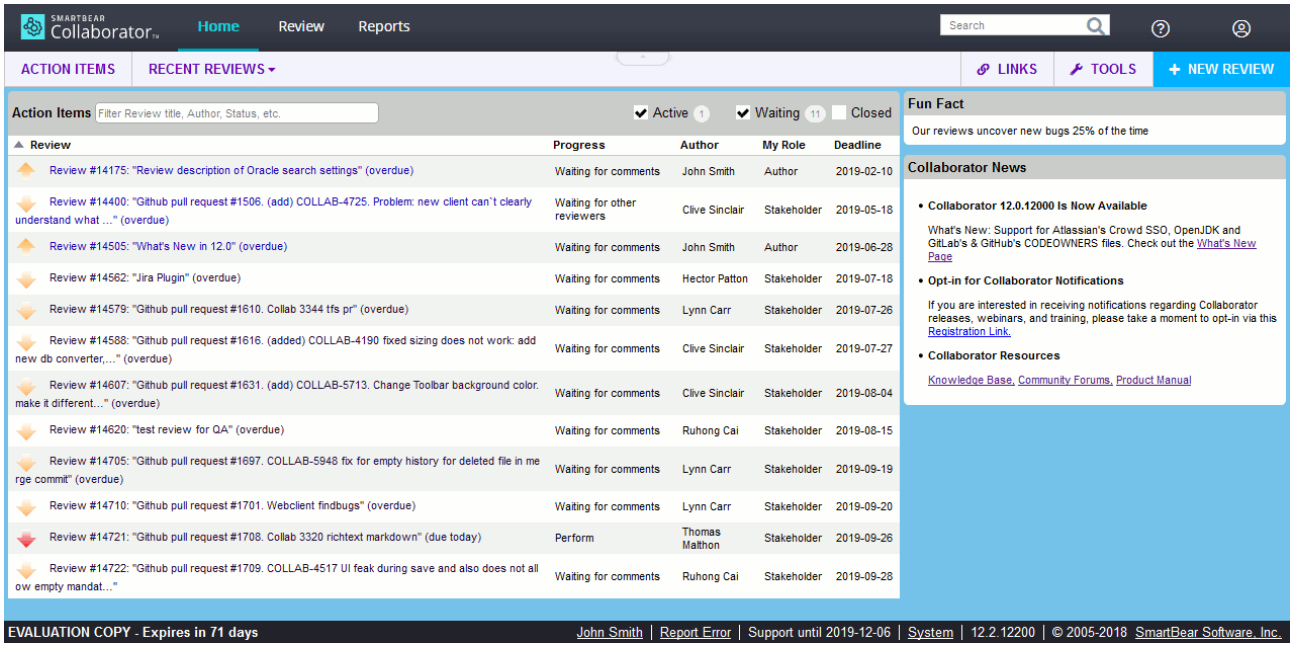

Рисунок 0.3 Collaborator, приклад списку робіт для перевірки

1.3 ЕФФЕКТИВНІСТЬ СИСТЕМИ ТА ЗАОЩАДЖЕННЯ РЕСУРСІВ

На графіку зображена залежність кількості робіт для перевірки експертом від частки робіт, які перевіряються іншими авторами та кількість перевірених авторами робіт, що були додатково верифіковані експертом. Зелені стрілки вказують оптимальну стратегію зниження навантаження на експерта.

1. На першому кроці (горизонтальна стрілка) експерт коригує в системі параметр “n” (додаток А), поступово переводячи його з 0 до 1 (100 %), в результаті експерт досі оброблює всі 100% робіт, але тепер він верифікує перевірені роботи, а отже, разом з цим, калібрує рейтинг рецензентів.
2. У точці “А” експерт, починає знижувати параметр “m” (додаток А) залишаючи “N” (додаток А) на 100 %. На цьому етапі рецензенти з високим рейтингом з меншою вірогідністю будуть верифіковані експертом, що знизить ризик неякісної перевірки автором.

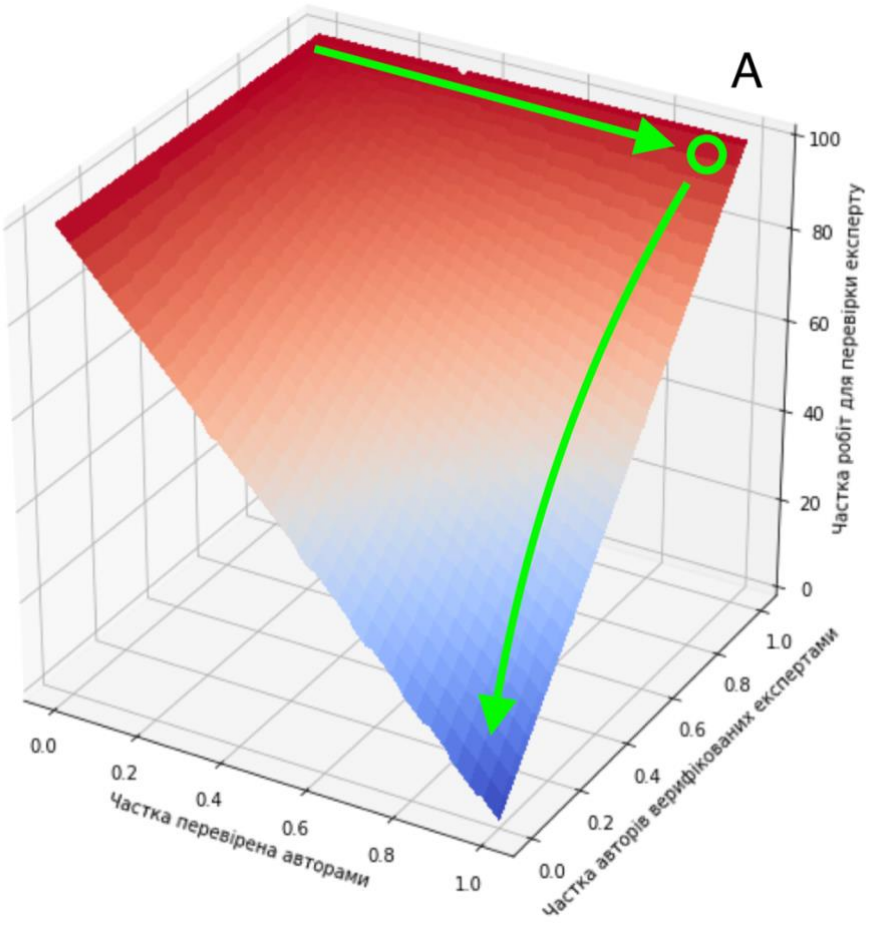

Рисунок 1.0.4.1 Оптимальна стратегія вибору параметрів системи

РОЗДІЛ 2: ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 МОВА ПРОГРАМУВАННЯ

У своїй роботі я використав мову Python (пайтон).
Python – високорівнева, інтерпретована мова програмування загального призначення. Створена Гідо ван Россумом та вперше випущена у 1991 році. Філософія дизайну Python підкреслює читабельність коду завдяки помітному використанню табуляції. Python динамічно-типізований та має збирач сміття. Він підтримує декілька парадигм програмування, включаючи структуроване (зокрема, процедурне), об'єктно-орієнтоване та функціональне програмування.

Переваги та недоліки мови Python.

Недоліки:

1. Пітон повільний. Оскільки це інтерпретована мова, вона *часто* у багато разів повільніше, ніж компільовані мови, але бувають і винятки, за яких, коли код на Python може виконуватись швидше, ніж, навіть, відповідний код на C.
2. Слабкий у мобільних застосунках.
3. Оскільки Python динамічно-типізований, під час виконання виявляється більше помилок.

Переваги:

1. Простий і зрозумілий синтаксис.
2. Наявна об'єктно-орієнтована парадигма.
3. Підтримує імперативне та функціональне програмування.
4. Підтримує декілька платформ (веб та мобільні обчислення).
5. Велика бібліотека.
6. Python легко розширюється за допомогою коду C / C ++ / Java.
7. Його можливість “склеювати” великі програмні компоненти.

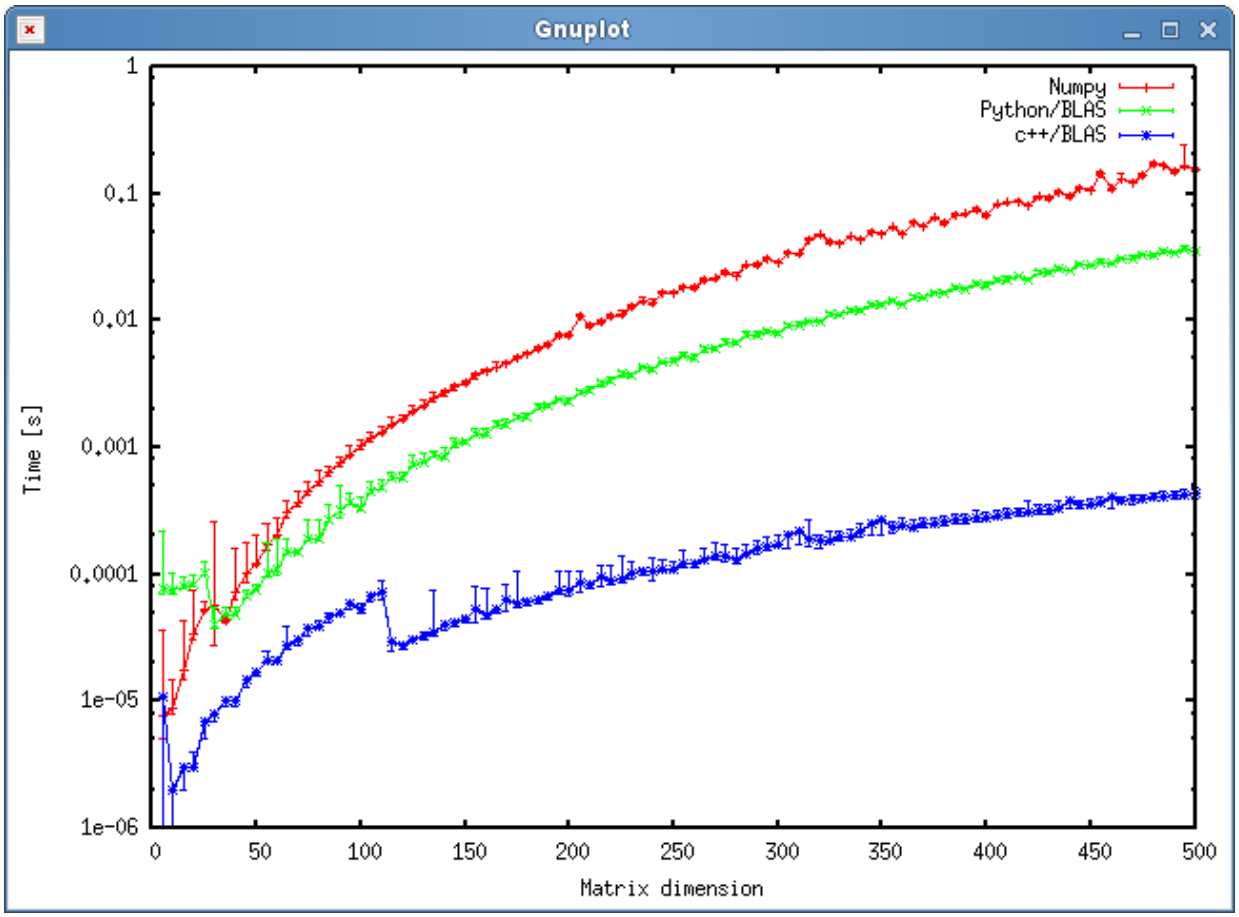

Рисунок 2.0.1.1 Порівняння швидкості множення матриць на Python, C++ та Numpy[6]

Я обрав Python як основну мову через умови, зазначені в пунктах 5,6 та 7 в перевагах цієї мови.

Однією з найголовніших умов мого застосунку є модульність, можливість, за потреби, під'єднати API та програмні модулі для додаткового аналізу робіт, а також, потенційно, можливість впровадити машинне навчання для виявлення найоптимальніших параметрів, таких як "m" та "e" з додатку "A" до цієї роботи. Саме тому ці 3 переваги, разом із бібліотекою Matplotlib[7] для переважили недоліки.

[6] NumPy - це бібліотека для мови програмування Python, яка підтримує великі багатовимірні масиви та матриці, а також велику колекцію математичних функцій високого рівня для роботи над цими масивами.

[7] Matplotlib - це бібліотека графіків для мови програмування Python та її чисельного розширення з математики NumPy.

2.2 СТЕК ЗАСТОСУНКУ

2.2.1 ВЕБ ЧАСТИНА – Django

Основну частину проекту займає веб застосунок написаний на мові Python за допомогою фреймворку Django.
Django - це веб-фреймворк високого рівня для мови Python, який надає можливість швидкої розробки та має чистий, прагматичний дизайн.

Особливості Django:

- Django був розроблений, щоб допомогти розробникам перенести програми від концепції до завершення якнайшвидше.
- Django включає десятки додаткових додатків, які можна використовувати для вирішення загальних завдань веб-розробки. Django піклується про автентифікацію користувачів, адміністрування вмісту, карти сайтів, RSS-канали та багато інших завдань.
- Django допомагає розробникам уникнути багатьох поширених помилок у безпеці, таких як інжекція SQL, XSS, підробка веб-сайтів та підключення кліків. Його система аутентифікації користувачів забезпечує безпечний спосіб управління обліковими записами та паролями користувачів.
- Деякі з найбільш завантажених сайтів на планеті використовують здатність Джанго швидко та гнучко масштабуватись, щоб задовольнити найважчі вимоги з оптимізації балансування трафіку.
- Компанії, організації та уряди використовували Django для побудови широкого спектру речей - від систем управління контентом та соціальних мереж до наукових обчислювальних платформ.

2.2.2 БАЗА ДАНИХ – SQLite

SQLite - це система управління реляційними базами даних, що міститься в бібліотеці C. На відміну від багатьох інших систем управління базами даних, SQLite не є двигуном бази даних клієнт-сервер. Скоріше, вона вбудована в кінцеву програму. SQLite має прив'язку до багатьох мов програмування. Це, мабуть, найбільш широко використовуваний двигун бази даних, оскільки його сьогодні використовують декілька широко розповсюджених браузерів, операційних систем та вбудованих систем (таких як мобільні телефони).

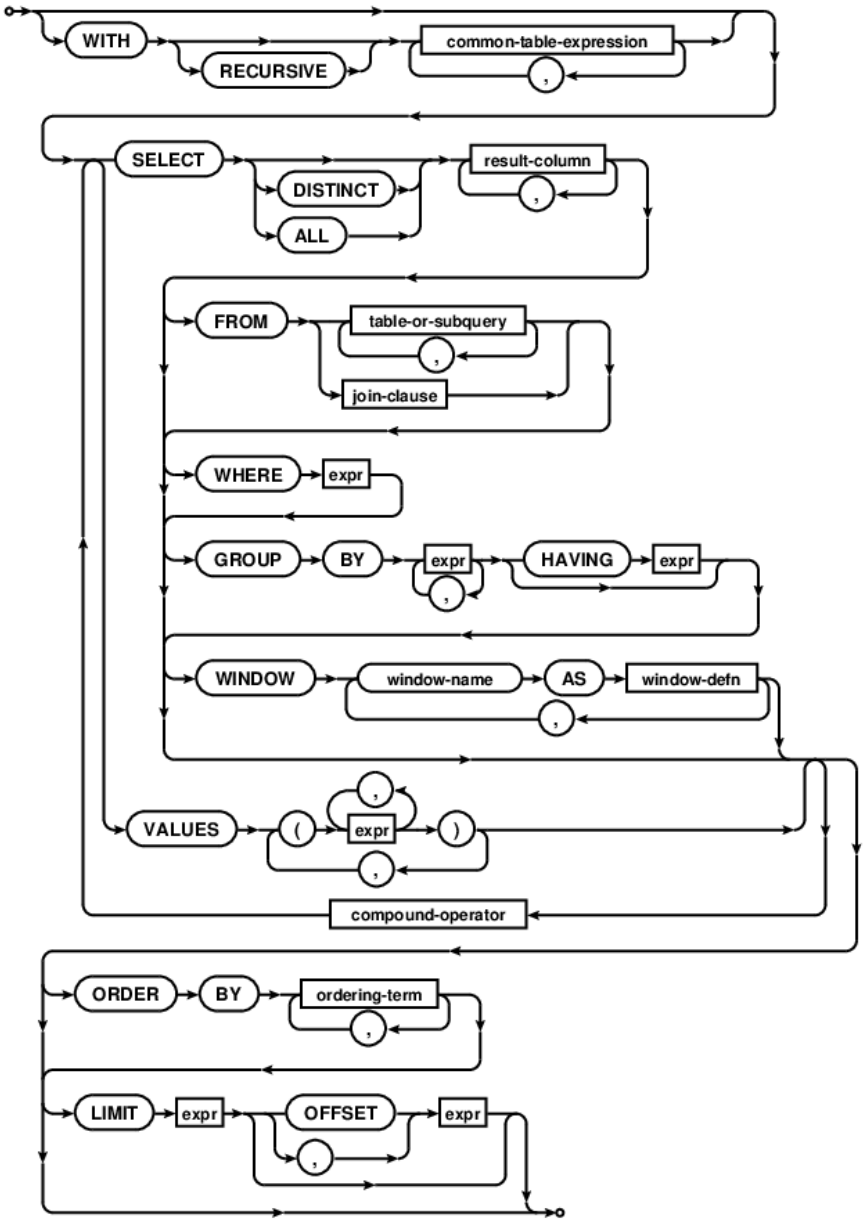

Рисунок 2.2.0.2.1 синтаксис SQL для SQLite

2.3 ДОПОМІЖНІ БІБЛІОТЕКИ/МОДУЛІ

2.3.1 Difflib [3]

Ця бібліотека надає класи та функції для порівняння послідовностей (в тому числі текстових). Вона може бути використана, наприклад, для порівняння файлів, і може показувати різницю в різних форматах, включаючи HTML та уніфікований diff[8]. Бібліотека може порівняти пари послідовностей будь-якого типу, якщо елементи послідовності є хешованими.

Основний алгоритм є доповненням до алгоритму, опублікованого в кінці 1980-х Ратклiффом та Обершельпом під назвою "узгодження гештальтного шаблону". Алгоритм знаходить найдовший суміжний збіг послідовностей, який не містить «сміття», наприклад, порожні рядки або пробіли.

(Поводження зі сміттям - це розширення до алгоритму Ратклiффа та Обершельпа.) Та ж сама ідея потім застосовується рекурсивно до фрагментів послідовностей ліворуч та праворуч від відповідного підпорядкування.

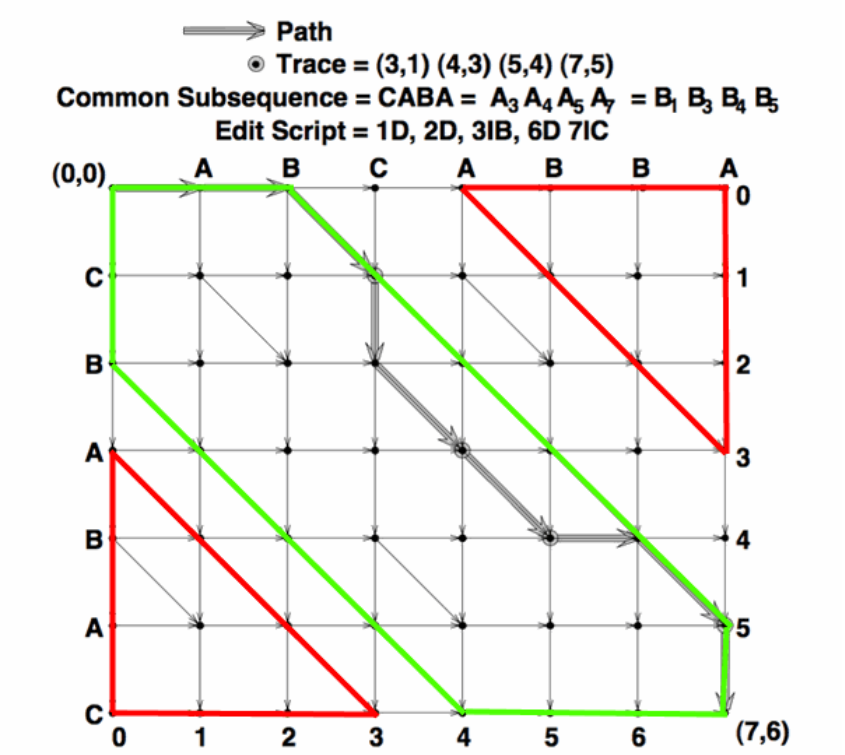

Рисунок 2.3.1.1 Алгоритм "Узгодження гештальтного шаблону"

[8] Уніфікований diff (або diff уніфікований формат, уніфікований контекст diff, unidiff) - це формат diff, який зазвичай використовується для файлів патчів.

2.3.2 PYFTPDLIB [4]

Pyftpdlib - це бібліотека високого рівня, яка дозволяє легко писати асинхронні портативні FTP-сервери за допомогою мови Python.

Оскільки роботи авторів для розподілу та перевірки надходять у формі файлів – доцільне застосування FTP протоколу, як одного зі шлюзів, котрим автори можуть надсилати роботи.

FTP (“Протокол передачі файлів”) – це стандартний мережевий протокол, який використовується для передачі комп'ютерних файлів між клієнтом і сервером в комп'ютерній мережі. FTP побудований на архітектурі моделі клієнт-сервер, використовуючи окремі з'єднання управління та даних між клієнтом і сервером. Хочу зауважити, що, хоч, FTP шлюз наявний в системі, для більшої зручності основним є шлюз веб застосунку, який працює на HTTP.

2.3.3 Matplotlib [5]

Matplotlib - це бібліотека графіків для мови програмування Python.

Цю бібліотеку я застосував для аналізу та візуалізації результатів моєї роботи.

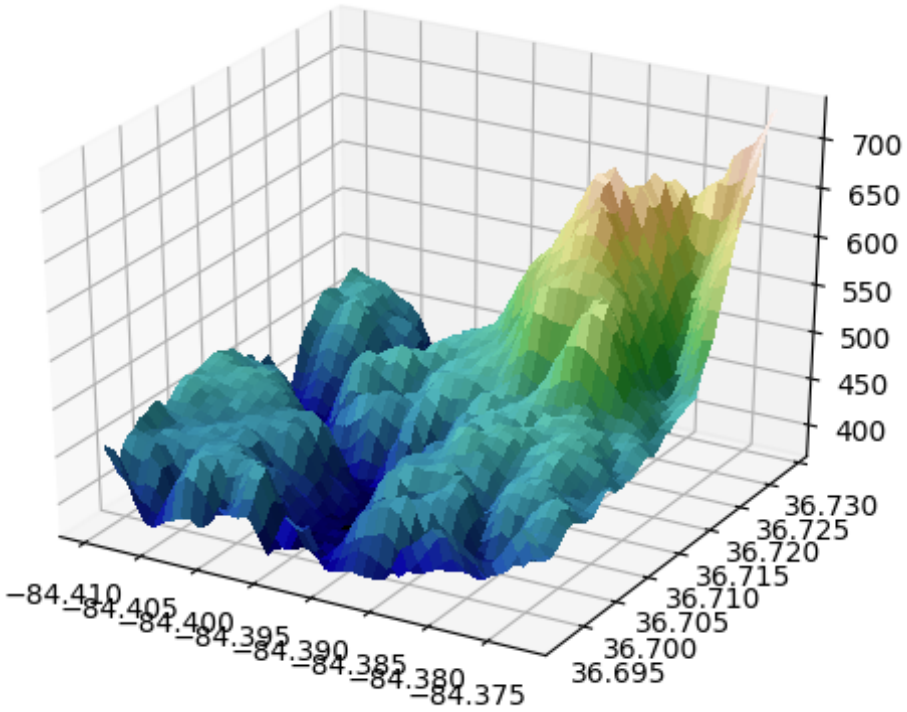

Рисунок 2.3.0.3.1 Приклад 3-D візуалізації

2.3.4 SMTPLIB [6]

Модуль smtplib визначає об’єкт сеансу SMTP-клієнта, який може використовуватися для надсилання пошти на будь-яку Інтернет-машину з демоном SMTP або ESMTP.

Цей модуль використовується для сповіщення клієнтів, а також, задля можливого надсилання файлів для перевірки за відсутності доступу до інших протоколів (HTTP, FTP), або як ще один *вихідний* шлюз.

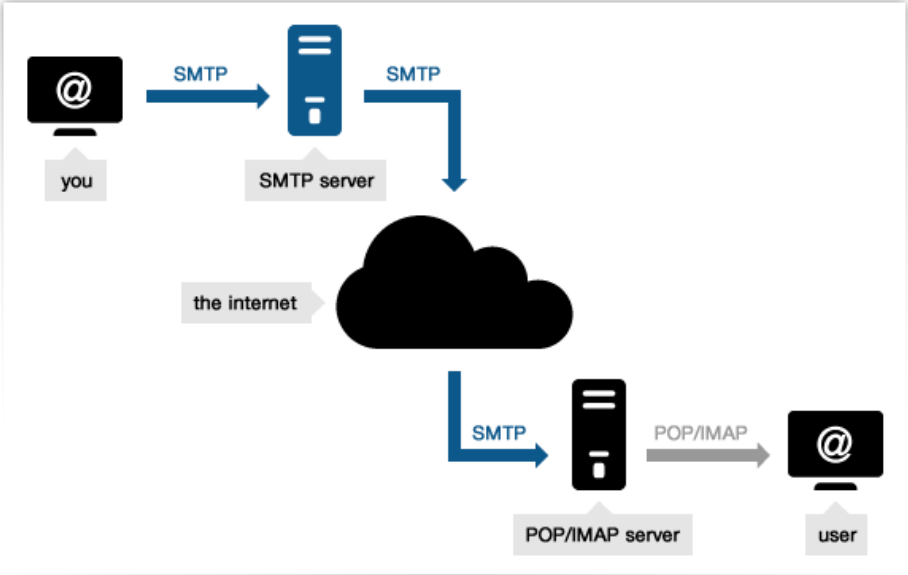

Рисунок 2.3.0.4.1 Життєвий цикл пакету SMTP

2.3.5 LANGUAGETOOL [7]

Languagetool - це програма з відкритим вихідним кодом та API для перевірки граматики.


```
"message": "Знайдено потенційну орфографічну помилку.",
"shortMessage": "Орфографічна помилка",
"replacements": [
  {
    "value": "Дорога"
  },
  {
    "value": "Зарога"
  },
  {
    "value": "Рарога"
  },
```


Рисунок 2.3.5.1 Приклад відповіді API для слова "Дарога" (Через "а")

РОЗДІЛ 3: АНАЛІЗ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

3.1 СТРУКТУРА ЗАСТОСУНКУ

3.1.1 БАЗА ДАНИХ

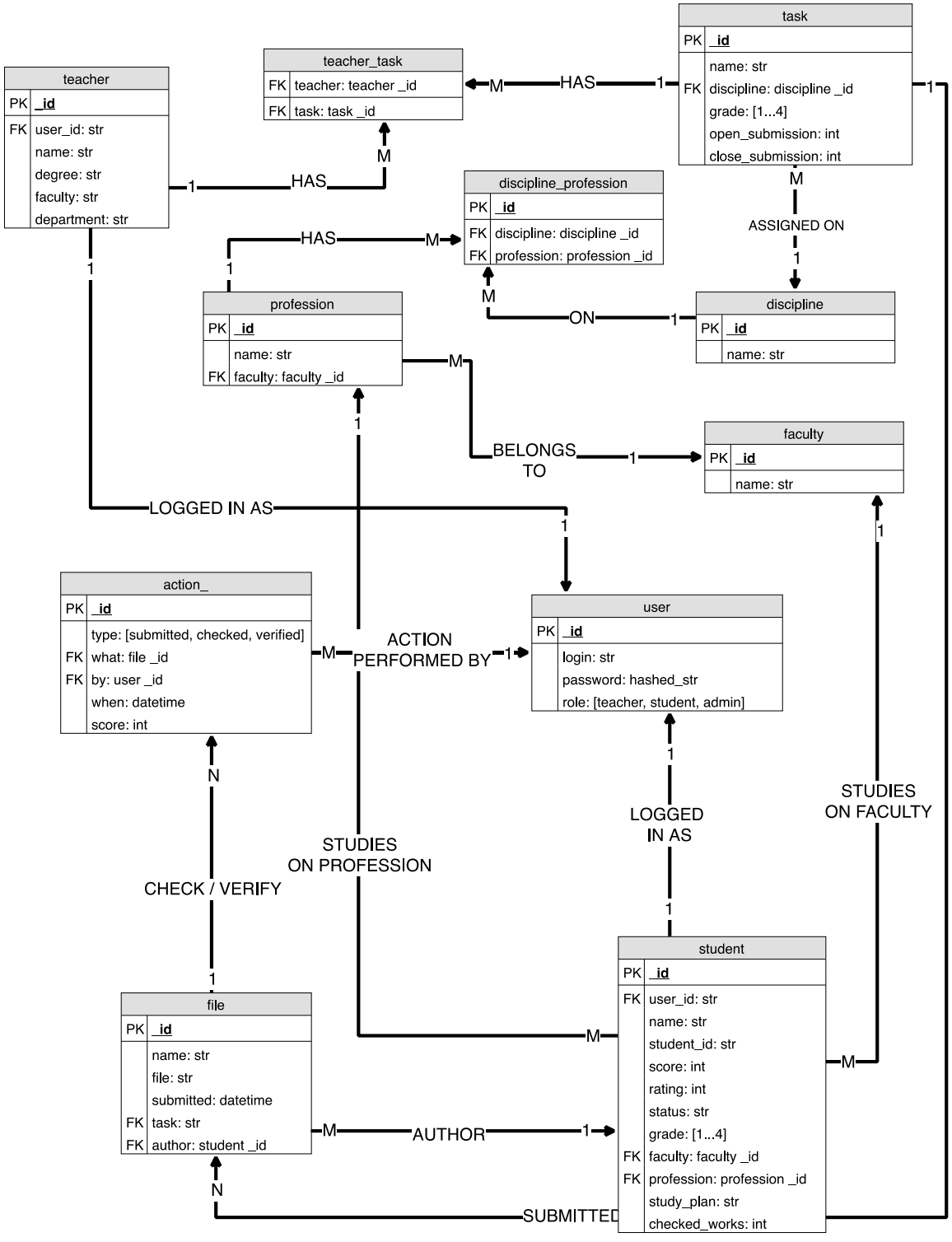

Рисунок 3.1.1.0.1 ER-модель бази даних

3.1.2 ВЕБ ЗАСТОСУНОК

1 Блок з класами, що не залежать від веб застосунку:
1.1 Comparator – знаходить відмінності між файлами
1.2 GrammarCheckAPI – надсилає текст файлу для перевірки на граматику.
1.3 ProbabilityCalc – вираховує вирогідності, що автор стане рецензентом, та, що робота може бути перевірена іншим автором.
1.4 SQL_DB – створює та заповнює базу даних
1.5 Sql_util – SQL скрипти
1.6 Util – параметри для системи, та вспоміжні функції
2 Блок templates – веб-сторінки клієнта.
3 Частина структури фреймворку Django: класи для створення DTO об'єктів, маппінг URL-адрес до обробників, самі обробники, тести та головний класс, що все контролює.

4 Директорія, де зберігаються надіслані файли
4.1 HTTP – для файлів з веб застосунку
4.2 Інші – для групування файлів надісланих через FTP

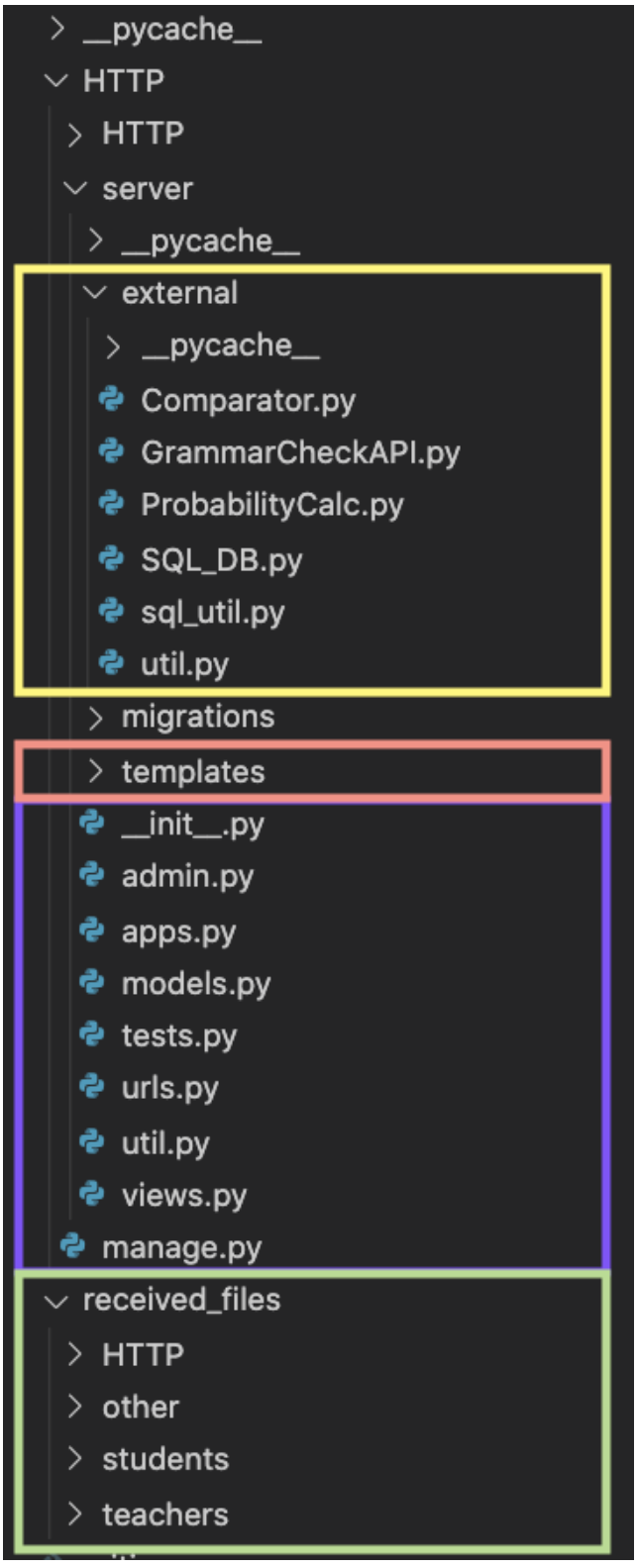

Рисунок 3.1.2.1 HTTP сервер

1 DB – База даних SQLite
2 FileFetcher – клас, що приймає файли по протоколу FTP
3 FileSender – клас, що відправляє дані по протоколу SMTP

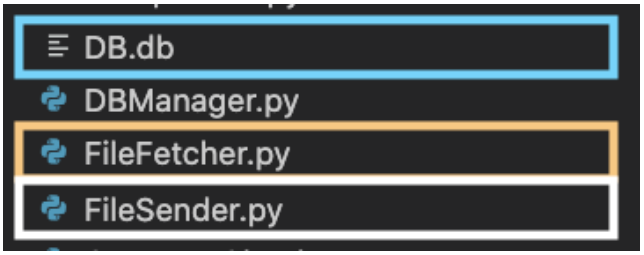

Рисунок 3.1.2.2 FTP, SMTP та база даних

3.2 РОБОЧИЙ ПРОЦЕС

3.2.1 НАДСИЛАННЯ НА ПЕРЕВІРКУ

1 Автор авторизується на клієнті сайту та потрапляє на сторінку для відправлення роботи на перевірку
2 Файл асоціюється із автором, перевіряється на граматичні [та плагіат] помилки, та зберігається у базу даних зі “станом” “submitted” (поданий для перевірки)
3 Також для відправлення файлів можна використовувати FTP клієт, наприклад FileZilla. Файли надіслані до віддаленого репозиторію одразу копіюються до бази даних та до іншої, прихованої репозиторії

Рисунок 3.2.1.1 Надсиланятня файлу для перевірки

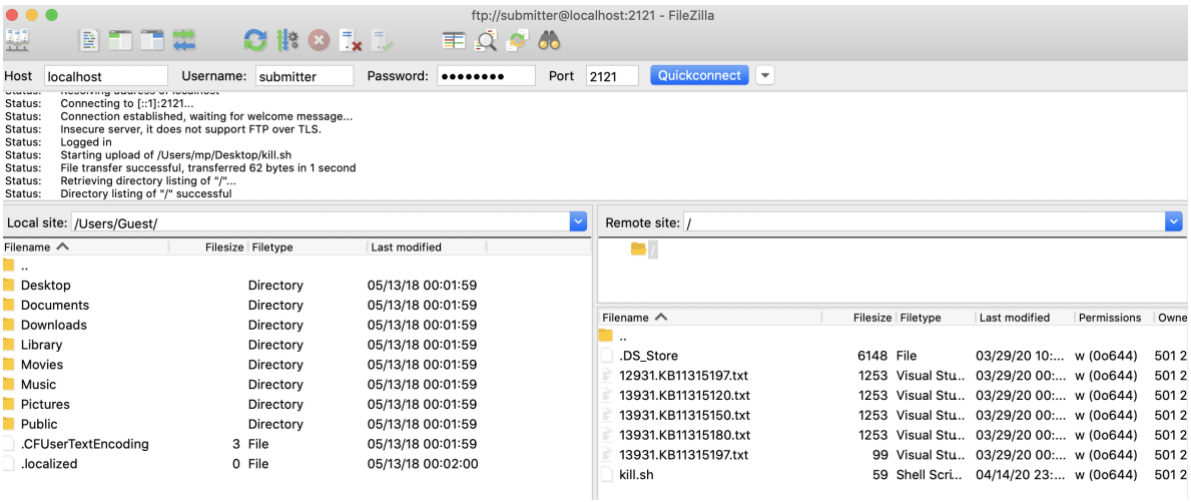

Рисунок 3.0.2.1.2 Інтерфейс FTP клієнта – FileZilla

3.2.2 ПРИЗНАЧЕННЯ РЕЦЕНЗЕНТІВ

1. За командою експерта “assign” (призначити) – роботи зі статусом “поданий для перевірки” призначаються для перевірки рецензентам зі статусом “доступний” з такими умовами:
   1.1. Рецензент не може перевіряти свою роботу
   1.2. Рецензенти призначаються відповідно рейтингу автора: автор з більшим рейтингом – відповідно отримає рецензента з більшим рейтингом
   1.3. Автор з низьким рейтингом більш сприятливий до перевірки рецензентом ніж експертом

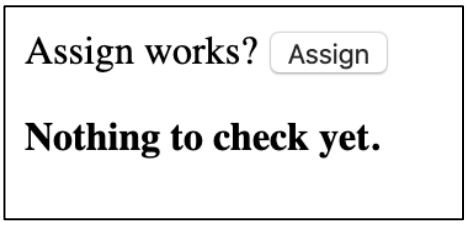

Рисунок 3.2.2.1 Призначити роботи для перевірки

2. Призначені роботи з’являються в облікових записах рецензентів у категорії “перевірити”, та, за потреби, надсилаються поштою.
3. Роботи не призначені рецензентам відправляються на перевірку одразу експертам.
4. Роботи можна завантажити, а після перевірки – надіслати перевірену файл у поле для відповідної роботи
5. Також можна виставити запропонований бал за роботу, який буде враховано при калібруванні рейтингу автора та рецензента

Рисунок 3.0.3.2.2 Роботи які треба перевірити

3.2.3 ПОРІВНЯННЯ ВИХІДНОЇ ТА ПЕРЕВІРОНОЇ РОБІТ

1. Перевірений файл додатково перевіряється на помилки за допомогою API languagetool.org [7]
2. Вихідний та перевірений файли порівнюються за допомогою бібліотеки difflib [3], яка знаходить послідовність дій, які потрібно виконати над початковим файлом щоб з нього отримати перевірений.

Рисунок 3.2.3.1 Відповідь API зі знайденими помилками

Я використовував функцію *get_opcodes()*

Ця функція повертає список з 5-ти кортежів, що описують, як перетворити послідовність a на послідовність b.

Кожен кортеж має форму (назва перетворення, i1, i2, j1, j2).

Перший кортеж має i1 == j1 == 0, а інші кортежі мають i1, рівний i2 від попереднього кортежу, і, таким же чином, j1 дорівнює попередньому j2.

Цю функцію я застосовував для порівняння вхідної роботи та перевіреної автором-рецензентом, для ідентифікації виконаних перетворень (виправлень).

Для кожного перетворення задаються коефіцієнти, що надалі визначають оцінку, якою система оцінює автора (додатково до оцінки рецензента).

```
>>> a = "qabxcd"
>>> b = "abycdf"
>>> s = SequenceMatcher(None, a, b)
>>> for tag, i1, i2, j1, j2 in s.get_opcodes():
...     print('{:7}   a[{}:{}] --> b[{}:{}] {!r:>8} --> {!r}'.format(
...         tag, i1, i2, j1, j2, a[i1:i2], b[j1:j2]))
delete    a[0:1] --> b[0:0]      'q' --> ''
equal     a[1:3] --> b[0:2]     'ab' --> 'ab'
replace   a[3:4] --> b[2:3]      'x' --> 'y'
equal     a[4:6] --> b[3:5]     'cd' --> 'cd'
insert    a[6:6] --> b[5:6]       '' --> 'f'
```

Рисунок 3.2.3.2 Приклад порівняння двох текстових послідовностей

3. Враховуючи частку та природу виправлень система пропонує свою оцінку за роботу автора, базуючись на заданих параметрах для кожного з виправлень.

```
Comparing  Lorem ipsu  and  Lorem ipsu
equal     a[0:393] --> b[0:393]
==========================
insert    a[393:393] --> b[393:397]
==========================
Grade for author is:  96.9620253164557
```

Рисунок 3.2.3.2 Виправлення та запропонована оцінка автору

4. Отримані помилки та послідовність дій зберігаються разом з перевіреним файлом в базі даних в табличці actions

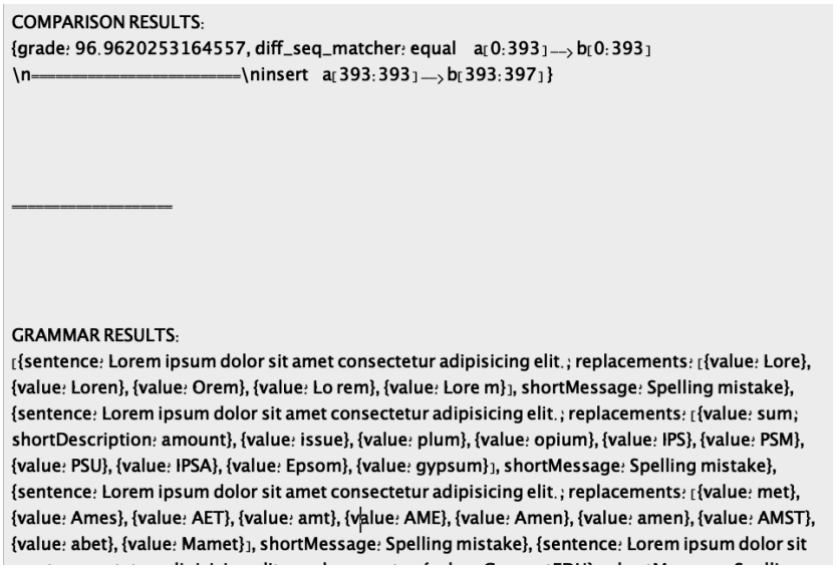

Рисунок 3.2.3.3 Інформація про конкретну перевірку

5. Також виставляється пропонована рецензентом оцінка, а оцінка системи, що базується на різниці між фалами записується разом із отриманими помилками та послідовністю дій

6. Перевіреному файлу встановлюється стан “checked” (перевірений), цей файл з'являється в історії перевірок рецензента та файл стає доступним для подальшої верифікації експертом

Рисунок 3.2.3.4 Перевірений файл

3.2.4 ВЕРИФІКАЦІЯ ЕКСПЕРТОМ

1. На цьому етапі система випадково обирає рецензентів для перевірки за принципом: чим нижчий рейтинг рецензента – тим більш вирогідно, що він буде верифікований експертом.
2. Після перевірки експертом – 3 файли: початковий, рецензований та верифікований порівнюються між собою та відбувається калібрування рейтингів автора та рецензента:
   a) Варіант **рецензента** виявився ближчим до варіанту експерта ніж автор – **рейтинг рецензента** збільшується, а автору виставляється остаточна оцінка – середнє, зважене оцінок системи, рецензента, та різниці між варіантом експерта.
   b) Варіант **автора** виявився ближчим до варіанту експерта ніж рецензент – **рейтинг автора** збільшується, а автору виставляється остаточна оцінка – середнє, зважене оцінок системи, та варіантом експерта.
3. Відкалібровані рейтинги оновлюються у базі даних
4. Рецензент знову переходить у режим “доступний” (для рецензування)

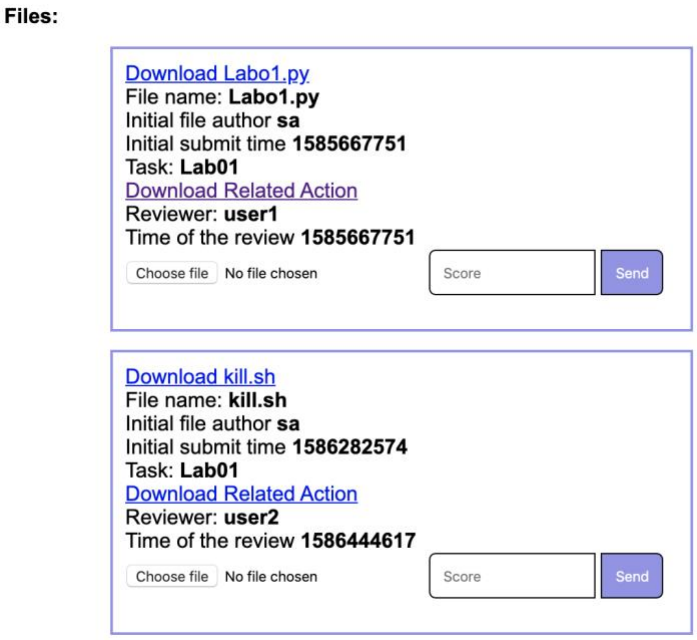

Рисунок 3.2.3.5 Файли для верифікації експертом (можливо завантажити початковий файл та його перевірку експертом)

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Рисунок 3.2.3.6 Формула косинусної міри схожості

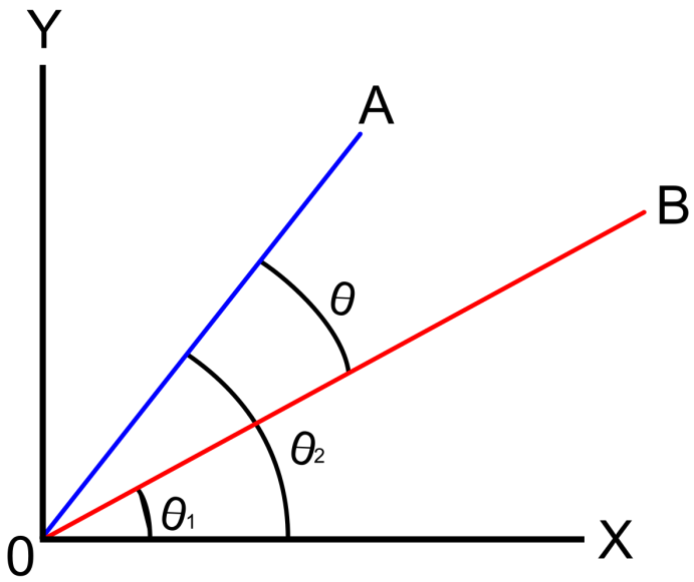

Рисунок 3.2.3.7 Варіанти автора (X), рецензента (B) та експерта(A) у вигляді векторів

На малюнку 3.2.3.6 зображено варіант, коли варіант **рецензента** виявився ближчим до варіанту експерта ніж автор.

В цьому випадку система збільшує рейтинг рецензента наступним чином:

1. Вираховується зворотня відстань (кут) між A та B (1 - θ)
2. Вираховується відстань (кут) між B та X (θ_1)
3. Рейтинг множиться окремо на обидві змінні
4. І їх сумма додається до початкового рейтингу

Висновки

У результаті виконання роботи я розробив програму (веб застосунок) для автоматичного збору файлів для перевірки робіт, перевірки робіт на та помилки системою та верифікації перевірених авторами робіт від експертів (викладачі, керівники), а також придумав декілька варіантів покращення застосунку:

- Під’єднати модуль з машинним навчанням, який буде знаходити ідеальний набір параметрів для розподілу та верифікації робіт
- Під’єднати систему перевірки на плагіат
- Контейнеризувати застосунок за допомогою docker
- Впровадити перевірку косинусною відстаню між файлами
- Видаляти “сміття” з файлів (коментарі до коду, тощо) та звіряти хеш суми файлу надісланого до системи та файлу для перевірки іншому автору
- Підключення системи контролю версій GIT замість таблиці “actions” в базі даних для більш зручного та раціонального збереження та контролю змін у файлах.
- Зробити безперервне призначення рецензентів роботам, замість призначення за командою експерта
- Переписати модулі для обчислення на C++ для пришвидшення застосунку

Список джерел

Веб ресурси

[1] - https://www.softwaretestinghelp.com/code-review-tools/

[2] - https://www.crummy.com/software/BeautifulSoup/bs3/documentation.html

[3] - https://docs.python.org/3/library/difflib.html

[4] - https://pyftpdlib.readthedocs.io/en/latest/faqs.html#introduction

[5] - https://matplotlib.org/

[6] - https://docs.python.org/3/library/smtplib.html

[7] - https://languagetool.org/http-api/swagger-ui/#/

[8] - https://www.cse.iitb.ac.in/~varsha/allpapers/packet-scheduling/wfqJuniper.pdf

Додаток А

Перелік прийнятих скорочень

Типи користувачів

Автор – автор роботи, яку перевіряє рецензент та/або експерт [студент, розробник]

Рецензент – автор, призначений системою, що перевіряє іншого автора, та може бути верифікований (не може перевіряти свою роботу) [студент, розробник]

Експерт – користувач з авторитетною думкою, з правом остаточного вердикту [викладач, керівнк тезнічного відділу розробки]

Умовні класифікація робіт

A – всі роботи надані для аналізу

Ā– всі роботи, що попередньо аналізовані автоматично (п.2)

n – частка робіт обраних для перевірки іншими авторами

N = (Ā*n) – роботи перевірені іншими авторами

Ñ – всі роботи перевірені іншими авторами, проаналізовані автоматично (п.4)

e = (1-n) – частка робіт обраних для перевірки експертами

E = (1-N) – роботи обрані для перевірки експертами

m – частка перевірених робіт, що буди обрані для верифікації експертами

M = (Ñ*m) – роботи обрані для повторної перевірки експертами

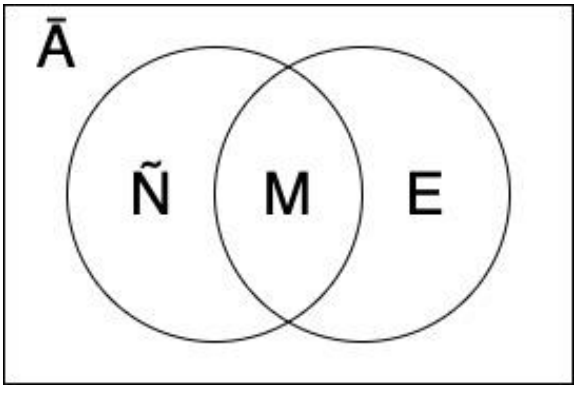

Рисунок 0.4 Множини робіт за типом перевірки

Додаток Б

Інші скорочення

DTO – Data Transfer Object

FTP – File Transfer Protocol

SMTP – Simple Mail Transfer Protocol

Додаток В

Процес обрання рецензентів та робіт для рецензії
А також робіт для верифікації експертами

```
import sys
from server.external.util import constants
import numpy as np
from server.external.SQL_DB import DB
from server.external.sql_util import files_to_check_select_sql,
available_students_select_sql
from pprint import pprint


def choose_ntcf(files, ntcf_amount, key=lambda x: x["author_id"]):
    if len(files) <= 1:
        return []
    np_probabilities = [x["author_prob"] for x in files]
    ntcf = np_unique(files, np_probabilities, ntcf_amount, key=key)
    return ntcf


def choose_verify_ntcp(actions):
    # actions = adptor(actions_dict)
    verify_amount = int(len(actions) * constants["verify_ntcp_percent"])
    one_percent_checker = one_percent(actions, True)
    add_prob_to_be_reviewed(actions, one_percent_checker)
    ntcf = choose_ntcf(actions, verify_amount, lambda x: x["id"])
    return ntcf


def choose_ntcf_checkers(students, ntcf_amount):
    np_probabilities = [x["reviewer_prob"] for x in students]
    # choose <ntcf_amount> of files with each file probability <probabilities>
    def key(x): return x["id"]
    ntcf = np_unique(students, np_probabilities, ntcf_amount, key=key)
    return ntcf


def np_unique(items, np_probabilities, amount, max_depth=100, key=lambda x: x):
    if len(items) == amount:
        return items
    chosen = np.random.choice(items, amount, p=np_probabilities)
    unique_items = set(list(map(key, chosen)))
    while len(unique_items) != len(chosen) and max_depth > 0:
        chosen = np.random.choice(items, amount, p=np_probabilities)
        unique_items = set(list(map(key, chosen)))
        max_depth -= 1
    if max_depth <= 0:
        return []
```

```
    return chosen


def get_reviewer_prob(stud_rating, one_percent_val):
    stud_prob = stud_rating / (100 * one_percent_val)
    return stud_prob


def get_author_prob(stud_rating, one_percent_val):
    stud_prob = (100 - stud_rating) / (100 * one_percent_val)
    return stud_prob


def assoc_checkers_ntcf(ntcf, ntcf_checkers):
    result = {}
    if len(ntcf) != len(ntcf_checkers):
        print("Incompatible checkers length and author files length")
        return {}
    if len(ntcf) <= 1:
        print("Guarantied self-check")
        return {}
    sorted_ntcf = sorted(ntcf, key=lambda x: x["rating"])
    sorted_checkers = sorted(ntcf_checkers, key=lambda x: x["rating"])
    checkers_ids = list(map(lambda x: x["id"], sorted_checkers))
    result = dict(zip(checkers_ids, sorted_ntcf))
    self_check = False
    while True:
        for k, v in result.items():
            if k == v["author_id"]:
                self_check = True
                break
        if self_check:
            checkers_ids = checkers_ids[1:] + checkers_ids[:1]
            result = dict(zip(checkers_ids, sorted_ntcf))
            self_check = False
        else:
            break
    return result


def add_prob_to_be_reviewed(files, one_percent_author):
    for file in files:
        probability = get_author_prob(file["rating"], one_percent_author)
        file["author_prob"] = probability


def add_prob_to_review(students, one_percent_author):
    for student in students:
```

```
        probability = get_reviewer_prob(student["rating"], one_percent_author)

        student["reviewer_prob"] = probability


def one_percent(items, inverse=False):
    if inverse:
        return sum([100 - v["rating"] for v in items]) / 100
    else:
        return sum([x["rating"] for x in items]) / 100


def get_ntcf_amount(items):
    return int(len(items) * constants["student_check_margin"])


def adptor(dictionary):
    result = []
    for k, v in dictionary.items():
        v["checker_id"] = k
        result.append(v)
    return result


def checker_assign(student_check):
    adapted = adptor(student_check)
    one_percent_author = one_percent(adapted, True)
    add_prob_to_be_reviewed(adapted, one_percent_author)
    np_probabilities = [x["author_prob"] for x in adapted]
    ntcf_amount = get_ntcf_amount(adapted)
    def key(x): return x["id"]
    tcc = np_unique(adapted, np_probabilities, ntcf_amount, key=key)
    return tcc


def main(files={}, students={}):
    """
    Assignes margin of in put files to be reviewed by other available students.

    @params:
        :param files:  Files, eligible for check: dict
        :param students: Available students to check
        : return: {"reviewed_by_students": {who: what}, "reviewed_by_teachers": []}

    """
    if not (files or students):
        dbo = DB()
    if not files:
```

```
        files = dbo.run_query(files_to_check_select_sql)
    if not students:
        students = dbo.run_query(available_students_select_sql)

    print("\nAll files to be checked: {}".format([x["name"] for x in files]))

    print("\nAll available students: {}\n".format(
        [x["user_id"] for x in students]))
    ntcf_amount = get_ntcf_amount(files)

    # probability that your work will be checked -> inverse ratings: ( 100 - rating
) /100
    one_percent_author = one_percent(files, True)
    add_prob_to_be_reviewed(files, one_percent_author)
    ntcf = choose_ntcf(files, ntcf_amount)
    one_percent_reviewer = one_percent(students)
    # probability that you will check
    add_prob_to_review(students, one_percent_reviewer)
    ntcf_checkers = choose_ntcf_checkers(students, ntcf_amount)
    # print("These students:\n{}".format(ntcf_checkers))
    reviewer_file = assoc_checkers_ntcf(ntcf, ntcf_checkers)
    pprint("Reviewer and task that he will check: {}".format(reviewer_file))
    teacher_file = [x for x in files if x not in ntcf]
    return {"student_check": reviewer_file, "teacher_check": teacher_file}


if __name__ == "__main__":
    main()
```

Додаток Г

Калібрація рейтингів

```
def get_cosine(vec1, vec2):
intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])
    sum1 = sum([vec1[x] ** 2 for x in list(vec1.keys())])
    sum2 = sum([vec2[x] ** 2 for x in list(vec2.keys())])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator


def text_to_vector(WORD, text):
    words = WORD.findall(text)
    return Counter(words)


def calibrate_rating(request, file_id, action_id):
    WORD = re.compile(r"\w+")
    expert_file = str(request.FILES['myfile'].read().decode("utf-8"))
    submitted_file = File.objects.get(id=file_id)
    initial_file = str(submitted_file.file)
    action = Action.objects.get(id=action_id)
    content_dict = json.loads(action.content)
    reviewer_file = str(content_dict["file"])
    reviewer = Student.objects.get(user=action.by_field.id)
    author = Student.objects.get(user=submitted_file.author_id)
    author_rating = float(author.rating)
    reviewer_rating = float(reviewer.rating)

    expert_vector = text_to_vector(WORD, expert_file)
    author_vector = text_to_vector(WORD, initial_file)
    review_vector = text_to_vector(WORD, reviewer_file)

    expert_author = get_cosine(expert_vector, author_vector)
    expert_review = get_cosine(expert_vector, review_vector)
    author_review = get_cosine(author_vector, review_vector)

    print("expert_author: ", expert_author)
    print("expert_review: ", expert_review)
    print("author_review: ", author_review)

    if expert_review > expert_author:
        e_r_angle = expert_author - expert_review
        sigma = 1 - e_r_angle
```

```
        sigma1 = author_review
        reviewer_rating += ((reviewer_rating * sigma) +
                            (reviewer_rating * sigma1)) % 100
        author_score = (int(content_dict["COMPARISON
RESULTS"]["grade"]) + 3*int(
            request.POST.get('score')) + int(action.score)) // 3
        reviewer.rating = reviewer_rating
        reviewer.save()
    elif expert_review < expert_author:
        author_rating += (expert_author * 10) % 100
        author_score = (int(content_dict["COMPARISON
RESULTS"]["grade"]) + 3*int(
            request.POST.get('score'))) // 3
        author.rating = author_rating
        author.save()
    print("Author score for the assignment: ", author_score)
    return author_score
```