КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ НАВЧАЛЬНИХ КУРСІВ ІЗ СИСТЕМОЮ ПЕРЕВІРКИ ТЕКСТІВ ПРОГРАМ НА ПЛАГІАТ

Виконав студент 4 курсу
Кліщ Андрій Іванович

(підпис)

Науковий керівник:
доцент
Панченко Тарас Володимирович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент (підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«____»_____202__р.
протокол №_____

Завідувач катедри

Нікітченко М.С. (підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 40 сторінок, 28 ілюстрацій, 24 джерела посилань.

АНТИПЛАГІАТ ДЛЯ ТЕКСТІВ ПРОГРАМ, РОЗРОБКА ВЕБ САЙТУ, СИСТЕМА ПІДТРИМКИ ВИКЛАДАННЯ КУРСУ, МЕТОД НАЙДОВШОГО СПІЛЬНОГО ПІДРЯДКА, МЕТОД ШИНГЛІВ.

Об’єктом роботи є процес організації навчальних курсів з можливістю перевірки лабораторних робіт на плагіат. Предметом роботи є розробка веб-додатку для організації навчальних курсів з анти плагіат системою.

Метою роботи є розробка веб-застосунку, що надасть систему управління навчальними курсами з можливістю вчителем перевіряти програмні коди учнів на анти плагіат.

Методи розроблення: аналіз існуючих систем, розробка веб за стосунку, аналіз існуючих алгоритмів виявлення плагіату, розробка анти плагіат системи.

Інструменти розроблення. Для розробки системи було використано мову програмування Java та Spring Framework. Під час розробки в якості інтегрованого середовища було використано Intellij IDEA.

Результат роботи. Було досліджено уже існуючі веб-сайти для організації навчальних курсів, проведено аналіз уже існуючих методів перевірки на анти плагіат, розроблено в команді веб додаток для організації навчальних курсів, розроблено програму для перетворення коду до стандартного вигляду та програму для перевірки двох текстів програм на анти плагіат, інтегровано дану програму у веб-додаток.

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ 7
1.1 Існуючі системи для організації навчальних курсів 7
1.2 Системи для перевірки текстів на плагіат 8
РОЗДІЛ 2 ПЛАГІАТ І СПОСОБИ ЙОГО ПРИХОВУВАННЯ 11
2.1 Загальна інформація 11
2.2 Плагіат програмних текстів 12
РОЗДІЛ 3 АНАЛІЗ АЛГОРИТМІВ ПЕРЕВІРКИ НА ПЛАГІАТ 17
3.1 Основні алгоритми перевірки на плагіат 17
3.2 Вибір алгоритмів для системи 19
3.3 Метод пошуку найдовшого спільного підрядка 19
3.4 Метод шинглів 21
РОЗДІЛ 4 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОШУКУ ПЛАГІАТУ 23
4.1 Етап підготовки лексичного словника і нормалізації коду 23
4.2 Реалізація алгоритмів перевірки на плагіат. 25
РОЗДІЛ 5. ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ КУРСІВ 29
5.1 Етап проектування 29
5.2 Огляд інтерфейсу користувача та взаємодія з ним 30
ВИСНОВКИ 37
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 39
ДОДАТОК А 41
ДОДАТОК Б 47

ВСТУП

Актуальність роботи та підстави для її виконання. Сучасні навчальні заклади щоденно працюють з дуже великою кількістю людей. Кожного дня працівникам даних організацій приходиться слідкувати за успішністю учнів, перевіряти їх домашні і практичні роботи, спілкуватись, заповнювати різноманітні документації. Тому вчителям важко услідкувати за навчальним процесом. На допомогу освіті приходять автоматизовані системи.

Автоматизація освітнього процесу значно полегшує навчання для студентів та школярів і робочий процес для викладачів. Наприклад, уся важлива інформація знаходиться в готових папках із зручним доступом, виконані і нові роботи можна сортувати, загальні оцінки учнів вираховуються автоматично, записуються результати іспитів, тощо. Управління системою відбувається шляхом створення курсу викладачем. Студенти приєднуються до своєї групи(класу), і вчитель контролює виконання завдань(встановлення кінцевих термінів, виставлення оцінок, послідовність тем) .

Оскільки в еру карантину таких систем появилось дуже багато, ще одним завданням стало добавити певну особливість у проект. Цією особливістю стала система перевірки програмного тексту на схожість (у подальшому плагіат). На жаль, дана проблема стала дуже актуальною у наш час. Звісно, на відміну від наукових товариств, у студентів немає таких поганих намірів як крадіжка чужої інтелектуальної власності, адже їхньою ціллю є просте отримання оцінки. Але дана проблема переростає у більш глобальні, такі як зменшення рівня знань студентів та несправедливе оцінювання. Викладачі ж не можуть услідкувати за всіма роботами студентів, керуючись тільки власною пам’яттю і уважністю.

Звісно способи перевірки на плагіат є, але це зазвичай перевірка текстових файлів на ідентичність з різноманітними сайтами. Тобто їх можна адекватно використовувати тільки для обліку рефератів та доповідей.

Отже, у нас є дві різні системи , які якщо і об’єднювали , то поодинокі навчальні заклади. Тому **метою** проекту є розробка веб-сайту для керування навчальними курсами, де б викладач міг перевірити код лабораторних робіт студентів на плагіат. Для досягнення цієї мети було поставлено такі **завдання**:

- Дослідити веб-застосунки, робота яких заключається у організації навчальних курсів та перевірці текстів на плагіат.
- Дослідити уже існуючі методи перевірки текстів на подібність.
- Спроектувати та розробити систему для організації навчальних курсів.
- Розробити програмне забезпечення системи перевірки на анти плагіат.
- Інтегрувати систему в веб застосунок.

Об'єкт, методи й засоби розроблення. Об’єктом роботи є процес керування навчальними курсами з можливістю перевірки текстів лабораторних робіт учнів на плагіат. Предметом роботи є розробка веб застосунку для організації курсів з системою пошуку плагіату.

Перед роботою над сайтом було проведено аналіз уже існуючих подібних систем як в сфері навчання так і в сфері перевірки на плагіат. Були досліджені також уже існуючі методи перевірки як тексту так і програмного коду на подібність. Існуючі напрацювання та нові ідеї дали можливість набагато простіше розібратись в темі і виконати задачу.

Для реалізації програми було використано мову програмування Java. Для розробки веб-застосунку був використаний Spring Framework. Даний фреймворк бере на себе роль керування інфраструктурою проекту, що значно спрощує роботу програміста і дає йому сконцентруватися на самому додатку.

Під час розробки продукту використовувалося інтегроване середовище Intellij Idea. Воно має широку підтримку мови Java та хорошу взаємодію з git. Розробник надає безкоштовну ліцензію для студентів вищих навчальних закладів.

Можливі сфери застосування. Розроблений програмний продукт може застосовуватись для полегшення організації навчальних курсів на факультетах пов’язаних з комп’ютерними науками,

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ

1.1 Існуючі системи для організації навчальних курсів

1. Google Classroom [1].

Веб сервіс для надання послуг навчальним закладам. Значно спрощує обмін файлами між учнями і вчителями шляхом об’єднання у собі різних інших веб-систем від Google (див. рис. 1).

Рисунок 1 Логотип Google Classroom

2. Moodle [2].

Дослівний переклад абревіатури звучить так - модульне об'єктно-орієнтоване динамічне навчальне середовище (див. рис. 2). Система для організації навчального процесу. Підходить для організації дистанційних курсів, очного навчання , тощо.

Рисунок 2. Логотип системи Moodle

3. ILIAS [3].

Програмне забезпечення для підтримки навчального процесу. Використовується переважно у німецьких закладах (див. рис. 3).

Рисунок 3. Логотип системи ILIAS

4. Unicraft [4].

Система для створення курсів та автоматизації навчання в компаніях (рис. 4).

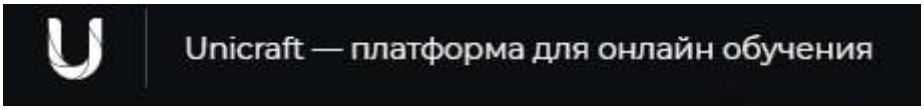

Рисунок 4. Логотип Unictaft

Отже, в інтернеті існує уже багато веб-систем для організації навчального процесу. Слід зазначити, що система буде більш спрямована на використання факультетом комп'ютерних наук та кібернетики, і також матиме деякі особливості (як система пошуку плагіату).

1.2 Системи для перевірки текстів на плагіат

Більшість сайтів з перевірки тексту на ідентичність[7] є теж однотипними. Зазвичай вони дають можливість порівняти ваш документ з текстами , що є на різноманітних сайтах, а також відшукують так звану «воду» (рис 5). Загалом вони виконують свою роботу добре, але їх проблема полягає в тому , що вони не порівнюють два файли між собою. Тобто перевірку серед обмеженого кола робіт вони не виконують. Для наочності наведемо приклади декількох таких сайтів.

Серед програм, які потрібно установлювати для перевірки на подібність текстів виділяють Advego Plagiatus[5] та Etxt Antiplagiat[6]. Перша відрізняється своєю більшою швидкодією, друга перевіряє текст більш ретельніше і дає нижчі результати унікальності. Обидві програми прості в користуванні.

Одним з найбільш популярних сервісів є Content-watch[7]. Має власний алгоритм перевірки на плагіат і перевірки можна робити без реєстрації,

звісно, з певними обмеженнями (довжина тексту та кількість перевірок). Сайт має одну особливість. Він може ігнорувати певний набір сайтів, які може вказати користувач.

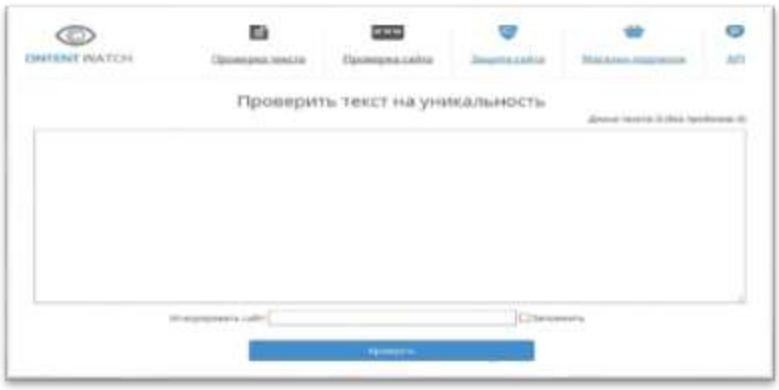

Рисунок 5 Приклад інтерфейсу одного з сайтів для перевірки тексту на унікальність.

Одним з найбільш просунутих сервісів вважається на даний момент Text.ru[8]. У нього є маса особливостей і переваг над простими сайтами, а саме може розрізняти перестановки різних слів чи фраз, находить плагіат якщо змінювати кожне п’яте чи четверту слово, навіть визначає зміну відмінків і часів у словах. Сайт є безкоштовним, швидким та вільним у доступі.

Прикладом сайту, що перевіряє не просто тексти, а саме документи на плагіат можна назвати StrikePlagiarism[9]. В системи є своя база даних і він може порівнювати файли у форматах DOC, ODT, TXT, PDF.

Для перевірки дипломних і курсових робіт у університетах часто використовуються програма Unicheck[10]. Дана система створена українськими розробниками і використовується не тільки Україною, але й Іспанією, США і іншими. Має масу можливостей: може формувати звіти по плагіату у різноманітних форматах, перевіряти 100 робіт використовуючи Інтернет та 35 робіт використовуючи внутрішню базу, вилучає правильно процитовані рядки та текст з вказаним джерелом. Також сервіс може

інтегруватися з різноманітними навчальними онлайн сервісами, такими як Moodle, Sakai, Canvas і багато інших. Крім цього система надає хороший інтерфейс викладачам, які мають багато можливостей, включаючи усування тексту, який вони вважають не плагіатом або коментування певних ділянок робіт.

Отже, щодо анти плагіат систем, що є у відкритому доступі, можна зробити висновок, що вони не підходять для перевірки лабораторних робіт студентів. Більшість є не достатньо якісними, або платними, а ті, що адекватно справляються зі своєю роботою працюють переважно тільки з простими текстами, а не програмним кодом, тому студент може багатьма способами зменшити рівень плагіату у своїй роботі добавляючи коментарі, зайві рядки і так далі.

РОЗДІЛ 2 ПЛАГІАТ І СПОСОБИ ЙОГО ПРИХОВУВАННЯ

2.1 Загальна інформація

Плагіат – це репрезентація виразів, думок чи ідей іншого автора як своїх [11]. Сучасна концепція плагіату як аморального явища, що спирається на оригінальність, виникла у 18 столітті разом з рухом романтизму. Виявлений плагіат зазвичай підлягає різноманітним покаранням, таким як відсторонення, штраф, виключення з навчального закладу, позбавлення волі, адже плагіат є порушенням академічної доброчесності та журналістської етики[12].

Звісно, у сучасному світі плагіат сам по собі не є злочином, хоча і може каратися судом на рівні з фальсифікацією, порушенням авторського права та/або моральних норм. Інколи плагіат розглядають як шахрайство [13], при використанні чужої роботи задля отримання академічного кредиту. Взагалі, хоч плагіат і авторське право дуже подібні поняття, вони не є рівноправними. Плагіат є ширшим поняттям, що визначається законом про порушення авторського права.

У різних країнах плагіат визначається і розглядається по різному. Наприклад, у деяких країнах плагіат є повною протилежністю до академічної доброчесності і розглядається як лестощі до професіонала [14]. Також різниця у визначенні плагіату або ж його рівня інколи може різнитись на рівні навчальних закладів.

Серед всіх визначень плагіату можна виокремити поняття академічного плагіату. Найкращим означенням цього поняття є версія Тедді Фішмана [15]: “використання ідей, концепцій, структур, текстів без належного визнання джерела , щоб отримати вигоду в обстановці, де очікують оригінальність”. Він також визначив п’ять властивостей плагіату, а саме використання робочих продуктів чи ідей, належність цих ідей до іншої особи чи джерела,

що ідентифікує певну особу, відсутність припису цих джерел, від роботи очікується оригінальність, за роботу є вигода і не обов’язково грошова.

2.2 Плагіат програмних текстів

Звісно, плагіат програмних кодів не є однозначним явищем, адже майже всі стандартні задачі, які зазвичай і є лабораторними, уже запрограмовані до нас. Тим більше якщо задача є невеликою , то створення іншого коду для неї призводить тільки до погіршення його якості і лаконічності. Проте, варто зауважити, студенти не є професіоналами,що здатні зразу найти найпростіше і найправильніше рішення для кожної задачі.

Для лабораторних з більшим обсягом структура коду повинна все таки у різних студентів сильно відрізнятись. Але ж студенти спілкуються між собою. Тому, варто сказати, що навіть якщо практичні подібні, в першу чергу потрібно перевірити теоретичні знання студента, і пройтися по самому коду, якщо учень буде губитись, то тоді вже можна говорити про плагіат. Проте метою системи все ж таки є перевірка коду, а не студента, щоб вчитель міг зрозуміти за ким потрібно більше спостерігати.

Найлегшим способом є здача чужої лабораторної без змін. У такому випадку, вчитель і сам може зрозуміти, що студент не добросовісно виконує свою роботу, адже здача лабораторних переважно відбувається в зазначений день усією групою. Слід все ж таки сказати, що викладачі не можуть просто фізично продивитись і запам’ятати всі лабораторні студентів. До того ж, студенти часто можуть здавати минулорічні роботи. Щоб дізнатись чи тут лабораторна унікальна чи ні, вже треба володіти величезною базою всіх лабораторних або ж міняти умови кожного року, що далеко не завжди є можливим.

Інколи студенти все ж таки пробують замаскувати чужий код під свій і надалі у цьому розділі ми розглянемо, найпопулярніші способи. Для наочності візьмемо просту частину Java коду (рис.6). Причиною цьому є те,

що анти плагіат систему працює тільки з С-подібними мовами і серед таких мов Java є найбільш поширеною.

```
private int Example(int n){
    int sum = 0;
    for(int i = 0; i<n; i++){
        sum++;
    }
    return sum;
}
```

Рисунок 6. Приклад простого коду на Java

Одним з найпопулярніших способів приховування крадіжки чужого коду є зміна назв змінних, функцій, методів , класів , тощо (рис. 7). Такий метод до того ж є і найпростішим, адже сучасні IDE позволяють проводити ре факторинг коду кількома кліками мишки. Ще однією причиною популярності даного методу є те, що такий спосіб не тільки не потребує знань у темі лабораторної, а навіть знання мови.

```
private int NewName(int maxValue){
    int result = 0;
    for(int index = 0; index<maxValue; index++){
        result++;
    }
    return result;
}
```

Рисунок 7. Замаскований плагіат шляхом перейменування змінних.

Ще одним менш популярним і ефективним є заміна ключових слів мови програмування (наприклад, модифікаторів доступу та типів змінних) (рис. 8). Він менше скриває подібність коду, до того ж при незнанні роботи програми чи мови це може приводити до помилок або виключень.

Найбільш ефективним для маскування, але потребуючим хороших знань у програмуванні методом є заміна різних операторів та конструкцій на

аналогічні. Очевидною є заміна циклів for на while (рис.9). Менш явними є заміна різноманітних конструкцій на потік у Java чи Linq запити у C#. Звісно,

```
public long Example(long n){
    long sum = 0;
    for(int i = 0; i<n; i++){
        sum++;
    }
    return sum;
}
```

Рисунок 8. Замаскований плагіат шляхом зміни ключових слів мови

з таким підходом постає очевидне питання: чи є дана програма плагіатом з настільки суттєвою зміною коду. І у даному випадку, можна навести два аргументи. По-перше, якщо програма є великою, то зміна певних конструкцій не робить програму іншою. По-друге, мова йде про навчальні лабораторні роботи. Якщо студент поміняв лабораторні без знань, то це свідчить, що він хороший програміст, а не вивчив тему, яку потребує від нього освітній план.

```
private int Example(int n){
    int sum = 0;
    int i = 0;
    while(i<n){
        sum++;
        i++;
    }
    return sum;
}
```

Рисунок 9. Замаскований плагіат шляхом зміни циклу for на while

Поганою практикою приховування подібності текстів є захаращування коду непотрібними рядками, змінними, тощо (рис. 10). Цей метод є найгіршим у багатьох сенсах. З одного боку, можна нагромадити дуже багато непотрібного коду, написати безліч непотрібних бібліотек, добавити

непотрібні функції і це виглядатиме як зовсім нова програма. Але якщо вчитель буде опитувати студента по коду, навіть якщо студент знатиме теорію, йому буде важко пояснити значення непотрібних рядків. До того ж, сучасні IDE виділяють певним чином невикористані функції чи бібліотеки, що легко можна помітити. Ще одним недоліком цього методу є те, що студент дуже псує оригінальний код. Якщо інший студент спробує застосувати цей метод до коду, що утворився в результаті, його код буде зіпсований ще більше. Таким чином, через декілька ітерацій код програми буде неможливо прочитати.

```
private int Example(int n){
    int sum = 0;
    int i = 0;
    int x = 0;
    int y = 1;
    int z = 15;
    for(int i = 0; i<n; i++){
        sum++;
        z = x+y;
    }
    return sum;
}
```

Рисунок 10. Приховування коду шляхом додавання непотрібних рядків.

Подібним за логікою, але можливо навіть в плані практики написання, редагування чи видалення коментарів до програмного коду(рис. 11). Цей метод вирішує зразу дві проблеми. По-перше, він допомагає студенту поміняти вигляд програми, а ,по-друге, він дає інформацію студенту про код, що допоможе йому здати практичну роботу.

І останнім способом приховування однаковості коду є розбиття або об’єднання структур, класів чи методів (рис.12) . Ефективний метод, що іноді може дуже сильно поміняти загальний вигляд програми. Цікаво є те, що інколи цей метод може навіть покращувати якість коду.

```java
public int Example(int n){
    int sum = 0;//наш результат
    for(int i = 0; i<n; i++){ //цикл для обрахування результату
        sum++;//на кожній ітерації додаєм один
    }
    return sum;//повертаємо результат
}
```

Рисунок 11. Приховування плагіату шяхом додавання коментарів

```java
public int Example(int n){
    int sum = 0;
    int result = Cycling(n,sum);
    return result;
}
private int Cycling(int n, int sum){
    for(int i = 0; i<n; i++){
        sum++;
    }
    return sum;
}
```

Рисунок 12. Приховування плагіату шляхом розбиття метод на два інші

Слід зазначити, що якщо студент сильно переймається за те ,щоб плагіат не помітили, він буде комбінувати ці методи і розбиратись в програмі. Також варто сказати, що якщо студент дуже сильно поміняв програму, розібрався у ній і розібрався у теорії, то можна вже сказати, що це вже його робота.

Отже, в даному розділі було розглянуто, що таке плагіат як в загальному, так і в розумінні програмування. Також описано як студенти можуть приховувати свою не доброчесність. Не всі способи можна визначити методами пошуку анти плагіату, і для деяких потрібно удосконалювати існуючі, але це буде більш докладно пояснено в наступних розділах. Також варто зазначити, що однакові невеликі лабораторні звісно завжди матимуть подібні розв’язки, тому головний «вердикт» буде за викладачем.

РОЗДІЛ 3 АНАЛІЗ АЛГОРИТМІВ ПЕРЕВІРКИ НА ПЛАГІАТ

3.1 Основні алгоритми перевірки на плагіат

Методів для перевірки текстів на плагіат є безліч. Загалом їх можна розділи на декілька видів:

- звичайна перевірка текстів на подібність;
- перевірка текстів з передчасною токенізацією тексту;
- побудова синтаксичних дерев та графів;
- перевірка метрик [16].

Звичайна перевірка на подібність являє собою порівняння рядків двох текстів. Даний метод є більш актуальним для перевірки не програмних текстів, адже він надає дуже неточні результаті при будь-якій зміні тексту коду. Проте плюсом даного методу є його швидкість.

Удосконаленим методом простого порівняння кодів є використання суфікс них дерев. Суфіксне дерево – це бор(структура даних, що зберігає у собі асоціативний масив, ключами якого є рядки), що містить усі суфікси декількох рядків. Дозволяє виявити чи входить рядок w у рядок t за час $O(|w|)$, де |w| довжина рядка w [17]. Проходячи по цьому дереву можна виявити чи є однакові рядки у двох текстах.

В основі методу побудови дерев лежить синтаксичне дерево. Синтаксичне або дерево деривації - це позначене, скінченне і орієнтоване дерево, яке позначає певну синтаксичну структура рядка відповідно до деякої безконтекстної граматики [18]. У програмуванні широко використовується у різноманітних парсерах. Внутрішні вершини дерева співставлені з відповідними операторами мови програмування, а листя з відповідними операндами. Перевірка на плагіат в синтаскиних деревах може нормалізувати подібні за смислом умовні оператори та виявити еквівалентні конструкції.

Проблема полягає у тому, що складність алгоритму сильно збільшується при перевірці більших проектів і реалізація таких алгоритмів є набагато важчою.

Для аналізу на основі графів використовують граф викликів. Граф викликів - це орієнтований граф, який зображує виклики між функціями в комп'ютерній програмі. Зокрема, кожен вузол являє собою деяку процедуру, а кожна дуга (f,g) показує, що процедура f викликає процедуру g. Може визначати подібності в двох програмах набагато вищого рівня, ніж звичайний пошук однакових рядків [19]. Ще однією перевагою цього методу є те, що даний граф знаходить конструкції, які ніколи не використовуються програмою. Проблемою знову ж таки є складна реалізація і великі затрати за складністю та часом.

Використання метрик базується на фіксуванні «балів» на певних ділянках коду за певними критеріями. Це можуть, наприклад, бути однакова кількість коментарів, викликів функції, об'явлення змінних і так далі. Переваг, як і недоліків, у цьому методі є багато. З плюсів можна виділити те, що метрики легко обчислити і їх легко порівняти. Головною ж проблемою є те, що вони можуть показувати завеликий рівень плагіату. Два фрагменти коду з однаковою оцінкою на певному наборі метрик можуть давати зовсім різні результати.

Токенізація передбачає заміну коду програми на набір токенів відповідно до певного лексичного словника. Зазвичай замінюються ключові слова. Цей метод відкидає коментарі, ідентифікатори, пробіли, що робить систему більш надійною до простих змін коду. Більшість академічних систем працює саме на цьому рівні.

Звісно, найефективнішими методами є гібридні. Наприклад, можуть поєднувати у собі швидкість простих методів і якість складних, як, наприклад поєднання дерева суфіксів та дерева синтаксичного аналізу.

3.2 Вибір алгоритмів для системи

Оглянувши і проаналізувавши способи приховування плагіату та алгоритми для його пошуку, можна обрати найоптимальніші варіанти знаходження подібності для розроблюваної системи.

Одразу можна сказати, що методи, які просто використовують порівняння рядків не розглядатимуться, адже навіть найпростішими перейменуваннями змінних можна значно знизити рівень плагіату.

Найбільш глибоким та смисловим є аналіз програмних текстів на основі синтаксичних дерев та графів викликів. Але окрім вище сказаних великих затрат часу та пам’яті , у даних способів є ще одна проблема. Різні варіанти лабораторних робіт зазвичай відрізняються тільки вхідними. Очевидною проблемою тоді стає те, що роботи студентів, що працювали незалежно над завданням, будуть мати подібні структури. Це може призвести до перебільшення реального значення рівня плагіату. Цього можна було б уникнути якби кожному студенту давали іншу умову лабораторної, але тоді б і не виникло потреби перевірки на плагіат, до того ж це багато зайвої роботи для вчителів.

Тому лишається два варіанти, метрики або токенізація. Метрики - поганий варіант по причинам описаним в минулім підрозділі. Методи ж з використанням лексичних словників є найкращим для цієї роботи варіантом. Вони не затратні, до того ж, словники не так важко робити, адже є групи подібних мов. Також цей спосіб покриває достатню велику частину методів схову крадіжки коду. Серед методів токенізаці були обрані метод найбільшого спільного підрядка та шинглів.

3.3 Метод пошуку найдовшого спільного підрядка

Метод найдовшого спільного підрядка полягає у тому, щоб знайти найдовший рядок, який є підрядком для двох чи більше рядків.

Формально, найдовшим спільним підрядком рядків s_1, s_2,…s_n називається рядок w*, для якого виконується умова ||w*|| = max ({||w|| | w ⊂ s_i, i = 1,…n}), операція w ⊂ s_i позначає те, що рядок w є підрядком рядка s_i.

Для знаходження найдовшого спільного підрядка двох рядків s_1 і s_2, довжина яких m і n відповідно, необхідно заповнити матрицю A_{ij} розміром (m+1) x (n+1), дотримуючись алгоритму (3.1).

$$\begin{cases} A0j = Ai0 = 0, & j = 0 \dots n, \quad i = 0 \dots m, \\ Aij = 0, \quad s1[i] \neq s2[j], & i \neq 0, \quad j \neq 0, \\ Aij = Ai-1, j-1, \quad s1[i] = s2[j], & i \neq 0, \quad j \neq 0. \end{cases} \quad (3.1)$$

Найбільше число A_{uv} в матриці являє собою довжину найдовшого спільного підрядка. Сам підрядок знаходиться за наступним правилом: s_1[u-A_{uv}+1]…s_1[u] та s_2[v-A_{uv}+1]…s_2[v].

Рівень плагіату визначається відношенням отриманої довжини спільного підрядка до довжини рядка, який досліджуємо на плагіат (3.2).

$$r(s1, s2) = 1 - \frac{|LCS(s1, s2)|}{|s1|} \quad (3.2)$$

Для наочності наведемо приклад. Нехай у нас є два рядки SUBSEQUENCE та SUBEUENCS. У заповненій матриці А (рис. 14) ми побачимо, що довжина найбільшого спільного підрядка буде 4, а сам рядок буде UENC [20].

		S	U	B	E	U	E	N	C	S
	0	0	0	0	0	0	0	0	0	0
S	0	1	0	0	1	0	0	0	0	0
U	0	0	2	0	0	0	1	0	0	0
B	0	0	0	3	0	0	0	0	0	0
E	0	0	0	0	1	0	0	1	0	0
U	0	0	1	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	2	0	0
N	0	0	0	0	0	0	0	0	3	0
C	0	0	0	0	0	0	0	0	0	4
S	0	1	0	0	1	0	0	0	0	0

Рисунок 13. Матриця А.

Тепер, оскільки довжина першого рядка 11, визначимо по формулі рівень плагіату, який складатиме приблизно 36%.

Варто також додати, що складність даного алгоритму складає O(m*n), Також слід, додати, що у ході реалізації алгоритм було покращено.

3.4 Метод шинглів

Вперше ідею шинглів для знаходження дублікатів запропонував Уді Манбер у 1994 році, а продовжив і удосконалив його ідею Андрій Бродер у 1997 році.

В даному алгоритмі найчастіше виділяють п’ять основних етапів:

1) Канонізація тексту.
2) Розбиття канонізованого тексту на шингли.
3) Визначення геш значень шинглів за допомогою 84х статичних функцій.
4) Вибір випадкових 84 значень контрольних сум.
5) Порівняння і отримання результату.

На першому етапі, текст приводиться до певної єдиної нормальної форми. Часто на цьому етапі текст очищається від «сміття», який ніяк не впливає на дублювання текстів. Зазвичай це розділові знаки.

Як тільки текст нормалізовано, його розбивають на так звані шингли. Шингл (з англійської луска, черепиця, галька) - це вибрані підпослідовності слів з тексту. З усіх порівнюваних текстів потрібно виділити підпослідовності слів, що йдуть один за одним в довжину шингла. Якщо текст розбивається таким чином, то отримаємо шинглів кількістю в n-l+1, де n- це кількість шинглів, а l- це довжина шингла.

Продуктивність алгоритму напряму залежить від кількості шинглів. Збільшення кількості шинглів характеризується ростом кількості операцій,

що може критично позначитися на продуктивності. Тому беруться не шингли, а контрольні суми, які обраховуються статичними геш функціями.

Наступним етапом є знаходження випадкової вибірки контрольних сум для обох текстів. Прикладом може бути вибір мінімального значення з кожного рядка для обох текстів.

Фінальним етапом є порівнювання двох вибірок. Результатом є відношення однакових елементів у двох вибірках.

Рівень плагіату між двома текстами A і B вираховуються за формулою (3.3), де |A| - це довжина вибірки A [21].

$$r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} \quad (3.3)$$

Отже, було обрано методи, що використовують токенізацію, а саме метод найбільшого спільного підрядка та метод шинглів. Звісно, дані методи були удосконалені у ході їх аналізу та розробки програмного коду для них, про що мова піде у наступному розділі.

РОЗДІЛ 4 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОШУКУ ПЛАГІАТУ

4.1 Етап підготовки лексичного словника і нормалізації коду

Для того, щоб нормалізувати код, потрібно привести його до якогось певного узагальненого вигляду. Для цього потрібно було створити певний лексикографічний словник, що для мов програмування зробити не важко, адже у більшості мов є певний набір ключових слів, який до того ж може повторюватись і означати те ж саме для декількох мов.

Для мого словника було обрано С-подібні мови, а саме Java [22], C++ та C#[23]. Дані мови є подібні між собою, мають багато однакових ключових слів і ідентифікаторів, що дає змогу написати для них спільний словник для нормалізації. Також варто зазначити, що дані мови програмування є найбільш поширеними на факультеті кібернетики при здачі лабораторних робіт.

Словник було зроблено наступним способом. Кожному ключовому слову у відповідність була поставлена певна буква алфавіту. Крім того, якщо декілька ключових слів подібні між собою за сенсом, то їх було об’єднано в групи і їм у відповідність поставлено одну і ту ж букву. Це зроблено для того, щоб при маскуванні плагіату шляхом зміни подібних ключових слів рівень унікальності не збільшувався.

Словник записаний у текстовому документі, завдяки чому його можна легко змінювати і доповнювати. Крім цього, граматику можна легко зчитати використовуючи програмні інструменти мови Java. Також у текстовий файл записано всі розділові знаки, що можуть використовуватися у програмних кодах.

Спочатку для нормалізації коду потрібно його очистити від тексту, що не повинен впливати на унікальність. Це в першу чергу коментарі і значення змінних. Також видаляється весь текст, що знаходиться в лапках. Для кожного видалення написаний окремий методі і видалення відбувається

поступово. Таким чином додавання коментарів чи зайвого тексту для приховування плагіату буде виявлено.

Після видалення непотрібного тексту, потрібно замінити всі переходи на наступний рядок на пробіли, щоб можна було легко знаходити ключові слова. Після цього всі ключові слова та їх значення у словнику записуються у мапу.

Після того як знайдено всі слова, що є у словнику, потрібно знайти слова, яких не має у мапі. Якщо слово примикає до ключового, що стоїть перед ним, то йому у відповідність ставиться ідентифікатор цього ключового слова; якщо після нього, то відповідно робиться те саме. Це робиться з наступних причин. Якщо кожну змінну ідентифікувати окремим символом, то унікальність коду зазвичай буде вищою, ніж це є насправді. Якщо ж для всіх змінних зробити один універсальний ідентифікатор, то буде протилежна проблема – унікальність часто буде занадто низькою. Тому, виходячи з факту, що змінна знаходиться біля свого типу, було прийняте рішення описане вище.

Звісно, у деяких мовах є такий тип змінної як var, який не каже про тип змінної. Для вирішення цієї проблеми всі числові значення замінюються на певний ідентифікатор (наприклад, #number#), і якщо перед змінною стоїть ідентифікатор типу, що не вказує вид змінної, то майбутні дії залежать від того чи наступне значення є числовим. Якщо так, то йому приписується ідентифікатор числових типів, якщо ні, то текстових.

Наступним кроком є видалення усіх пробілів та заміна усіх слів, що добавлено у мапу на ідентифікатори зі словника. Результатом є нормалізований код.

Наведемо приклад. Для коду зображеного на рисунку 14, результатом токенізації буде наступний вираз:
“doo{dЮЮЮ=;ddqq(Ю[]q){q()}ddqq(Юq){Юq=q(q);ЬIq=ЬI.q(Ю);ЬIq=q.q(q);c(q.c()){q=q.c(q.c(),q.c().c(n,q.c().c()));}c.D.D(q);}}”.

```
public class Part1 {
    static final String REGEX_WORD = "[A-ZA-ЯЇІЄЭЁЩa-za-яёэіїєщ]{4,}";
    public static void main(String[] args) {
         convert("part1.txt")
    }
    public static void convert(String fileName){
        String str = getInput(fileName);
        Pattern pattern = Pattern.compile(REGEX_WORD);
        Matcher matcher = pattern.matcher(str);
        while (matcher.find()){
            str = str.replace(matcher.group() , matcher.group().substring(2, matcher.group().length()));
        }
           System.out.println(str);
    }
}
```

Рисунок 14. Програмний код для прикладу нормалізації.

Як можна побачити результат нормалізації є набагато коротшим текстом, що також позитивно впливає на продуктивність розроблюваної системи. Уже у такому нормалізованому вигляді програмні коди і будуть перевірятися на плагіат. Код програми для нормалізації тексту можна побачити у додатку А.

4.2 Реалізація алгоритмів перевірки на плагіат.

Загалом алгоритм знаходження найбільшого спільного підрядка не було змінено. Проте залишилась одна проблема. Алгоритм дуже не стійкий до ставлення непотрібних рядків чи символів у текст. Наприклад, у нас є текст “you are my friend”, і такий же текст у якості плагіату. Якщо вставити посередині будь-яке слово і текст наприклад стане таким: “you are hi my friend”, то рівень унікальності збільшиться удвічі. Якщо добавити ще два слова, то рівень унікальності зросте ще удвічі.

Вирішення цієї проблеми є досить простим. Після того як знайдено найбільший спільний підрядок, його буде видалено з нормалізованого коду, і відновлено пошук. Так поки не буде знайдемо всі рядки. Таким чином рівень плагіату буде дорівнювати відношенню довжини всіх спільних рядків до довжини тексту всієї програми.

Варто сказати, що шляхом нормалізації коду, розміри матриці A, зменшились у багато раз, що значно покращило продуктивність розроблюваного алгоритму.

Алгоритм шинглів змінений уже трохи більше. Найбільш очевидною зміною було те, що використовуються не статичні функції для знаходження гешу слова, а вбудований у клас String метод getHashCode(). Це не буде значною мірою впливати на результат, але спростить процес кодування алгоритму.

Ще однією важливою зміною стало те, що було вирішено в кінці не брати випадкові контрольні суми, а весь токенізований текст. Це робиться по причині того, що програмні тексти лабораторних не є дуже великими, до того ж у токенізованому вигляді. Ще однією причиною є те, що випадкові вибірки зазвичай дають не однаковий результат, що може значною мірою впливати на точність.

Також було вирішено замінити формулу обрахунку плагіату. В оригінальній формулі в знаменнику об'єднання двох текстів. Часто студенти можуть виконати тільки частину практичної роботи, тому в знаменнику формули було залишено лише текст програми, яку перевіряють на плагіат (4.1).

$$r(s_1, s_2) = \frac{|S(s_1) \cap S(s_2)|}{|S(s_1)|} \quad (4.1)$$

Основною проблемою методу шинглів був вибір розміру шингла. Очевидно, що розмір 1 був замалим для перевірки коду на плагіат, адже процент унікальності був би занадто малий. Це витікає з того, що набір символів, яким заміняється програмний код під час нормалізації є обмеженим. Тому постала задача знаходження конструкції, яка найчастіше зустрічається у С-подібних мовах.

Конструкція while не так часто зустрічається у сучасних програмних кодах, до того ж вона є не достатньо великою, та й параметром може бути

дуже різна кількість токенів. Конструкція if, хоч і є, мабуть, найчастішою в коді, теж не підходить з тих самих причин, що і while. Тому було прийняте рішення обрати конструкцію for.

Стандартним оголошенням циклу for є приблизно така конструкція: for(int i=0;i<n;i++){тіло циклу}. Після нормалізації код стане наступним: f(nn=n;n<n;n++){/*решта токенів*/}. Довжина токенізованого коду складає 16 символів. Отже, вирішено брати мінімальну довжину шингла у 17 символів.

Також було прийнято рішення, що довжина у 17 символів буде ще вказувати на мінімальну довжину підрядка у першому алгоритмі, що буде вказувати на плагіат. Таке рішення було прийняте виходячи з тих самих міркувань.

Після отримання результату було замічено, що у обох алгоритмів виходить різний результат по рівню плагіату. Зазвичай алгоритм шинглів надає менший результат, ніж метод найбільшого спільного підрядка. Для того, щоб вияснити причину такої поведінки наведемо простий приклад.

Варто сказати, що в даному прикладі будуть ігноруватися мінімальні розміри свідчення плагіату, які є в системі, оскільки приклади є малими і тільки для того, щоб наочно показати різницю у двох алгоритмах. Нехай уже є нормалізована програма P=”абвг”, та її плагіат K=”абдвг”. Використовуючи метод найбільшого спільного рядка, було вияснено, що є два спільні рядки “аб” та ”вг”. Таким чином рівень плагіату складатиме 80%. Тепер скористаємось методом шинглів. Множина шинглів для першого тексту буде наступна: {“аб”,”бв”,”вг”}. Для другої ж програми: {“аб”.”бд”,”дв”,”вг”}. Перетином двох множин буде {“аб”,”вг”}. За формулою рівня плагіату, отримано 50%. Звісно, реальні приклади є набагато більші за обсягом і різниця у процентах буде не такою великою, але варто пам’ятати, що вона є.

Отже, взявши дві невеличкі програми, було виведено на екран коефіцієнти плагіату (рис. 15). У прикладі одна програма є частиною іншої (приблизно третина). Тож можна сказати, що плагіат працює правильно.

Також варто зазначити, що якщо поміняти два файли з кодом місцями, то значення плагіату поміняється. У даному випадку він буде прямувати до 1. Це тому, що код меншої програми повністю міститься у більшій тому, можна сказати, що він повністю вкрадений. При перевірці двох документів з однаковим текстом рівень плагіату буде 1 (рис. 16). На зображеннях також показано середнє значення двох результатів.

```
Average value:
0.39124723247232474
Longest common string value:
0.4464944649446494
Shingles value:
0.336
```

Рисунок 15. Результат роботи системи перевірки на плагіат

```
Average value:
1.01
Longest common string value:
1.0
Shingles value:
1.02
```

Рисунок 16. Результат роботи системи з однаковими текстами.

Отже, хоча система не дає 100% точний результат рівня плагіату(оскільки два алгоритми показують різні значення), цього в повній мірі достатньо, щоб зрозуміти наскільки два тексти похожі між собою. Код програми для пошуку плагіату можна побачити у додатку Б.

РОЗДІЛ 5. ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ КУРСІВ

5.1 Етап проектування

Перш за все варто зазначити, що веб-частину було виконано у команді з п'яти осіб, як проект з навчального курсу. Також веб-частина є невід'ємною частиною роботи, адже система виявлення плагіату інтегрована у веб-застосунок, яким і будуть користуватись.

Для початку було продумано і розроблено різноманітні діаграми. Спочатку була створена use-case діаграма для того, щоб зрозуміти загальну структуру проекту і функціонал, який потрібно виконати. Після цього було ще створено декілька діаграм послідовності для того, щоб зв'язати струткуру проекту, яку ми отримали з use-case діаграми та структуру класів та методів у програмі.

Наступним етапом було створення інженерної специфікації. Вона у себе включала: мету розробки, опис прикладних компонентів, перелік вимог чинного законодавства, вимоги до технічної та інформаційної архітектури, до програмного забезпечення, показників навантаження, режимів функціонування, надійності, ергономіки, захисту інформації, патентної чистоти, лінгвістичного забезпечення, стандартизації та уніфікації. Також там описано склад , зміст і календарний план послуг з розробки системи.

Ще одним важливим етапом, була розробка бази даних. Схема розроблялась використовуючи MySQL Workbench. Для збереження даних по користувачах було створено таблицю users. Для організації курсів та навчальних процесів в них було створено такі таблиці: courses – таблиця курсів, tasks - таблиця завдань, documents - таблиця для збереження файлів. comments - таблиця для збереження коментарів до завдань, posts – таблиця постів.

Також слід зазначити, що була виконана формальна специфікація та верифікація для обрахунку балу за курс. У програмі використовувалась мова Dafny.

Отже, можна впевнено сказати, що етап проектування відбувся на високому (навіть професіональному) рівні. Були продумані всі етапи створення продукту та весь функціонал наперед, задля кращої організації проекту та праці над ним.

5.2 Огляд інтерфейсу користувача та взаємодія з ним

Веб-додаток було названо Smart Course. До головної сторінки (рис. 17) доступ є у всіх користувачів Інтернету. Натиснувши на кнопку Pomodoro можна перейти на веб-сайт з функціоналом Pomodoro timer. Також на головній сторінці є блоки з написом Something interesting. Під ними формуються випадкові статті з Вікіпедії [24]. Тобто, на головній сторінці присутній додатковий функціонал, який допомагає організувати краще свій час або ж дізнатись щось цікаве, якщо втомився працювати.

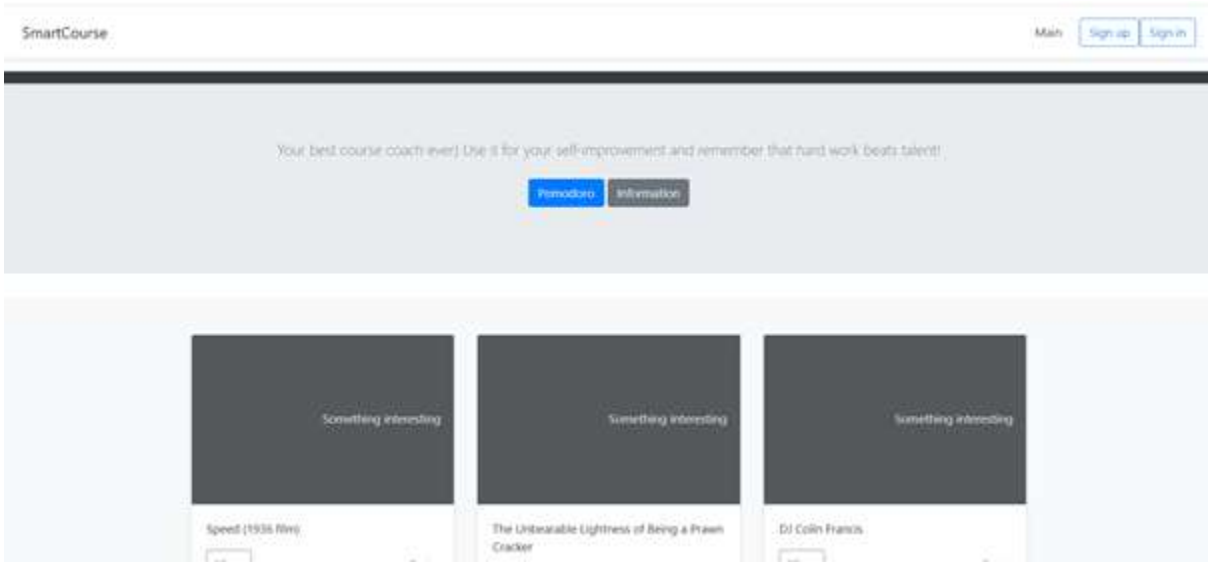

Рисунок 17. Головна сторінка веб застосунку

Щоб отримати доступ до всього функціоналу, потрібно зареєструватись як користувач. Для цього на панелі зверху є кнопка Sign up.

На всі поля реєстрації (рис. 18) є перевірка на коректність. Наприклад, ім’я не може мати менше, ніж 2 літери. Також є обмеження на електронну пошту: реєструватися можна тільки з доменом knu.ua. Пароль повинен мати як літери так і цифри для кращої конфіденційності.

Рисунок 18. Форма реєстрації

Для зареєстрованого користувача, на тій же панелі є кнопка Sign in, що дозволить йому зайти у систему. Для того, щоб увійти потрібно ввести свій логін і пароль. Двох користувачів з однаковим логіном бути не може.

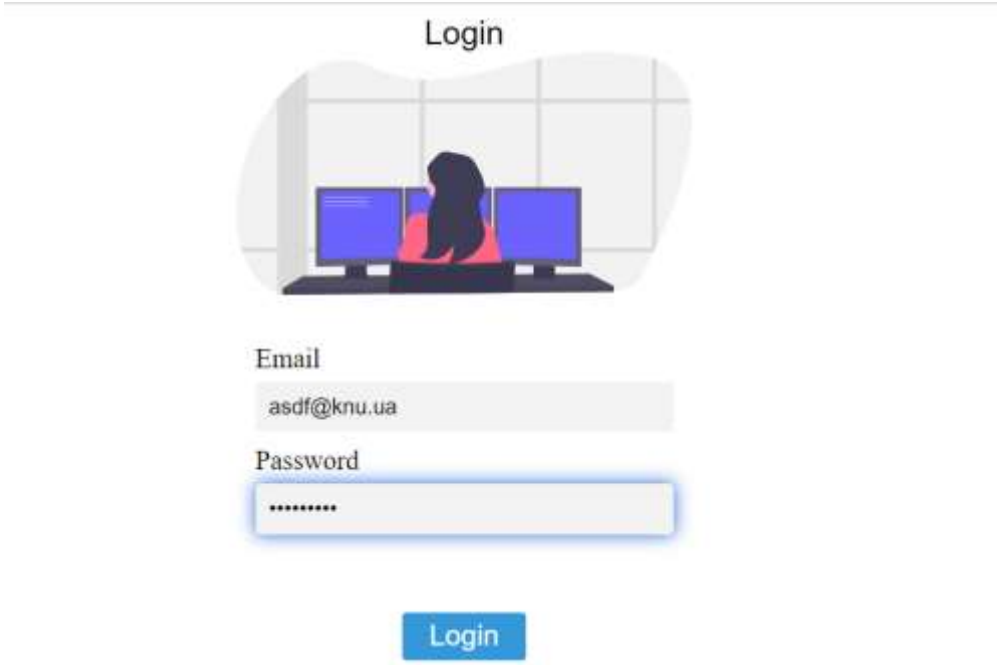

Рисунок 19. Форма логіну

Після реєстрації можна побачити головну сторінку з уже розширеним функціоналом (рис. 20). Також на головній сторінці знову ж таки буде додатковий інтерфейс у вигляді to do list-a , який заповнюється самостійно після отримання завдань на курсі. До того ж тепер на сторінці будуть кнопки для перегляду курсів або створення свого.

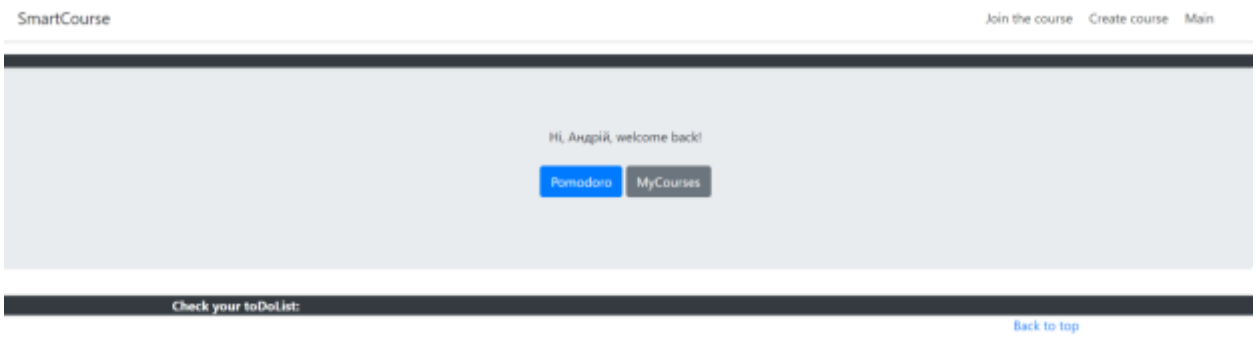

Рисунок 20. Головна сторінка після реєстрації.

Якщо перейти на сторінку створення (рис. 21), потрібно ввести назву курсу та натиснути “Course is ready!”. Створити свій курс може будь-який користувач.

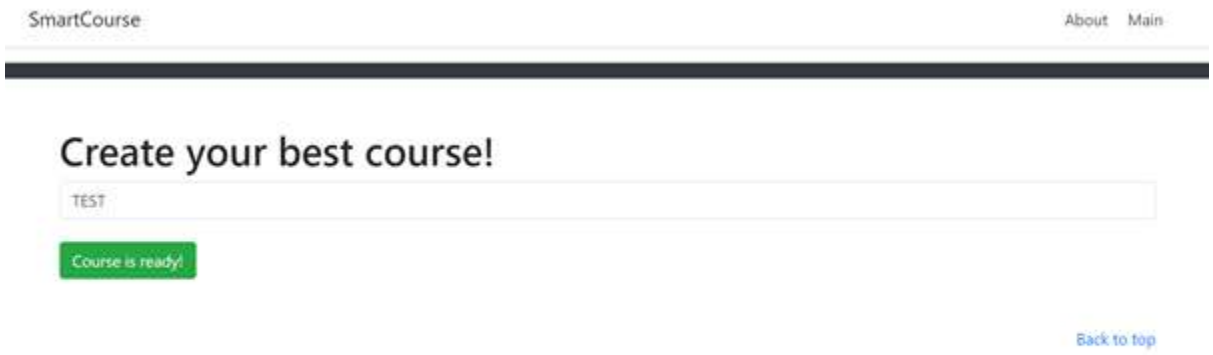

Рисунок 21. Сторінка створення курсу.

Після створення курсу потрібно перейти на головну сторінку курсу (рис. 22), де будуть пости, які може створювати вчитель для поширення інформації. Щоб написати пост треба заповнити відповідні поля і нажати “ADD POST”. Пости будуть бачити усі учасники курсу.

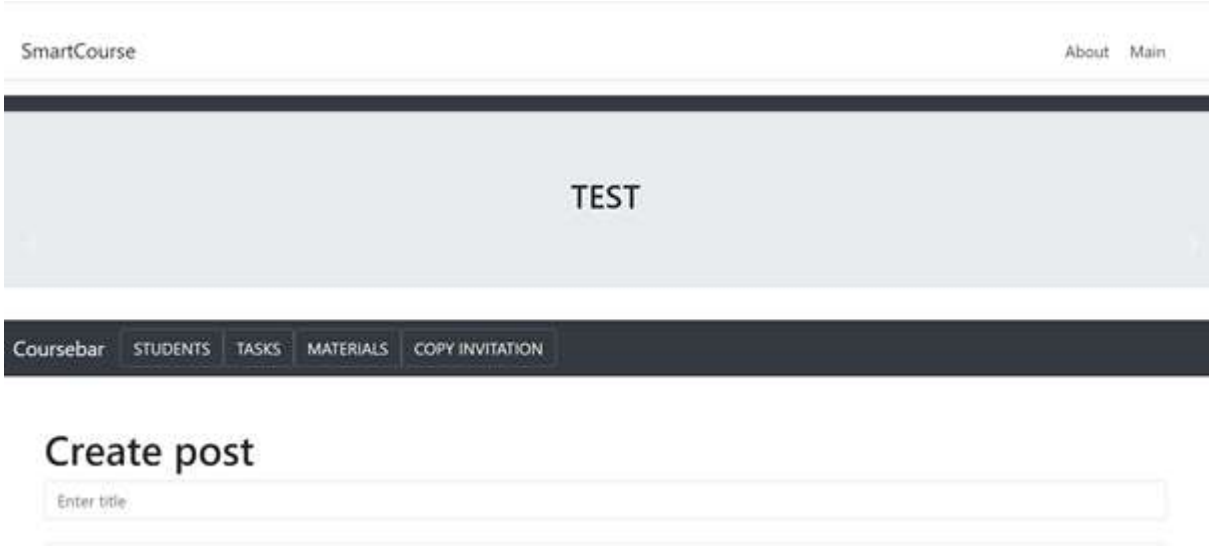

Рисунок 22. Головна сторінка курсу для вчителя

Щоб студенти мали можливість приєднатись до курсу, потрібно скопіювати його посилання. Для генерації посилання потрібно натиснути кнопку “COPY INVITATION”.

Для приєднання до курсу потрібно натиснути “Join course” і вставити посилання в поле, яке буде на сторінці приєднання (рис. 23).

Рисунок 23. Сторінка приєднання до курсу.

Після приєднання студент зможе бачити всі пости вчитель, а також всі завдання від них. Якщо завдання готове студент матиме можливість і завантажити його. Звісно у студента набагато менше функціоналу, ніж у керівника курсу.

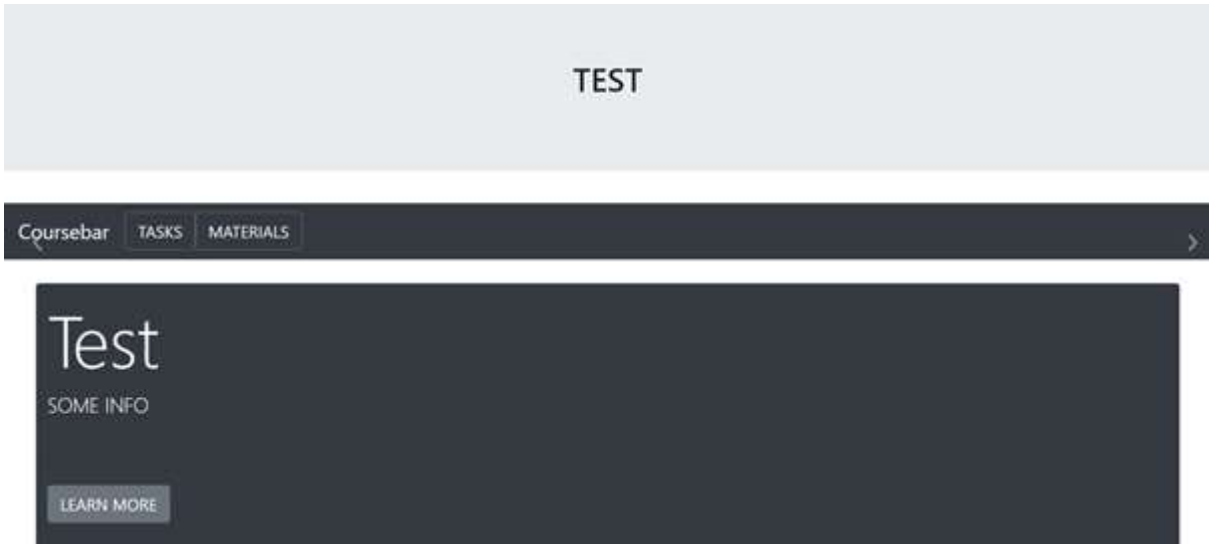

Рисунок 24. Сторінка курсу для учня.

Викладач може створити нове завдання потрібно натиснути “TASKS” (рис. 25). Вчитель має можливість описати завдання, поставити крайній термін здачі роботи, а також прикріпити кілька файлів при потребі.

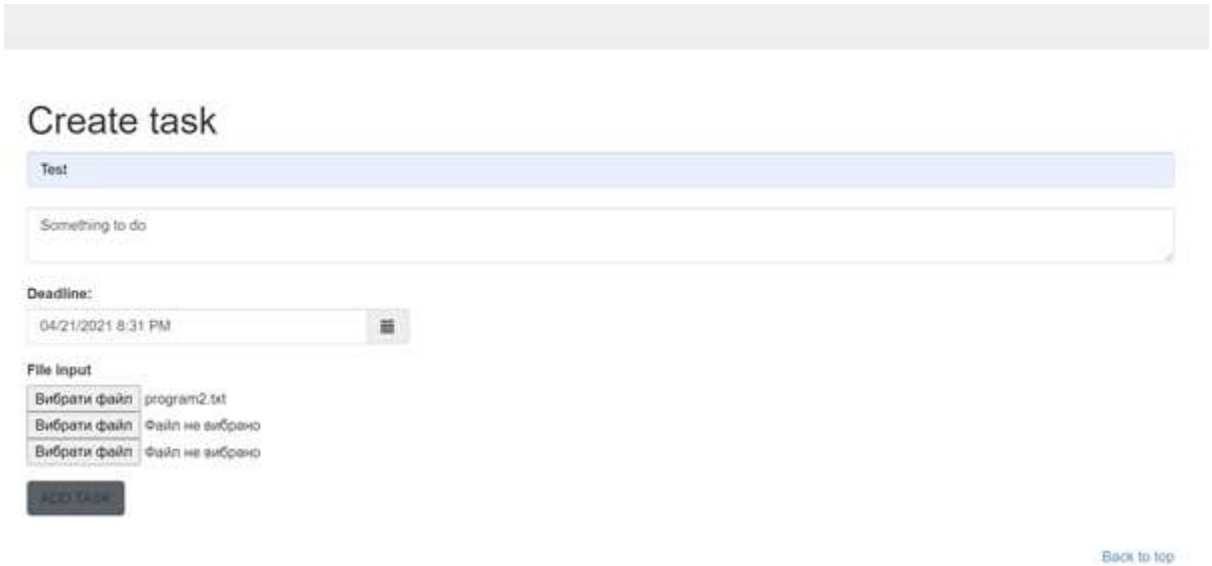

Рисунок 25. Сторінка створення завдання

Після того як викладач створив завдання, він має можливість редагувати і видаляти його. Щоб переглянути відповіді потрібно нажати кнопку “RESPONSES”.

Щоб здати завдання, студенту потрібно перейти на сторінку здачі завдання (рис. 26), де він може заповнити поля для опису відповіді і завантажити свої файли. Також студент має можливість залишити свій коментар під завданням.

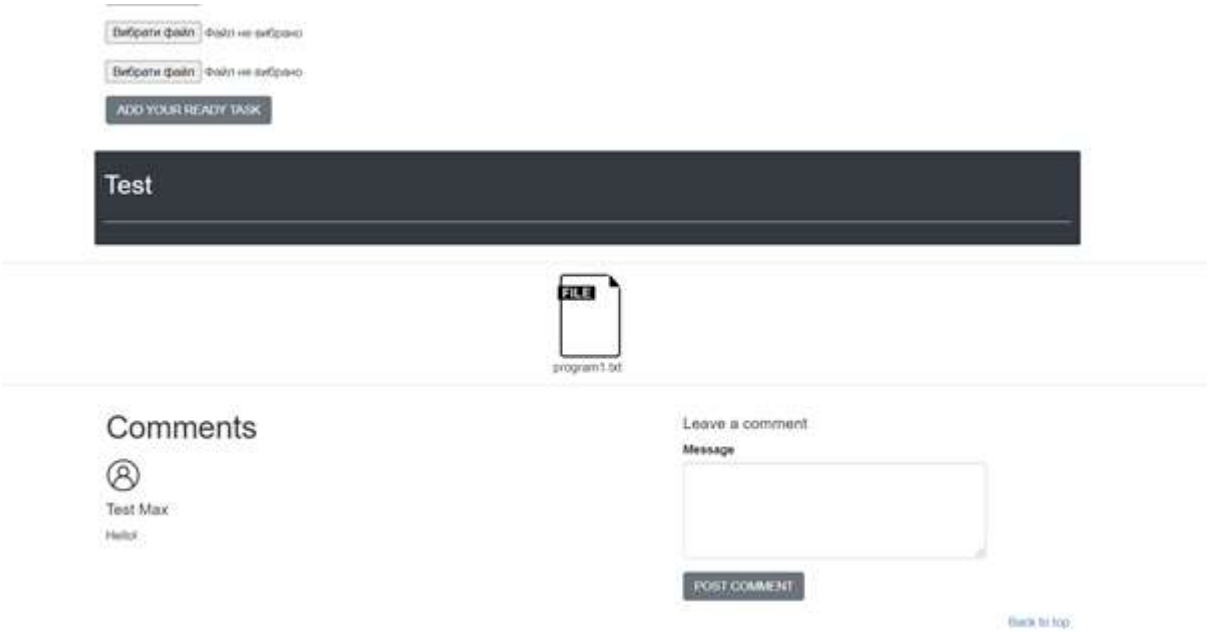

Рисунок 26. Сторінка здачі завдання.

Після того як студент здасть завдання, викладач зможе оцінити його роботу нажавши кнопку “Estimate”. Оцінку можна ставити у певному діапазоні, щоб вона не виходила за межі поставлених рамок.

Викладач має можливість перевірити роботу студента на плагіат використовуючи систему (рис. 27).

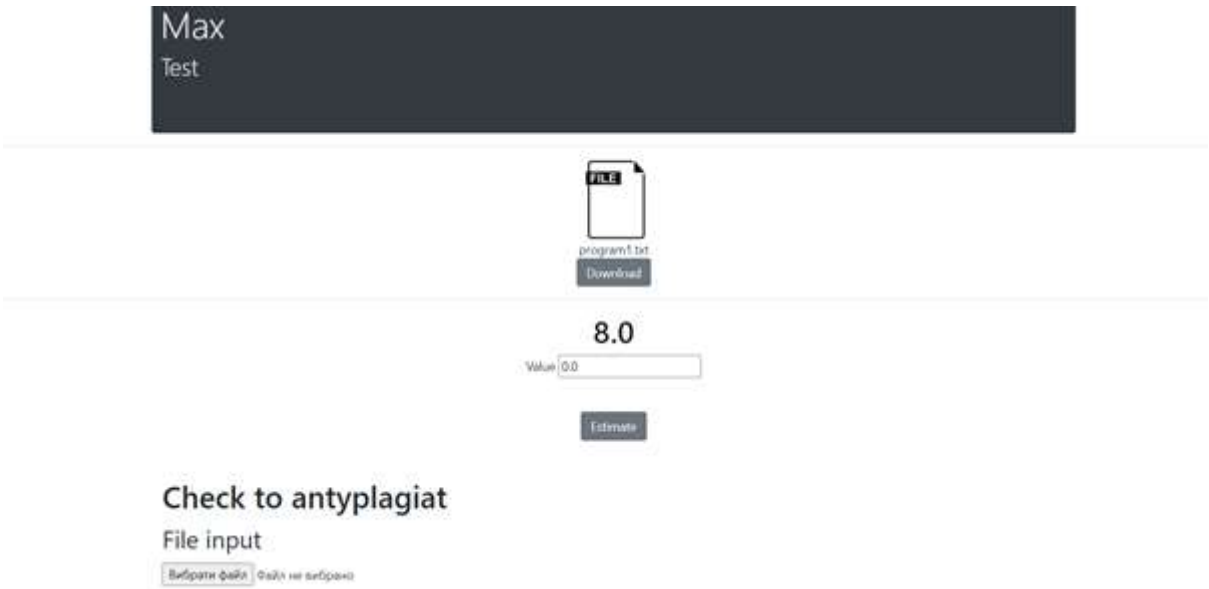

Рисунок 27. Сторінка перевірки на плагіат

Також є можливість перегляду всіх учасників курсу(рис. 28). Для цього потрібно натиснути “Course participants”.

Рисунок 28. Перегляд усіх учасників курсу.

Отже, весь головний функціонал веб-додатку для організації навчальних курсів реалізовано. Система пошуку плагіату успішно інтегрована в проект. Крім того, у проекту є декілька додаткових функцій, що допоможуть студентам краще організувати свій робочий час.

ВИСНОВКИ

У даній роботі розроблено систему перевірки на плагіат програмних текстів і створено веб додаток для організації навчальних курсів, в яких було успішно інтегровано систему.

Спочатку виконано аналіз існуючих систем у Інтернеті і зроблено висновок, що найпопулярніші плагіат системи пристосовані для порівняння простих текстів, а не програмних.

Досліджено різноманітні способи приховування плагіату у програмному коді, і можна сказати, що хоча деякі способи і не можливо повністю виявити, більшість все ж таки можна обійти під час перевірки на унікальність.

Також були проаналізовані різноманітні методи дослідження текстів на унікальність. Результатом аналізу став висновок, що найкращими методами, які підходять для перевірки програмного коду є методи, що використовують нормалізацію коду. Були обрані два методи перевірки на плагіат: метод найбільшого спільного підрядка та метод шинглів. Методи були трохи змінені та модифіковані для кращої роботи саме з програмними текстами. Як наслідок, була реалізована система, що нормалізує програмний код та перевіряє його на унікальність з іншим програмним кодом. Результати програми виявились достатньо точними, щоб зробити висновок про пригодність системи до застосування.

Було розроблено веб-застосунок для організації навчальних курсів. Створено основний інтерфейс і функціонал, що включає створення курсів, написання постів, надсилання і здача завдань, виставлення оцінок. Система по анти плагіату була інтегрована успішно. Також було добавлено додатковий функціонал для кращого планування робочого часу та загального розвитку.

Як загальний висновок можна сказати, що при подальшій роботі і тестуванням на великій кількості людей система може бути інтегрована у навчальний процес факультету комп’ютерних наук та кібернетики.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Classroom [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://classroom.google.com/ .
2. Moodle [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://moodle.org/
3. ILIAS [Електронний ресурс] – Режим доступу до ресурсу: https://www.ilias.de/
4. Unicraft [Електронний ресурс] – Режим доступу до ресурсу: https://www.unicraft.org/ .
5. Advego Plagiatus [Електронний ресурс] – Режим доступу до ресурсу: https://advego.com/plagiatus/ .
6. Etxt Antiplagiat [Електронний ресурс] – Режим доступу до ресурсу: https://www.etxt.ru/antiplagiat/
7. Content-watch [Електронний ресурс] – Режим доступу до ресурсу: https://content-watch.ru/
8. Text.ru [Електронний ресурс] – Режим доступу до ресурсу: https://text.ru/
9. StrikePlagiarism [Електронний ресурс] – Режим доступу до ресурсу: https://strikeplagiarism.com
10. Unicheck [Електронний ресурс] – Режим доступу до ресурсу: https://unicheck.com
11. Library plagiarism policies / Nelson, Robert S. // Random House Compact Unabridged Dictionary – Stepchyshyn: Assoc. of College & Resrch Libraries, 1995. – (ISBN 978-0-8389-8416-1)
12. Plagiarism [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Plagiarism
13. Custom Essay Writers, Freelancers, and Other Paid Third Parties / Newton, Philip M., Lang, Christopher., 2016.

14. Cultural Attitudes Towards Plagiarism / Dr. Lucas. // Lancashire England: Lancaster University – 2003.
15. "We know it when we see it' is not good enough: toward a standard definition of plagiarism that transcends theft, fraud, and copyright" / Fishman, Teddi., 2009. – (4th Asia Pacific Conference on Educational Integrity).
16. A Survey on Software Clone Detection Research / Cordy, James R., Roy, Chanchal Kumar. – School of Computing, Queen's University, Canada -- 2007.
17. Алгоритмы обработки строк / Окулов С. М.., 2013. – (ISBN 978-5-9963016-2-1).
18. Clone Detection Using Abstract Syntax Trees / Yahin, Andrew, Moura, Leonardo, Sant' Anna, Marcelo та ін.]. – Maryland: Proceedings of ICSM'98, 1998. – (IEEE).
19. Ryder, B.G. Constructing the Call Graph of a Program / Ryder, B.G., 1979. – (Software Engineering, IEEE Transactions on, vol. SE-5).
20. Найдовший спільний підрядок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/ Найдовший_спільний_підрядок
21. Поиск нечетных дубликатов [Електронний ресурс] – Режим доступу до ресурсу: https://intellect.icu/poisk-nechetkikh-dublikatov-shingling-algoritm-shinglov-dlya-veb-dokumentov-primer-realizatsii-6914
22. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com
23. C# [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/
24. Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/

ДОДАТОК А

```
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class Normalizer {
    static String separator = "";
    static List<String> punctuator = new ArrayList<>();
    static List<String> keyWord = new ArrayList<>();
    static List<String> numbers = new ArrayList<>();
    static Map<String , String> keyWordMap = new HashMap<>();

    Normalizer(){
        grammarReader("token.txt");
    }

    public String getNormalizedCode(String code){
        code = removeComments(code);
        code = removeText(code);
        findKeyWords(code);
        fillDictionary();
        return normalizeCode(code);
    }
    private String normalizeCode(String code){
        for(String s : keyWord){
            if(s.length()>1){
                code = code.replaceAll("\\b" +s + "\\b",
separator + keyWordMap.get(s));
            }
            else
            {
                if (keyWordMap.containsValue(s))
                {
                    code = code.replaceAll(separator + s,
separator + separator);
                    code = code.replaceAll(s, separator +
keyWordMap.get(s));
```

```
                    code = code.replaceAll(separator +
separator, separator + s);
                }
                else
                {
                    code = code.replace(s, separator +
keyWordMap.get(s));
                }
            }
        }
        for(String s : numbers)
        code = code.replace(s, "#number#");
        code = code.replace("#number#.#number#", "#number#");
        code = code.replace("#number#,#number#", "#number#");
        code = code.replace("#number#", separator +
keyWordMap.get("int"));
        code = code.replace(separator, "");

        //видалення відступів
        code = code.replace(" ", "");
        code = code.replace("\n", "");
        code = code.replace("\r", "");
        return code;
    }
    private static void fillDictionary()//заповнює словник
токенів
    {
        boolean allKeyworsAdded = true;
        do
        {
            for (int i = 1; i < keyWord.size(); ++i)
            {
                if (!keyWordMap.containsKey(keyWord.get(i)))
                {
                    if (keyWordMap.containsKey(keyWord.get(i-
1)))
                    {
```

```
                        keyWordMap.put(keyWord.get(i),
keyWordMap.get(keyWord.get(i-1)));
                    }
                    else
                    {
                        if
(keyWordMap.containsKey(keyWord.get(i+1)))
                        {
                            keyWordMap.put(keyWord.get(i),
keyWordMap.get(keyWord.get(i+1)));
                        }
                        else
                        {
                            allKeyworsAdded = false;
                        }
                    }
                }
            }
            while (!allKeyworsAdded);
    }
    public static void grammarReader(String fileName){
        try(FileReader reader = new FileReader(fileName))
        {
            Scanner sc = new Scanner(reader);
            separator = sc.nextLine();
          while(sc.hasNext()){
              String line = sc.nextLine();
              if(line.equals(separator)){
                  break;
              }
              punctuator.add(line);
          }
          while(sc.hasNext()){
              String line = sc.nextLine();
              if(line.equals(separator)){
                  break;
              }
```

```
            String[] separateLine = line.split(separator);
            String[] keys = separateLine[0].split(" ");
            if (separateLine.length > 1){
               for( String s : keys){
                   keyWordMap.put(s,separateLine[1]);
                }
            }

      }
    }
    catch(IOException ex){

        System.out.println(ex.getMessage());
    }
}

private static void findKeyWords(String code){
    code = code.replace("\n" , " ");
    code = code.replace("\r" , " ");
    for(String s : punctuator){
        code =   code.replace(s , " ");
    }
    String[] splitCode = code.split(" ");
    for(String s : splitCode){
        if(s.length()>0 && !keyWord.contains(s)){
            if (!isNumeric(s))
            {
                keyWord.add(s);
            }
            else{
                if(!numbers.contains(s))
                    numbers.add(s);
            }
        }
    }
}
private static String removeComments(String code){
    while (code.contains("//")){
```

```
            int i = code.indexOf("//");
            String buf1 = code.substring(0, i);
            String buf2 = code.substring(i + 2 );
            i = buf2.indexOf("\n");
            if (i != -1) buf2 = buf2.substring(i);
            code = buf1 + buf2;
        }
        while (code.contains("/*")){
            int i = code.indexOf("/*");
            String buf1 = code.substring(0, i);
            String buf2 = code.substring(i + 2 );
            i = buf2.indexOf("*/");
            if (i != -1) buf2 = buf2.substring(i);
            code = buf1 + buf2;
        }
        return code;

    }
    private static String removeText(String code){
        code = code.replace("\\\"", " ");
        while (code.contains("\"")){
            int i = code.indexOf("\"");
            String buf1 = code.substring(0, i);
            String buf2 = code.substring(i + 1);
            i = buf2.indexOf("\"");
            if (i != -1) buf2 = buf2.substring(i+1);
            code = buf1 + buf2;
        }
        code = code.replace("\\\"", " ");
        while (code.contains("\'")){
            int i = code.indexOf("\'");
            String buf1 = code.substring(0, i);
            String buf2 = code.substring(i + 1);
            i = buf2.indexOf("\'");
            if (i != -1) buf2 = buf2.substring(i);
            code = buf1 + buf2;
        }
        return code;
```

```
    }
    private static boolean isNumeric(String str){
        try {
            Double.parseDouble(str);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

}
```

ДОДАТОК Б

```
import java.util.ArrayList;
import java.util.List;

public class Plagiator {
    int MIN_LCS_LENGTH = 17; //мінімальна довжина однакової
ділянки коду, яка свідчить про плагіат
    public double longestCommonSubstringTest(String test,
String other){
        int originalLength = test.length();//початкова довжина
рядка
        int lcsLength;//довжина найдовшого спільного рядка
(НСР)
        do//повторюємо дії, поки НСР не буде закоротким
        {
            int n = test.length();
            int m = other.length();
            int[][] matr = new int[n][m];
            lcsLength = 0;
            int maxI = 0;
            for (int i = 0; i < n; i++)//заповнює матрицю
пошуку НСР
            {
                for (int j = 0; j < m; j++)
                {
                    if (test.charAt(i) == other.charAt(j))
                    {
                        matr[i][j] = (i == 0 || j == 0) ? 1 :
matr[i - 1][j - 1] + 1;
                        if (matr[i][j] > lcsLength)
                        {
                            lcsLength = matr[i][j];
                            maxI = i;
                        }
                    }
                }
            }
```

```
            if (lcsLength > 0)//якщо НСР знайдено
            {
                String lcs1 = test.substring(maxI + 1 -
lcsLength, maxI + 1);//знаходимо цей НСР
                test = test.replace(lcs1, "");//видаляємо з
рядка, який перевіряємо, всі входження НСР
                String lcs2 =
other.substring(other.indexOf(lcs1),
other.indexOf(lcs1)+lcsLength);
                other = replaceFirst2(other , lcs2 , "");//із
рядка, з яким перевіряємо, видаляємо лише перше входження НСР
            }
        }
        while (lcsLength >= MIN_LCS_LENGTH);//повторюємо дії,
поки НСР не буде закоротким

        //коефіцієнт плагіату знаходиться як відношення
довжини унікальної частини тексту до всієї довжини рядка
        return 1.0 - (double)test.length() / originalLength;
    }
    public double WShinglingTest(String test, String
other)//метод шинглів
    {
        //шукаємо шингли рядка test та відразу рахуємо їхній
хеш
        int testCountShingles = test.length() - MIN_LCS_LENGTH
+ 1;
        List<Integer> testShingles = new ArrayList<Integer>();
        for (int i = 0; i < testCountShingles; ++i)
        {
            testShingles.add(test.substring(i, i+
MIN_LCS_LENGTH).hashCode());
        }

        //шукаємо шингли рядка other та відразу рахуємо їхній
хеш
```

```
        int otherCountShingles = other.length() -
MIN_LCS_LENGTH + 1;
        List<Integer> otherShingles = new
ArrayList<Integer>();
        for (int i = 0; i < otherCountShingles; ++i)
        {
            otherShingles.add(other.substring(i, i+
MIN_LCS_LENGTH).hashCode());
        }
        long count = intersection(testShingles ,
otherShingles).size();
        //коефіцієнт плагіату знаходиться як відношення
кількості однакових хешів до кількості всіх хешів рядка test
        return (double)count/
testShingles.stream().distinct().count();
    }
    public double AveragePlagiatTest(String test, String
other)//середнє арифметичне обох методів
    {
        return (longestCommonSubstringTest(test, other) +
WShinglingTest(test, other)) / 2;
    }
    private static <T> List<T> intersection(List<T> list1,
List<T> list2) {
        List<T> list = new ArrayList<T>();
        for (T t : list1) {
            if(list2.contains(t)) {
                list.add(t);
            }
        }
        return list;
    }
    private static String replaceFirst2(String source, String
target, String replacement) {
        int index = source.indexOf(target);
        if (index == -1) {
            return source;
        }
```

```
            return source.substring(0, index)
                    .concat(replacement)

.concat(source.substring(index+target.length()));
        }
}
```