ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Магістр»

«Система для ідентифікації плагіату в програмному коді»

(тема кваліфікаційної роботи українською мовою)

«A System for Detecting Plagiarism in Source Code»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 122 Комп'ютерні науки .
(код, назва спеціальності)
Освітня програма Комп'ютерні науки .
(назва)
Вітвицький Микола Олександрович
(прізвище, ім'я, по-батькові здобувача)

Керівник к.ф.-м.н., Ткач Т.Б. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент к.т.н., доцент кафедри інженерії ПЗ національного університету "Одеська політехніка", Зіноватна Світлана Леонідівна
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій .
№ ___ від ___ _____ 2024 р.

Завідувачка кафедри

_____ _____.
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _.
протокол № ___ від _____ 2024 р.

Оцінка _____ / _____ / _____.
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ _____.
(підпис) (прізвище,ім'я)

Одеса 2024

АНОТАЦІЯ

В роботі розробляється система для перевірки вихідного коду програм на плагіат. Дослідження спрямоване на створення системи для виявлення плагіату в програмному коді з використанням алгоритму локально-чутливого хешування. Основна мета роботи полягає у розробці системи для ідентифікації плагіату в програмному коді, що допоможе зменшити час, необхідний для перевірки вихідних кодів на плагіат, шляхом зниження асимптотичної складності алгоритму при збереженні його точності, а також у впровадженні нових етапів нормалізації та токенізації тексту. Проєкт спрямований на розробку програмного забезпечення, яке дозволить перевіряти на плагіат як звичайні тексти, так і програмні коди. Для цього було вдосконалено алгоритм локально-чутливого хешування, включивши етапи нормалізації та токенізації текстів, що дозволяє досягти асимптотичної складності O(n), де n є сумарною довжиною оброблюваного тексту. Результатом проєкту є програмна система, яка використовує вдосконалений алгоритм локально-чутливого хешування для перевірки текстів, зокрема програмного коду, на наявність плагіату. Розробка реалізована за допомогою кросплатформеного фреймворку Qt, а також мов програмування C++ та Python3.

Ключові слова: алгоритм, корпус робіт, локально-чутливе хешування, оптимізація, плагіат, система перевірки, Qt.

ABSTRACT

The project focuses on developing a system for detecting plagiarism in program source code. The research aims to create a system for identifying plagiarism in code using the locality-sensitive hashing (LSH) algorithm. The primary goal of the project is to design a system that facilitates the identification of plagiarism in program code, reducing the time required for plagiarism detection by lowering the algorithm's asymptotic complexity while maintaining its accuracy. Additionally, the project introduces new steps for text normalization and tokenization.

The project is directed towards developing software capable of detecting plagiarism in both regular text and source code. To achieve this, the locality-sensitive hashing algorithm was enhanced by incorporating text normalization and tokenization steps, enabling an asymptotic complexity of O(n), where *n* represents the total length of the processed text.

The outcome of the project is a software system that employs the improved locality-sensitive hashing algorithm to detect plagiarism in texts, including program source code. The development was implemented using the cross-platform framework Qt along with the programming languages C++ and Python3.

Keywords: algorithm, corpus of work, locality-sensitive hashing, optimization, plagiarism, detection system, Qt.

ЗМІСТ

ВСТУП

Система для виявлення схожих програмних кодів сьогодні може мати широке застосування. Щодня створюються нові проєкти, розробляються алгоритми для вирішення різних завдань, і реалізуються численні програмні рішення. Оскільки будь-яка людина має доступ до численних джерел інформації, таких як Інтернет або роботи знайомих і друзів, часто зустрічається чимало неоригінальних матеріалів – випадків плагіату. Система, здатна швидко визначити, чи є код унікальним, залишається вкрай необхідною в подібних ситуаціях.

На сьогодні серед інструментів для перевірки програмного коду на плагіат немає таких, які дозволяли б перевіряти код не лише з Інтернету, а й з локальних баз даних, наприклад, з корпусу робіт, що зберігаються в навчальному закладі, або коду, що генерується в реальному часі під час контестів, таких як олімпіади з програмування. Розроблювана система призначена саме для навчальних закладів: вона дозволяє зберігати студентські роботи за всі роки та порівнювати їх між собою. Крім того, система підтримує перевірку коду учасників контестів на плагіат під час змагань. Це значно спрощує роботу викладачів, надаючи їм можливість у будь-який момент дізнатися, чи виявлено системою підозру на неунікальність роботи.

Мета роботи полягає в розробці програмної системи, здатної перевіряти на плагіат тексти, включаючи програмний код.

Для досягнення мети були поставлені такі завдання:

– визначити основні проблеми, проаналізувати існуючі рішення та запропонувати власні підходи;

– розробити алгоритм пошуку схожих програмних кодів із покращеною асимптотичною складністю порівняно з наявними методами;

– спроєктувати систему, визначивши варіанти використання, сценарії та розробивши її архітектуру;

– реалізувати систему на основі спроєктованої архітектури, використовуючи мову програмування C++;

– створити зручний і зрозумілий користувацький інтерфейс;

– провести оцінювання ефективності алгоритму та функціональне тестування системи.

Об’єкт дослідження – процес автоматизованої ідентифікації плагіату в програмному коді, включаючи механізми аналізу схожості, порівняння текстових і структурних компонентів коду.

Предмет дослідження – програмна система, її архітектура, функціональні можливості, а також методи та алгоритми для виявлення плагіату в програмному коді.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Основні поняття

Плагіат програмного коду – це практика привласнення чужих фрагментів програмного коду або повних програмних рішень, без належного посилання на оригінального автора. Це може включати як копіювання коду з відкритих джерел, так і використання чужих рішень, які, можливо, не є доступними для широкої аудиторії (наприклад, робіт колег, однокурсників або знайомих). У контексті навчання або професійної діяльності плагіат коду зазвичай розглядається як неетична поведінка, оскільки порушує права інтелектуальної власності автора оригінального коду.

Плагіат – використання чужої інтелектуальної власності з присвоєнням авторства [1].

Плагіат у програмному коді є дуже поширеною проблемою, зумовленою кількома основними причинами:

– прагнення повторно використовувати код для скорочення часу розробки та тестування програми чи програмного проєкту;

– необхідність підвищення продуктивності програмного забезпечення (цей підхід може бути актуальним, якщо компілятор не має потрібних вбудованих оптимізацій);

– обмежений час на розробку або нестача знань у програміста;

– свідоме використання плагіату або привласнення інтелектуальної власності іншого автора [1].

Свідоме та необґрунтоване використання плагіату під час написання програмного коду може призвести до збільшення обсягу початкового коду проєкту та зниження його структурної зв’язаності. Плагіат у програмному коді є не менш актуальною проблемою і в процесі навчання програмуванню, оскільки він може обмежити глибоке розуміння матеріалу й завадити розвитку навичок, необхідних для створення нових самостійних рішень.

Часте використання плагіату може стати звичною практикою через простоту та швидкість такого підходу [2].

Плагіат також є серйозною проблемою під час змагань зі спортивного програмування, де запозичення коду заборонено правилами. Порушення цього правила призводить до дискваліфікації учасників, якщо було виявлено плагіат [3].

Отже, з огляду на ці причини, можна зробити висновок, що питання виявлення плагіату в програмному коді є актуальним і потребує автоматизованого вирішення [4].

Таким чином, для підтримки високих стандартів академічної та професійної етики важливо створювати та використовувати інструменти для автоматизованого виявлення плагіату у програмному коді, що допомагає забезпечити чесність і унікальність роботи програмістів.

Корпус – сукупність текстів і програмного коду, доступних для аналізу в системі [5]. Дані корпусу мають бути впорядковані за темами та змістом, щоб мінімізувати кількість порівнюваних документів.

Контест – це програмістське змагання, де учасники створюють код для вирішення задач, який тестується в єдиній базі даних. Участь може бути як індивідуальною, так і командною.

Вихідний код – це текст програми, написаний мовою програмування, зрозумілий для людини.

Порівняння текстів і кодів починається після завантаження корпусу та документа, використовуючи спеціалізовані алгоритми для оцінки схожості файлів.

Алгоритм – це визначена обчислювальна процедура, яка приймає один або кілька вхідних параметрів і повертає результат [6]. Рівень схожості між текстами та кодами допомагає системі визначити, чи є робота плагіатом.

Унікальність – протилежне до плагіату поняття. Відсотки плагіату та унікальності мають суму 100%. Наприклад, якщо унікальність документа становить 80%, то відсоток плагіату дорівнює 20% (100% - 80%).

Асимптотична складність – це характеристика алгоритму, яка відображає, як час його виконання залежить від розміру вхідних даних у межі. Ефективніший алгоритм зазвичай кращий для великих обсягів даних, окрім випадків із малими вхідними значеннями [7].

Ціль асимптотичної складності кінцевого алгоритму для розроблюваної системи – $O(n)$, де n – сумарна довжина текстів корпусу.

«О» велике – математичне позначення для порівняння асимптотичної поведінки (асимптотичної складності) функцій [7]. З'ясуємо, що означає «О» велике. Для заданої функції $g(n)$ через $O(g(n))$ позначимо множину функцій (формула 1.1):

$$O(g(n)) = \{ f(n): \text{існують такі додатні сталі } c_1,\ c_2 \text{ і } n_0, \text{ що } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ для всіх } n \ge n_0 \} \quad (1.1)$$

Функція $f(n)$ належить до множини $O(g(n))$, якщо існують додатні константи c_1 і c_2, такі що значення $f(n)$ розташовуються між $c_1 g(n)$ та $c_2 g(n)$ для достатньо великих n. Оскільки $O(g(n))$ є множиною, формально можна було в написати «$f(n) \in O(g(n))$», підкреслюючи, що $f(n)$ є елементом цієї множини $O(g(n))$. Однак частіше використовують запис «$f(n) = O(g(n))$», який позначає те ж саме поняття.

Для користувачів важливо, щоб результати обчислень були отримані максимально швидко. Якщо розглядати часові витрати з точки зору асимптотичної складності й прийняти довжину тексту рівною 10^5 (що приблизно відповідає сумарній довжині вихідних кодів учасників програмного контесту або дипломній роботі на 120 сторінок), то час, необхідний для перевірки такого документа, залежить від обраної асимптотичної складності алгоритму:

- для $O(n^2)$ – 5060-5075 ms;
- для $O(n\log_2 n)$ – 55-65 ms;
- для $O(n)$ – 25-35 ms.

Бачимо значну різницю у швидкості обробки залежно від асимптотичної складності алгоритму. Алгоритми з нижчою складністю, наприклад $O(n)$ або $O(n\log_2 n)$, демонструють значно вищу продуктивність навіть для великих обсягів даних. Це ще раз підкреслює важливість вибору оптимального алгоритму, особливо в задачах з великими масштабами даних, де продуктивність може вплинути на загальний результат [7].

1.2 Методи виявлення плагіату

Методи виявлення плагіату програмного коду включають різні техніки та алгоритми, що дозволяють автоматично або напівавтоматично порівнювати програмні коди для визначення їхньої схожості. Ось детальний опис основних методів.

Текстове порівняння. Розглянемо два основних методів.

Порівняння рядків: цей метод полягає у порівнянні вихідних кодів на рівні рядків. Програми розбиваються на рядки, і кожен рядок порівнюється між собою. Прості методи можуть включати використання алгоритму Левенштейна для підрахунку відстані між рядками. Такий підхід може бути корисним для виявлення точних копій, але має обмежену ефективність, якщо плагіат маскується шляхом незначних змін, таких як додавання зайвих пробілів, коментарів або переформатування.

Метод порівняння за ключовими словами. В цьому методі підхід полягає в пошуку певних ключових слів або шаблонів, що часто використовуються в мовах програмування (наприклад, for, if, while, return). Цей метод зосереджується на аналізі лише важливих частин коду. Проте він є досить простим і не забезпечує високу точність у випадку складних структур або різноманітних стилів кодування.

Методи аналізу структури коду включають наступні.

Абстрактне синтаксичне дерево (AST). У цьому методі вихідний код перетворюється на абстрактне синтаксичне дерево, яке відображає логічну

структуру коду, ігноруючи деталі синтаксису. AST дозволяє фокусуватися на структурних компонентах програми, таких як функції, цикли, і операції, не звертаючи уваги на форматування чи дрібні синтаксичні зміни. AST підходить для виявлення схожих алгоритмів і структур, навіть якщо було змінено деталі реалізації.

Граф потоку керування (CFG). В цьому методі код представляється у вигляді графа, де вершини – це інструкції або блоки коду, а ребра – це можливі переходи між цими блоками. Цей метод виявляє подібності в структурі управління, дозволяючи виявити копіювання на рівні логіки, навіть якщо програмісти модифікували окремі елементи коду. CFG ефективний для виявлення алгоритмічних подібностей, проте є досить ресурсомістким для реалізації та обробки.

Методи локально-чутливого хешування (LSH) включають наступні.

Шинглінг – це коли код розбивається на підпослідовності символів або слів, які називаються шинглами. Потім шингли хешуються, і хеш-коди порівнюються між собою для визначення схожості. Локально-чутливе хешування дозволяє обробляти великі масиви даних та знижує час, необхідний для пошуку схожих фрагментів коду. Цей метод ефективний для виявлення часткових співпадінь, навіть якщо код містить незначні зміни.

MinHash є варіантом локально-чутливого хешування, який порівнює схожість документів на основі ймовірності однакових хеш-кодів шинглів. За допомогою MinHash можна швидко визначити, чи є схожість між фрагментами коду, знижуючи обсяг обчислень порівняно з повним порівнянням. Цей метод підходить для великих корпусів текстів і забезпечує високу продуктивність.

Стилометричний аналіз. Аналіз стилю коду: стилометричний аналіз використовує специфічні характеристики стилю кодування, такі як відступи, розміщення фігурних дужок, назви змінних, розміри та структури функцій. Виходячи з того, що кожен програміст має свій унікальний стиль написання коду, можна виявити плагіат, навіть якщо структура і синтаксис були змінені.

Стилометричний аналіз складний у реалізації та вимагає значної кількості зразків, але може бути ефективним у випадках, коли плагіат ретельно замаскований.

Профілювання програміста: в цьому методі система аналізує попередні роботи програміста для створення його профілю стилю кодування. Потім новий код порівнюється з цим профілем, щоб визначити, чи відповідає він звичайному стилю програміста. Цей підхід корисний у навчальних закладах, де є велика кількість робіт одного автора.

Ще один метод – це аналіз байткоду. Деякі системи можуть компілювати вихідний код у байткод і порівнювати його, оскільки компілятор зазвичай залишає певні шаблони, які можна використовувати для виявлення схожості. Це дозволяє абстрагуватися від конкретних синтаксичних особливостей різних мов програмування, порівнюючи вже зкомпільовані інструкції.

Порівняння на рівні машинного коду – це коли після компіляції код можна порівнювати на рівні машинних інструкцій, хоча це вимагає більшої обчислювальної потужності. Цей підхід підходить для низькорівневого аналізу та може використовуватись у випадках, коли потрібно виявити плагіат навіть у застарілих компільованих проєктах.

1.3 Огляд існуючих програмних рішень

Розглянемо програмні системи для перевірки коду.

SIM (Software Similarity Tester) – це система для визначення подібності текстів, створена голландським науковцем Діком Грюном [8]. Вона існує у двох версіях: для аналізу текстів природних мов і для перевірки вихідних кодів програм.

Версія для природних мов є універсальною та не залежить від конкретної мови, за умови, що текст представлений у кодуванні UTF-8. Версія для аналізу вихідних кодів програм підтримує такі мови

програмування, як C, C++, Java, Pascal, Modula-2, Miranda, Lisp та Assembler 8086.

Результати перевірки подібності відображаються у форматі порівняння (diff) або у вигляді відсотків, причому схожі блоки коду або тексту представлені поруч у двох паралельних колонках. Система складається з дев'яти окремих модулів: вісім з них призначені для аналізу зазначених мов програмування, а один — для звичайних текстів.

Серед основних обмежень SIM — неможливість правильно враховувати зміни в структурі тексту, наприклад, переміщення блоків коду чи фрагментів тексту.

SIM є доступною у вигляді консольного застосунку з відкритим вихідним кодом, що дозволяє розробникам адаптувати її для власних потреб. Завдяки своїй простоті та широким можливостям вона залишається популярним інструментом для виявлення плагіату в програмному коді та текстових документах.

JPlag. Система JPlag була створена у 1996 році для автоматизованого виявлення плагіату, насамперед у студентських роботах. Її тестування проводилося на корпусі студентських програмних кодів. JPlag підтримує такі мови програмування, як C++, Java, C, Python, а також ряд інших мов, які перебувають у стадії бета-тестування [9].

Робота системи складається з двох основних етапів: спочатку вихідний код перетворюється на послідовність лексем, після чого виконується попарне порівняння цих лексем. Для порівняння використовується ефективний алгоритм Greedy String Tiling, який дозволяє знаходити схожі ділянки навіть за наявності незначних змін у коді [10].

JPlag доступна у двох формах: як консольний застосунок та як бібліотека для інтеграції у проекти на мові Java. Програма має відкритий вихідний код, що дозволяє розробникам адаптувати її для своїх потреб. Завдяки простоті використання та надійності, JPlag широко застосовується у навчальних закладах для перевірки академічних робіт.

MOSS (Measure of Software Similarity). Система MOSS (Measure of Software Similarity) була розроблена у 1994 році Алексом Ейкеном у Каліфорнійському університеті в Берклі. Вона підтримує широкий спектр мов програмування, серед яких C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula-2, Ada, Perl, TCL, Matlab та інші [11].

Основою алгоритму MOSS є боротьба з методами приховування плагіату, такими як зміна імен змінних чи порядок функцій. Для цього система використовує метод просіювання (winnowing), що дозволяє генерувати ідентифікаційні мітки (так звані «fingerprints»). Ці мітки є ключовими для порівняння програмного коду.

MOSS функціонує як веб-додаток, де користувачі можуть надсилати свої роботи на перевірку через спеціальний скрипт. Система безкоштовна для некомерційного використання, але для отримання доступу необхідно зв’язатися з правовласниками MOSS. Вона залишається однією з найпопулярніших систем для виявлення плагіату завдяки своїй точності та простоті використання.

SID (Software Integrity System). Система SID (Software Integrity System) була розроблена групами UCSB та UW Bioinformatics. Вона підтримує мови програмування C++ та Java. SID спеціалізується на забезпеченні цілісності програмного забезпечення, допомагаючи виявляти плагіат і слідкувати за схожістю коду.

Попри обмежений набір підтримуваних мов, SID широко використовується в навчальних закладах та дослідницьких установах завдяки своїй простоті та точності в аналізі. Виконання алгоритму відбувається в 2 етапи:

– вихідний код синтаксично аналізується і перетворюється на послідовність токенів;

– для всіх програм попарно обчислюється d(x, y) – міра спільної інформації за допомогою алгоритму TokenCompress. Після цього всі пари

сортуються за незростанням d і виводяться.

Автори стверджують, що система показує дуже високу ефективність знаходження плагіату, але основним їх недоліком є замала кількість мов програмування, що сприяє різкому звуженню можливих сфер використання системи.

Для доступу до системи та відправки текстів на перевірку передбачено веб-інтерфейс.

Розглянуті системи для перевірки вихідних кодів програм на плагіат мають різні підходи до реалізації алгоритмів і пропонують користувачам різні функціональні можливості. Це створює необхідність порівняння цих інструментів за загальними критеріями, щоб оцінити їхні переваги та недоліки.

Основними критеріями для оцінювання є:

– складність алгоритму, яка визначає швидкість і ефективність роботи системи;

– кількість підтримуваних мов програмування, що впливає на універсальність інструменту;

– можливість поширення алгоритму на інші мови програмування за допомогою токенізаторів (лексичних аналізаторів);

– наявність графічного інтерфейсу, що полегшує взаємодію з програмою для кінцевого користувача;

– політика доступу, зокрема, чи є система безкоштовною для використання, або ж потребує ліцензії.

Головні характеристики, які визначають якість систем перевірки текстів і програмного коду на плагіат, включають асимптотичну складність алгоритму перевірки, спектр підтримуваних мов програмування та здатність до адаптації для роботи з новими мовами за допомогою токенізаторів.

Другорядними, але важливими характеристиками є зручність інтерфейсу користувача та доступність системи. Ці аспекти суттєво

впливають на досвід використання та поширення продукту серед розробників і освітніх закладів.

Детальну порівняльну характеристику систем для перевірки вихідних кодів на плагіат представлено у таблиці 1.1, що дозволяє об'єктивно оцінити їх функціонал і знайти оптимальний варіант залежно від вимог користувача.

Таблиця 1.1 – Порівняльна характеристика систем перевірки вихідних кодів програм на плагіат

Характеристика	SIM	JPlag	MOSS	SID
Складність алгоритму	$O(n^2)$	$O(n^3)$	$O(n^2)$	$O(n\log_2 n)$
Кількість мов програмування	8	4 (8 у бета- режимі)	25	2
Поширення токенізатором	Так	Так	Ні	Так
Графічний інтерфейс	Ні	Частково	Частково	Так
Вільний доступ	Так	Так	В некомерційних цілях	Ні

Згідно з результатами порівняння, основним недоліком більшості існуючих систем перевірки вихідних кодів на плагіат є висока асимптотична складність алгоритмів. Складності на рівні $O(n^2)$ або $O(n^3)$ істотно знижують швидкість перевірки при роботі з великими обсягами даних, що обмежує їх ефективність. Крім того, значна частина цих систем має або відсутній, або обмежений графічний інтерфейс, що створює труднощі для пересічного користувача.

Системи можна умовно поділити на дві групи:

1. Програми з відкритим вихідним кодом, які є вільно доступними, але мають низку недоліків, таких як обмежений функціонал чи підтримка мов програмування.

2. Програми з розширеними можливостями, які перевершують аналоги за окремими параметрами, однак мають обмеження у використанні та модифікації через комерційні чи ліцензійні умови.

На сьогодні жодна з наявних систем повністю не відповідає ключовим потребам користувачів, таким як швидка, доступна перевірка вихідних кодів на плагіат зі зручним графічним інтерфейсом. Це залишає значний простір для вдосконалення.

Для задоволення сучасних потреб необхідна програмна система, яка:

- не обмежується певним набором мов програмування;
- має низьку асимптотичну складність, що забезпечує високу швидкість перевірки;
- пропонує інтуїтивно зрозумілий і функціональний графічний інтерфейс.

Наявні рішення не відповідають цим вимогам через значні часові витрати, обмеження у підтримці мов програмування та складність використання через відсутність сучасного графічного інтерфейсу. Це робить їх малопридатними для звичайного користувача, особливо в умовах дедалі більших вимог до зручності та продуктивності програмних продуктів.

1.4 Постановка завдання

Метою роботи є розробка програмної системи для перевірки вихідних кодів програм та текстів на плагіат шляхом застосування модифікованого алгоритму локально-чутливого хешування, що має асимптотичну складність O(n), де n – сумарна довжина оброблюваного тексту. Система має бути реалізована у форматі клієнт-серверної архітектури, призначеної для аналізу та оцінки унікальності вихідних кодів і текстів.

Завдання розробки – створення програмної системи, яка дозволяє завантажувати вихідні коди програм та студентські тексти, аналізувати їх та визначати рівень схожості за допомогою локально-чутливого хешування. Отримані результати передаються викладачу для оцінки. Система має бути сумісною з операційною системою Windows і підтримувати роботу як на настільних, так і на портативних комп’ютерах.

Розроблена програмна система повинна забезпечувати такі функціональні можливості:

1. Реєстрація користувачів: система має дозволяти користувачам зареєструватися в ролі студента або викладача.
2. Завантаження робіт студентами: студенти повинні мати можливість завантажувати свої роботи для перевірки.
3. Автоматична перевірка на плагіат: система повинна автоматично аналізувати завантажені роботи на наявність плагіату.
4. Прийняття рішень про авторство: на основі перевірки система повинна визначати, чи є робота оригінальною; у разі виявлення плагіату студент додається до списку плагіаторів.
5. Доступ викладачів до робіт студентів: викладачі повинні мати змогу переглядати всі студентські роботи, пов’язані з предметами, які вони викладають.
6. Детальний перегляд робіт: викладачі повинні отримувати доступ до детального аналізу робіт окремих студентів.
7. Можливість самостійного аналізу: система має надавати викладачам інструменти для самостійного аналізу студентських робіт.
8. Редагування списку плагіаторів: викладачі повинні мати змогу оновлювати список плагіаторів після додаткового аналізу.
9. Автоматичне створення звітів: система повинна генерувати звіти для викладачів із зазначенням документів, з якими було виявлено схожість.

10. Комунікація зі студентами: викладачам повинна бути доступна функція надсилання студентам електронних листів із результатами перевірки їхніх робіт.

Нефункціональні вимоги до системи:

1. Модуль аналізу: підсистема, що відповідає за перевірку робіт на плагіат, повинна бути реалізована мовою програмування C++.

2. Збереження завантажених даних: усі завантажені вихідні коди та тексти мають зберігатися в корпусі системи.

3. Захист даних під час збою: система повинна гарантувати збереження всіх даних у разі виникнення технічного збою.

2 РОЗРОБКА ВДОСКОНАЛЕНОГО АЛГОРИТМУ ПОШУКУ ПЛАГІАТУ В ПРОГРАМНОМУ КОДІ

Основним завданням є виявлення плагіату у вихідному програмному коді та текстових матеріалах. У цьому розділі представлено розробку вдосконаленого алгоритму, що спрямований на ефективне виконання поставленої задачі. Запропонований алгоритм має покращену асимптотичну складність під час порівняння двох файлів, що забезпечує швидкість і точність перевірки навіть для великих обсягів даних.

Особливістю алгоритму є його здатність працювати як з текстами загального призначення, так і з програмним кодом, враховуючи їх специфіку. Для цього були реалізовані додаткові етапи обробки даних, зокрема:

– нормалізація тексту, що видаляє незначущі символи, вирівнює форматування та усуває варіативність синтаксису;

– токенізація тексту, яка перетворює текст або код у послідовність лексем, що дозволяє ефективно аналізувати структуру й зміст файлів;

– оптимізація порівняння, яка забезпечує асимптотичну складність O(n), де *n* – це загальна довжина тексту або коду.

Розроблений алгоритм знижує обчислювальну складність без втрати точності, що робить його придатним для використання у великих освітніх чи корпоративних середовищах, де необхідно перевіряти значні масиви текстів або вихідних кодів. Крім того, алгоритм підтримує багатопотокову обробку, що забезпечує масштабованість і продуктивність навіть при одночасній перевірці великої кількості файлів.

Таким чином, удосконалений алгоритм не тільки вирішує задачу виявлення плагіату, але й підвищує ефективність роботи системи загалом, роблячи її конкурентоспроможною та зручною для широкого кола користувачів.

Розроблений алгоритм має асимптотичну складність порівняння двох файлів $O(n)$, де n – сумарна довжина текстів корпусу, на відміну від його

аналогів, що мають асимптотичну складність $O(n^3)$, $O(n^2)$, $O(n \log n)$.

2.1 Класифікації плагіату в програмному коді

Існують різні підходи до виявлення плагіату в програмному коді [14].

Розглянемо основні класифікації:

– класифікація Bellon;

– класифікація Roy;

– класифікація по Зельцеру.

Підхід, заснований на класифікації Беллона [15], здійснює виявлення плагіату без урахування форматування програмного коду. Він передбачає поділ дублікатів на декілька типів, залежно від їхньої схожості.

Тип 1. Ідентичні фрагменти програмного коду: цей тип охоплює фрагменти, які є повністю однаковими, тобто їхній текстовий вміст не змінений. Усі символи, пробіли, розділові знаки та інші елементи залишаються без змін. Такий збіг є найпростішим для виявлення, оскільки не потребує глибокого аналізу структури коду чи змісту. Приклад ідентичних фрагментів наведено на лістингу 2.1.

Фрагмент 1:

```
int gcd(int a, int b) { // some comments here if(b==0)
return a; return gcd(b, a%b);
```

Фрагмент 2:

```
int gcd(int a, int b)
{
if (b == 0) { return a; } // some other comments here return
gcd(b, a % b); }
```

Лістинг 2.1 – Ідентичні фрагменти програмного коду

Цей тип плагіату характеризується повним копіюванням коду з

незначними змінами, такими як додавання або видалення пробілів, коментарів, табуляцій, перенесення рядків тощо.

При звичайному послідовному порівнянні такі зміни виявити неможливо.

Тип 2**.** Ідентичні фрагменти програмного коду, але з модифікованими назвами ідентифікаторів, які визначаються користувачем. Це можуть бути назви змінних, констант, класів, структур, методів, функцій тощо. Такий вид плагіату змінює лише семантичні назви, зберігаючи структуру та логіку вихідного коду.

Нижче наведено приклади фрагментів коду, у яких застосовано плагіат 2 типу (лістинг 2.2).

Фрагмент 1:

```
int gcd(int l, int r) { // some comments here if(r==0)
return l; return gcd(r, l%r);
}
```

Фрагмент 2:

```
int gcd(long a, long b)
{
if (b == 0L) { return a; } return gcd(b, a % b); }
```

Лістинг 2.2 – Фрагменти програмного коду з плагіатом 2 типу

Для цього типу плагіату характерне змінення форми коду, назв змінних та значень, що їм присвоюються. Проте синтаксична структура коду залишається незмінною, через що такі фрагменти вважаються сплагійованими.

Тип 3. Цей тип є розширенням типу 2 і включає додавання або видалення окремих частин коду. Це можуть бути нові блоки, рядки коду або видалення певних фрагментів, які не змінюють загальної логіки роботи програми (лістинг 2.3).

Фрагмент 1:

```
int gcd(int l, int r) { // some comments here if(l==0) return r; // add new line
if(r==0) return l; return gcd(r, l%r);
}
```

Фрагмент 2:

```
int gcd(long a, long b)
{
if (b == 0L) { return a; } return gcd(b, a % b); }
```

Лістинг 2.3 – Фрагменти програмного коду з плагіатом 3 типу

На прикладі двох наведених фрагментів можна побачити, що деякі ділянки коду було повністю скопійовано, а до них додано прості обчислення у вигляді окремих рядків, які не змінюють семантику коду. Таким чином, такі зміни відносяться до третього типу плагіату в програмному коді.

Класифікація Roy [16] об'єднує всі три типи плагіату з класифікації Bellon та відносить їх до загальної категорії «синтаксичної близькості». Окрім того, ця класифікація включає ще один, четвертий тип плагіату у вихідному коді.

Типи плагіату за класифікацією Roy:

Тип 1 – 3: Синтаксично близькі фрагменти (відповідають класифікації Bellon).

Тип 4. Семантично близькі фрагменти. До цього типу належать семантично подібні фрагменти коду, які можуть бути реалізовані за допомогою різноманітних синтаксичних конструкцій. У таких випадках, навіть при зміні структури та синтаксису, функціональність коду залишається незмінною, що ускладнює виявлення плагіату.

Таким чином, класифікація Roy розширює існуючий підхід Bellon, додаючи можливість аналізу складніших варіантів плагіату, які стосуються семантичних подібностей у програмному коді (лістинг 2.4).

Фрагмент 1:

```
int gcd(int l, int r) { // some comments here while (b != 0)
{
swap(a, b); b %= a;
}
return a;
}
```

Фрагмент 2:

```
int gcd(long a, long b)
{
if (b == 0L) { return a; } return gcd(b, a % b); }
```

Лістинг 2.4 – Фрагменти програмного коду з плагіатом

Із двох наведених фрагментів можна побачити, що код не використовує спільну логіку обчислень. Обидва фрагменти описують різні алгоритми, але результат їх обчислень буде однаковий - найбільший спільний дільник двох цілих значень.

В даному випадку програміст запозичує ідею розв'язку задачі, але пише абсолютно іншу програму.

Класифікація по Зельцеру [17] виділяє вищевказані типи клонів, а також враховує факт форматування вихідного коду. Детально про кожен із типів:

Тип 1. Повна та точна копія програмного коду з мінімальною кількістю змін.

Співпадає з типом 1 за класифікаціями Bellon та Roy.

Тип 2. Копія коду зі зміною змінних на обчислювальні вирази.

Цей тип плагіату передбачає, що змінні у вихідному коді можуть бути замінені на обчислювальні вирази. Наприклад, значення змінної може бути представлене як результат арифметичної операції або іншого виразу. Незважаючи на ці зміни, логіка і загальна структура коду залишаються ідентичними оригіналу, що дозволяє розглядати такі фрагменти як скопійовані.

Тип 3: копія коду зі зміною окремих виразів.

У цьому типі плагіату певні вирази у коді замінюються іншими. Це може включати підстановку різних функцій, методів або навіть блоків умовного виконання, які виконують ту саму задачу, але з іншим синтаксисом. Хоча код виглядає модифікованим, його семантика залишається незмінною, що вказує на копіювання з мінімальними змінами.

Спільні риси класифікацій.

Усі три згадані типи плагіату (синтаксично і семантично близькі фрагменти) мають багато спільного. Вони базуються на модифікаціях вихідного програмного коду, які, хоча й змінюють його зовнішній вигляд, не зачіпають основну логіку або функціональність. Незалежно від рівня складності змін, алгоритми виявлення плагіату повинні бути здатні знаходити такі схожості.

Об'єднання цих типів у загальну класифікацію дозволяє чітко описати методи копіювання коду і розробити ефективні алгоритми для їх виявлення.

Типи плагіату у програмному коді поділяються залежно від рівня схожості та їхньої наявності в різних класифікаціях:

Тип 1: цей тип є спільним для всіх трьох класифікацій (Bellon, Roy та Зельцера). Він охоплює випадки ідентичності коду без змін.

Тип 2: зустрічається в класифікаціях Bellon і Roy, а також частково відповідає критеріям класифікації за Зельцером. Цей тип враховує схожість із незначними змінами, наприклад, зміною форматування.

Тип 3: також є спільним для Bellon і Roy і частково врахований у класифікації Зельцера. Він включає фрагменти коду з помітнішими змінами, такими як перефразування чи структурні модифікації.

Тип 4: унікальний для класифікації Roy і охоплює складніші форми плагіату, наприклад, коли збережено лише функціональну логіку, але структура або синтаксис значно змінені.

Для більш детального розуміння схожостей та відмінностей між типами плагіату розглянемо таблицю 2.1, у якій наведено порівняння всіх трьох класифікацій. Таблиця дозволяє проаналізувати, як кожен тип плагіату

представлений у різних підходах, визначити спільні риси та унікальні аспекти кожної класифікації.

Таблиця 2.1 – Співставлення класифікацій

Класифікація Bellon	Класифікація Roy	Класифікація по Зельцеру
Тип 1	Тип 1	Точна копія
Тип 2	Тип 2	Тип 2
Тип 3	Тип 3	Тип 3
–	Тип 4	–

З наведеної порівняльної таблиці видно, що четвертий тип плагіату в програмному коді за класифікацією Roy є унікальним і не представлений в інших класифікаціях. Фрагменти вихідного коду, які належать до цього типу, зазвичай не вважаються плагіатом, оскільки вони демонструють нову логіку та підхід до розв’язання задачі.

З семантичної точки зору такі фрагменти суттєво відрізняються, навіть якщо вирішують аналогічні завдання. Саме тому цей тип плагіату, який базується на семантичній близькості, зазвичай не враховується під час перевірок коду на плагіат, оскільки він не порушує унікальності чи авторського підходу до написання коду.

Це підкреслює значення семантичного аналізу у відмінності між плагіатом і творчим переосмисленням задачі, що є важливим аспектом у розробці алгоритмів для виявлення плагіату в програмному коді.

2.2 Розробка алгоритму пошуку плагіату в програмному коді

Після ознайомлення з основними типами плагіату в програмному коді та аналізу алгоритму пошуку схожих фрагментів можна перейти до розробки

алгоритму для виявлення плагіату. Алгоритм складається з кількох етапів: нормалізації коду, токенізації та перевірки схожості документів.

Перший етап: нормалізація вихідного коду.

Нормалізація – це процес очищення та перетворення коду, який дозволяє видалити несуттєві елементи, що не впливають на виконання алгоритму. Основна мета цього етапу – привести код до уніфікованого формату, зручного для подальшої обробки.

Нормалізація вихідного коду включає такі дії:

– приведення символів до нижнього регістру – усі символи стають єдиного формату, що дозволяє уникнути чутливості до регістру;

– заміну символів табуляції на пробіли – це забезпечує єдиний підхід до відступів;

– видалення переносу рядків – код перетворюється в єдиний потік тексту;

– заміна багатьох пробілів одним – видаляються зайві пробіли, що покращує компактність даних.

На виході нормалізації вихідний код представлений у спрощеному, уніфікованому вигляді, готовому до подальших кроків.

Другий етап: токенізація.

Після нормалізації отриманий код необхідно розділити на лексеми, або токени. Токенізація – це процес розбиття тексту на мінімальні одиниці мови програмування, які мають власний сенс. Токени дозволяють спростити структуру коду, підготувавши його для ефективного порівняння.

До основних типів токенів належать:

– назви та ідентифікатори (змінні, функції, класи тощо);

– константи (числові або символьні);

– ключові слова (наприклад, if, while, return);

– знаки операцій (наприклад, +, -, *, /);

– розділювачі (дужки, коми, крапки з комами тощо).

Токенізація базується на визначенні дозволених значень для кожного типу токена. Для реалізації процесу токенізації може використовуватися окремий модуль, який працює автономно. На виході формується масив токенів, який є структурованим представленням вихідного коду.

Третій етап: перевірка схожості.

Заключним етапом є перевірка схожості між документами. Для цього використовується вдосконалений алгоритм локально-чутливого хешування (LSH). Цей алгоритм має асимптотичну складність виконання O(n), де *n* – сумарна довжина текстів у корпусі.

Алгоритм працює на основі хешування токенів і дозволяє знаходити схожі фрагменти, навіть якщо вони мають незначні зміни. Завдяки своїй ефективності, алгоритм здатний перевіряти великі обсяги коду з високою швидкістю.

Деталі реалізації.

Процеси нормалізації та токенізації є критичними для підготовки даних. Усі ці етапи дозволяють забезпечити високу точність і швидкість перевірки програмного коду на плагіат. На виході алгоритм забезпечує ідентифікацію плагіату за рахунок ефективної обробки тексту і виявлення схожих ділянок.

На рисунку 2.1 представлений клас токенів, що застосовується при токенізації. Він ілюструє основні типи токенів та їхні властивості, спрощуючи процес аналізу і порівняння коду. Завдяки цьому підходу забезпечується уніфікований підхід до структурування даних. Крім того, це сприяє підвищенню точності аналізу програмного забезпечення.

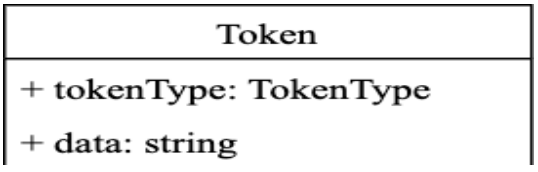

Рисунок 2.1 – Клас Token

Клас Token включає такі поля:

– tokenType: це поле типу TokenType, яке містить перелік патернів для різноманітних лексем. Відповідає за класифікацію лексем відповідно до їхніх характеристик (наприклад, ідентифікатори, ключові слова, оператори тощо).

– data: це поле типу string, яке зберігає поточний знайдений токен (лексему). Це безпосередньо текстове значення, яке було виявлене в процесі лексичного аналізу.

Кожен патерн для поля tokenType можна описати, використовуючи множину можливих значень, наведених у таблиці 2.2. Ця таблиця демонструє типи лексем, які система може розпізнавати (наприклад, числові значення, знаки пунктуації, оператори тощо), і відповідні правила їхнього визначення.

Таблиця 2.2 – Перелік та опис токенів

Назва лексеми	Токен	Можливі значення
Ключові слова	KEYWORD	if, else break, continue, return true, false new, delete private, protected, public switch, case, default const_cast, dynamic_cast, reinterpret_cast, static_cast signed, unsigned try, catch typedef, typename, typeid operator volatile, void, virtual, explicit, friend, inline, static class, enum, struct e xport, externinclude, using, namespace

Продовження таблиці 2.2

Назва лексеми	Токен	Можливі значення
Цикли	CYCLE	for, do, while
Типи змінних	VARIABLE_TYPE	short, int, long bool char, string float, double
Одинарні оператори	OPERATOR_1	+, -, *, /, % . >, <, = !, ?, : &, |, ^, ~
Подвійні оператори	OPERATOR_2	++, -- >=, <= :: &&, || //
Потрійні оператори	OPERATOR_3	>>=, <<=
Константи та числові дані	NUMBER	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Назви ідентифікаторів	IDENTIFIER	ім'я змінної, функції, класу, структури, або іншого об'єкта
Одинарні оператори	OPERATOR_1	+, -, *, /, % . >, <, = !, ?, : &, |, ^, ~

Продовження таблиці 2.2

Подвійні оператори	OPERATOR_2	++, -- >=, <= :: &&, || //
Потрійні оператори	OPERATOR_3	>>=, <<=
Константи та числові дані	NUMBER	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Назви ідентифікаторів	IDENTIFIER	ім'я змінної, функції, класу, структури, або іншого об'єкта

Використовуючи цю таблицю, будь-який фрагмент коду можна перетворити на набір токенів. Після токенізації оброблені дані стають придатними для перевірки на схожість.

Перевірка схожості документів здійснюється за допомогою алгоритму локально-чутливого хешування, який має асимптотичну складність O(n), де n – сумарна довжина оброблюваного тексту. Для додаткової перевірки коректності цього методу розробник може порівняти його результати з алгоритмом Вагнера-Фішера. Хоча останній має значно більшу асимптотичну складність $O(n^2)$, він демонструє високу точність у визначенні схожості текстів.

Алгоритм локально-чутливого хешування (Locality-Sensitive Hashing, LSH) є імовірнісним методом для зниження розмірності багатовимірних даних [19]. Його основна мета полягає в розробці хеш-функцій, які дозволяють схожим об'єктам з високою ймовірністю отримувати схожі хеш-значення.

Цей метод є потужним інструментом для подолання «прокляття розмірності», яке виникає при роботі з багатовимірними даними. Зі збільшенням розмірності даних традиційні методи, такі як пошук за індексами, стають менш ефективними, і перебір стає більш оптимальним. LSH дозволяє створювати структури, що забезпечують швидкий наближений пошук схожих n-вимірних векторів.

У рамках цієї роботи алгоритм LSH використовується для виявлення пар схожих документів в наборі. Алгоритм можна умовно розділити на два етапи:

– етап шинглінгу;

– обчислення коефіцієнта Жаккара між заданим документом і всіма іншими документами.

У поточній реалізації алгоритму Locality-sensitive Hashing асимптотична складність складає $O(n \log n)$, де n – загальна довжина текстів у корпусі. Така складність зумовлена використанням збалансованих бінарних дерев пошуку для зберігання множин, що не є оптимальним для цього алгоритму. Замінивши підхід на більш ефективний метод сортування підрахунком великих чисел, а також застосувавши малі теореми Ферма і бінарне піднесення до степеня, асимптотична складність алгоритму знизилась до $O(n)$, де n – сумарна довжина текстів корпусу.

Хешування – це процес перетворення вхідних даних довільної довжини в рядок фіксованої довжини, що складається з бітів. Операція перетворення визначається хеш-функцією [20]. Метою хешування є швидке порівняння даних, щоб визначити, чи однакові об'єкти між собою.

Оскільки розмір множини вихідних даних обмежений порівняно з множиною вхідних даних, то можуть виникати ситуації, коли різні об'єкти мають однакове хеш-значення. Це явище називається колізією хеш-функції. Хеш-функції, які не мають колізій, називаються ідеальними [20].

Основні вимоги до хеш-функцій:

– висока швидкість обчислення;

– мінімальна ймовірність виникнення колізій на заданих наборах даних.

На практиці важко досягти ідеальних результатів у обох напрямках, тому зазвичай вибирають компромісне рішення, де й швидкість обчислення, ймовірність колізій є достатньо прийнятними для ефективного використання програми.

Поліноміальний хеш – це тип хеш-функції, який має таку особливість: при видаленні чи додаванні символів на початку або кінці рядка, обчислення нового хешу не є трудомістким процесом. Завдяки цій особливості поліноміальне хешування є дуже підходящим для алгоритмів, де необхідно хешувати рядки фіксованої довжини, зокрема для шинглінгу.

Формула поліноміального хешування (формула 2.1) буде наведена далі.

$$H = \left(\sum_{i=0}^{n-1} P^i S_i\right) mod M, \tag{2.1}$$

де H – значення поліноміального хешу,

s – це рядок,

n – значення до якого треба робити хешування,

P – константа, зазвичай обирається просте число, не менше за максимальний з кодів символів рядка,

M – велике просте число.

Позначимо через H_i значення хеша рядка від позиції i до $i+n-1$. Виходячи з формули 3.1 можемо зробити висновок, що якщо знайдено хеш H_i та треба знайти хеш цього ж рядка від позиції $i+1$ до $i+n$ включно, то можна використати рекурентну формулу 2.2.

$$H_{i+1} = \left((H_i + P^n S_{i+n} - S_i) * P^{-1}\right) mod M \quad (2.2)$$

Для ефективних підрахунків будемо використовувати такі алгоритми:

1. Піднесення в додатній степінь *n* числа *a* виконуємо за допомогою бінарного піднесення до степеня, яке працює за принципом:

– Для парного *n* наведено формулу 2.3.

$$a^n = \left(a^{\frac{n}{2}}\right)^2 = a^{\frac{n}{2}} * a^{\frac{n}{2}} \quad (2.3)$$

– Для непарного *n* наведено формулу 2.4.

$$a^n = a^{n-1} * a \quad (2.4)$$

2. Таким чином операція піднесення до степеня числа матиме складність

$O(log_2 n)$ (замість $O(n)$, як у звичайному підході).

3. Для піднесення числа у степінь -1 потрібно використовувати малу теорему Ферма, яка має вигляд:

$$a^{p-1} \equiv 1 \ (mod\ p), \quad (2.5)$$

де p – просте число,

a – ціле число, що не ділиться на p.

Виходячи з малої теореми Ферма (формула 2.5) можемо підрахувати значення a^{-1} за формулою 2.6.

$$a^{p-2} \equiv a^{-1} \ (mod\ p) \quad (2.6)$$

Алгоритм сортування підрахунком великих чисел є варіацією традиційного сортування підрахунком, який використовує інформацію про діапазон значень елементів для сортування послідовності чисел [7]. Асимптотична складність цього алгоритму становить O(n+k), де n – кількість елементів у послідовності, а k – розмір діапазону.

Модифікація сортування підрахунком, яка ефективна для великих діапазонів значень порівняно з розмірами масиву, називається розрядовим сортуванням [7].

Цей метод сортує числа за розрядами в порядку їх зростання, і його складність дорівнює O(rn), де r — кількість розрядів числа. Залежно від системи числення можна зменшити кількість розрядів до 3-4, що дозволяє ефективно застосовувати сортування підрахунком для окремих розрядів. Така модифікація розрядового сортування отримала назву сортуванням підрахунком великих чисел.

Припустимо, ми маємо великі числа, значення яких наближаються до 10^18. Розіб'ємо кожне число на три частини, кожна з яких має довжину 6 цифр. Наприклад, число 123456789098765432 розбивається на три частини: частина 1 – 123456, частина 2 – 789098, частина 3 – 765432. Далі створюємо три порожні масиви розміру 10^6. На першому етапі в масив 1 за індексом x поміщаємо числа, де частина 3 дорівнює x. Потім рухаємося по масиву 1, забираємо з нього числа і додаємо їх у масив 2 за тим самим принципом, після чого повторюємо операцію для масиву 3. У результаті, після сортування, масив 3 містить відсортовану послідовність.

Розглянемо цей алгоритм на прикладі з п’яти чисел, де кожне число не перевищує 10^3. Нехай A = *{980, 733, 225, 125, 984}*. Наше завдання – впорядкувати ці числа у порядку неспадання. Спочатку розбиваємо кожне число на три частини, кожна з яких складається з одного розряду. Після першого етапу сортування (за третьою частиною числа) отримаємо:

A = { 980, 733, 984, 225, 125}

Результат другої ітерації (за другою частиною числа):

A = { 225, 125, 733, 980, 984}

Результат третьої ітерації (за першою частиною числа):

A = { 125, 225, 733, 980, 984}

Таким чином, цей алгоритм дозволяє здійснити сортування чисел, використовуючи O(m) додаткової пам'яті, де m — це найбільше значення однієї з частин числа. Часова складність алгоритму в такому випадку дорівнює O(m * log(m * a)), де a — найбільше число в масиві. Оскільки при правильному виборі константи m значення log(m * a) буде незначним (наприклад, у нашому випадку воно дорівнює 3), асимптотичну складність алгоритму можна оцінити як O(m).

У поточному алгоритмі для сортування застосовується підхід з асимптотичною складністю $O(m * \log_2 m)$, де m — кількість хешів у наборі.

Шинглінг. На етапі шинглінгу кожен документ перетворюється на набір символів (поліноміальних хешів), кожен з яких має фіксовану довжину k (це так званий k-шингл). Основна ідея полягає в тому, щоб представити кожен документ у колекції як набір k-шинглів. Наприклад, якщо взяти документ «Nadal», то для 2-шинглів набір виглядатиме так: {Na, ad, da, al}. Для 3-шинглів це буде {Nad, ada, dal}. Приклади поділу тексту на шингли представлені на рис. 2.2 та 2.3.

Значення k можна обирати будь-яким сталим числом. Однак, якщо k занадто мале, то буде згенеровано багато повторюваних рядків, які зустрічаються в більшості документів. Це може призвести до того, що багато пар документів матимуть велику кількість спільних шинглів, навіть якщо документи насправді не мають спільних фрагментів. Наприклад, при k = 1 більшість текстів міститиме загальновживані символи, і тому множини шинглів цих текстів матимуть багато спільних елементів.

Це може призвести до того, що точність виявлення плагіату знизиться, оскільки система виявить багато схожих пар, які насправді не є плагіатом. Тому важливо правильно обирати значення k, щоб забезпечити баланс між чутливістю до схожості та точністю виявлення плагіату.

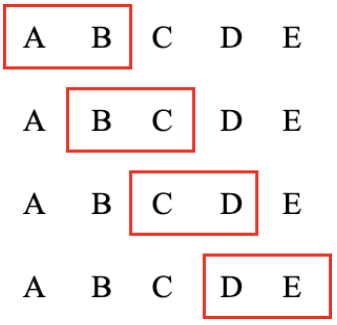

Рисунок 2.2 – Поділ тексту на шингли при $k = 2$

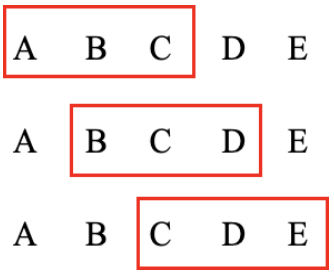

Рисунок 2.3 – Поділ тексту на шингли при $k = 3$

Вибір значення k також залежить від бажаного балансу між точністю та ефективністю обробки даних. Якщо значення k буде занадто великим, то це може призвести до зменшення кількості спільних шинглів між документами, що знизить здатність алгоритму виявляти схожості. Тому важливо враховувати тип і розмір даних при налаштуванні параметра k, щоб забезпечити максимальну ефективність алгоритму перевірки на плагіат. Наприклад, якщо множина даних складається з електронних листів, то оптимальним варіантом для k буде 5. Це випливає з міркувань, які пов'язані з розмірами алфавіту і повідомлення.

Припустимо, що лист може містити тільки латинські букви і символ пробілу (хоча на практиці можуть зустрічатися всі видимі ASCII символи і не тільки). Тоді виходить, що всього може бути $27^5 = 14{,}348{,}907$ різних шинглів. Знаючи, що розмір середньостатистичного листа значно менший ніж 14 мільйонів символів, ми припускаємо, що вибір параметра $k = 5$ є допустимим.

Однак такі міркування є спрощеними і не відображають всієї складності ситуації. Очевидно, що в текстах листів зазвичай використовується більше ніж 27 символів. Крім того, не всі символи мають однакову ймовірність появи. Наприклад, частіше зустрічаються голосні літери, пробіли, розділові знаки та символи нового рядка, тоді як рідкісні літери, як-от «w», мають набагато нижчу частоту використання.

Це означає, що навіть короткі тексти можуть містити багато 5-шинглів із поширеними символами, що збільшує ймовірність того, що різні за змістом документи матимуть спільні шингли. Таким чином, спільність шинглів між документами може бути більш значущою, ніж здається на перший погляд.

Коефіцієнт Жаккара. Після подання кожного документа у вигляді шинглів потрібно обчислити показник для вимірювання схожості між документами – коефіцієнт Жаккара. Для двох документів із відповідними множинами шинглів A і B він визначається за формулою (2.7) і є відношенням розміру перетину множин A∩B до розміру їх об’єднання A∪B.

$$J(A,B) = \frac{A \cap B}{A \cup B}, \tag{2.7}$$

де $J(A, B)$ – коефіцієнт Жаккара,

A – набір шинглів першого документа,

B – набір шинглів другого документа.

Припустимо, що в нас є два документи з текстами «Abcde» та «Abcfj». Множини 2-шинглів для них виглядатимуть так: для першого документа A: {Ab, bc, cd, de}, а для другого документа B: {Ab, bc, cf, fj}. Для визначення перетину цих множин використовуємо діаграму Венна, як показано на рисунку 2.4.

За допомогою цієї діаграми ми бачимо, що перетин множин A та B містить тільки два елементи: {Ab, bc}. Таким чином, коефіцієнт Жаккара між цими двома множинами можна обчислити як відношення кількості спільних

елементів до загальної кількості елементів у об'єднаній множині. У даному випадку коефіцієнт Жаккара буде рівний 2/6, що еквівалентно 1/3.

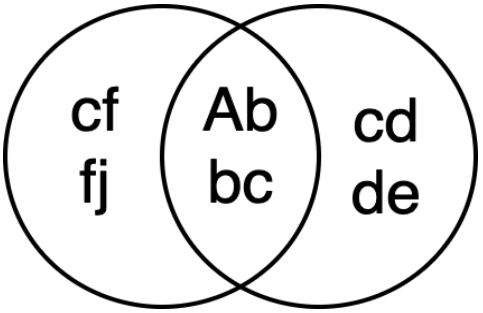

Рисунок 2.4 – Діаграма Венна для розрахунку коефіцієнта Жаккара

Збільшення кількості спільних шинглів між документами призводить до зростання коефіцієнта Жаккара. Це означає, що ймовірність того, що два документи будуть визначені як схожі, також зростає. На основі проведених досліджень було встановлено, що найоптимальнішим значенням коефіцієнта Жаккара є 0.2. Якщо отримане значення більше за 0.2, то документи слід вважати схожими.

Зобразимо описаний вище ефективний алгоритм локально-чутливого хешування за допомогою схеми алгоритму 2.5.

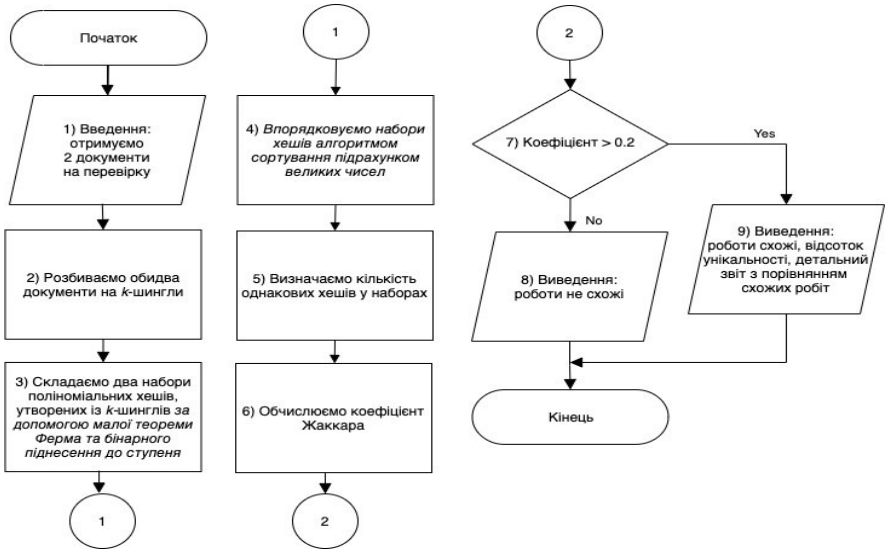

Рисунок 2.5 – Схема алгоритму перевірки схожості двох документів

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Опис функціональних вимог

Функціональні вимоги визначають дії, які система повинна виконувати, описуючи, як вона реагує на різні вхідні дані та поводиться в конкретних ситуаціях. Ці вимоги визначають функціональність програмного забезпечення, яку мають реалізувати розробники, щоб задовольнити потреби користувачів.

Список функціональних вимог до програми був оформлений у вигляді варіантів використання та опису сценаріїв взаємодії з проєктованою системою.

Варіанти використання програмного забезпечення представляють події, що відображають взаємодію користувача із системою, описуючи її поведінку та реакцію на зовнішні запити. Вони дозволяють визначити цілі зацікавлених осіб і виявити вимоги до поведінки системи, які слугують основою для подальшого, деталізованого проєктування.

Діаграму варіантів використання розроблюваної системи для перевірки вихідних програмних кодів та текстів на плагіат наведено на рисунку 3.1. На даному рисунку можна побачити основних користувачів розробленої програмної системи (Студент та Викладач), та можливі сценарії їх поведінки:

– завантаження роботи в систему;
– перевірка роботи на плагіат, що виконується системою автоматично;
– додавання плагіатчиків до списку;
– перегляд робіт студентів; аналіз робіт студентів;
– оновлення кінцевого списку плагіатчиків.

Наведемо детальний опис сценаріїв використання проєктованої системи.

Назва варіанта використання: завантаження роботи в систему.

Передумова варіанта використання: відкрита сторінкадля завантаження

роботи.

Основна послідовність дій сценарію:

1. Студент авторизується в системі.

2. Система відображає форму для завантаження роботи.

3. Студент завантажує свою роботу в систему.

4. Студент відправляє роботу на перевірку.

Гарантія успіху: студент завантажив свою роботу в систему.

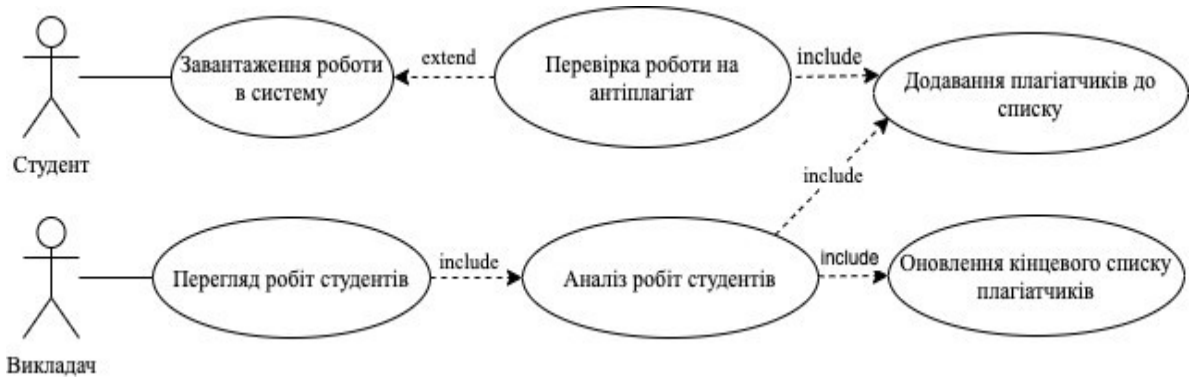

Рисунок 3.1 – Діаграма варіантів використання

Назва варіанта використання: перевірка роботи на анти плагіат.

Передумова варіанта використання: користувач відправив роботу на перевірку.

Основна послідовність дій сценарію:

1. Система перевіряє роботу на плагіат.

2. Система видає результат з перевірки.

Гарантія успіху: система перевірила роботу на антиплагіат і видала результат.

Назва варіанта використання: додавання прізвищдо списку підозрюваних на плагіат.

Передумова варіанта використання: система видала результат перевірки анти плагіат.

Основна послідовність дій сценарію:

1. Якщо робота списана, то прізвище студента буде додано до списку підозрюваних на плагіат.

Гарантія успіху: при виявленні плагіату прізвище користувача додається в список.

Назва варіанта використання: перегляд робіт студентів.

Передумова варіанта використання: викладач запросив перегляд робіт студентів.

Основна послідовність дій сценарію:

1. Викладач авторизувався в системі.

2. Викладач запросив роботи студентів на перевірку.

Гарантія успіху: система видала викладачеві список робіт студентів.

Назва варіанта використання: аналіз робіт студентів.

Передумова варіанта використання: викладач запросив можливість самостійної перевірки робіт студентів на анти плагіат.

Основна послідовність дій сценарію:

1. Викладач переглянув роботи студентів.

2. Викладач переглянув список прізвищ плагіатчиків, згенерований системою автоматично.

3. Викладач переглянув детальний звіт з порівнянням роботи студента з іншими роботами.

4. Викладач знайшов плагіат в роботі студента або спростував результат автоматичної перевірки.

Гарантія успіху: викладач виніс вердикт щодо робіт студентів.

Назва варіанта використання: оновлення кінцевого списку прізвищ плагіатчиків.

Передумова варіанта використання: викладач отримав список передбачуваних плагіатчиків і хоче зробити зміни згідно перевірки, зробленої власноруч.

Основна послідовність дій сценарію:

1. Викладач вносить зміни в кінцевий список плагіатчиків.

2. Гарантія успіху: зміни в списку плагіатчикі в успішно збережені.

3.2 Проєктування архітектури системи

Розроблювана система має клієнт-серверну архітектуру. У цій системі функції сервера виконує персональний комп’ютер, а основна програма з користувацьким інтерфейсом виступає клієнтом.

Для зменшення навантаження на обчислювальні ресурси комп’ютерів користувачів використовується технологія тонкого клієнта. Усі дані в такій архітектурі зберігаються, генеруються та обробляються на сервері. Основна функція клієнта полягає у забезпеченні зручної взаємодії з користувачами.

На рисунку 3.2 наведено основні функції клієнта та сервера, їхню взаємодію, схему обміну повідомленнями між ними, а також розподіл ключових завдань, які виконуються під час роботи системи.

Рисунок 3.2 – Функції та взаємодія клієнта та сервера

Функції клієнта.

Надання користувацького інтерфейсу: система забезпечує зручний інтерфейс для користувача, який дозволяє легко взаємодіяти з системою.

Завантаження роботи студента: користувач має можливість завантажити свою роботу через систему. Після завантаження файл надсилається на сервер для зберігання та перевірки на плагіат.

Взаємодія з базою даних: клієнт отримує необхідні дані з бази даних, що зберігається на сервері.

Реєстрація користувача: користувач може зареєструватися через форму, після чого дані реєстрації надсилаються на сервер.

Надсилання електронного листа студенту: викладач має можливість відправити студенту електронного листа з відгуком на його роботу.

Функції сервера.

Перевірка документів на плагіат: після завантаження роботи студентом, файл надсилається на сервер, де проходить перевірку на плагіат і зберігається.

Генерація звіту про схожість робіт: коли викладач запитує детальний звіт, сервер генерує порівняння робіт студента та відправляє його у форматі .html, який можна переглядати в браузері або зберегти як .pdf.

Збереження робіт студентів: всі завантажені студентами роботи зберігаються на сервері і надаються за запитом.

Збереження бази даних: база даних, що містить необхідну інформацію, зберігається на сервері і надається клієнту за запитом.

Запити клієнта до сервера.

Отримання завантажених робіт студентів.

Отримання детального звіту про плагіат у роботах студентів для викладача.

Отримання інформації з бази даних.

Оновлення бази даних.

Запит результатів перевірки робіт на плагіат.

Сервер відповідає на ці запити, надаючи клієнту необхідні дані.

3.3 Детальне проєктування логічного представлення системи

На діаграмі класів зображено основні класи, які використовуються в модулі перевірки робіт на плагіат. Серед них є класи, що представляють об'єкти, такі як Користувач, Студент, Викладач, Робота, Предмет, а також Інформація про роботи-плагіат. Окрім цих класів, діаграма містить класи, які реалізують функціональність перевірки робіт на плагіат.

З огляду на використання технології тонкого клієнта, для проєктування класів перевірки на плагіат було застосовано структурний шаблон проєктування Замісник (Proxy). Цей шаблон дозволяє замінювати реальні об’єкти спеціальними об’єктами-замінниками, які перехоплюють виклики до оригінального об’єкта. Це дає змогу виконати додаткові дії до або після передачі виклику реальному об'єкту [24].

У відповідності до цього шаблону було створено два класи:

1. PlagiatCheckerServer – відповідає за перевірку на плагіат на стороні сервера.

2. PlagiatCheckerProxy – надає інтерфейс для здійснення запитів з боку клієнта.

Клас PlagiatCheckerProxy виконує роль посередника, передаючи запити клієнтської програми на сервер і забезпечуючи взаємодію через спільний інтерфейс PlagiatChecker, який реалізується обома класами. Такий підхід дозволяє ефективно розподіляти обов’язки між клієнтом і сервером, забезпечуючи зручність і масштабованість системи.

Відповідна діаграма класів представлена на рис. 3.3.

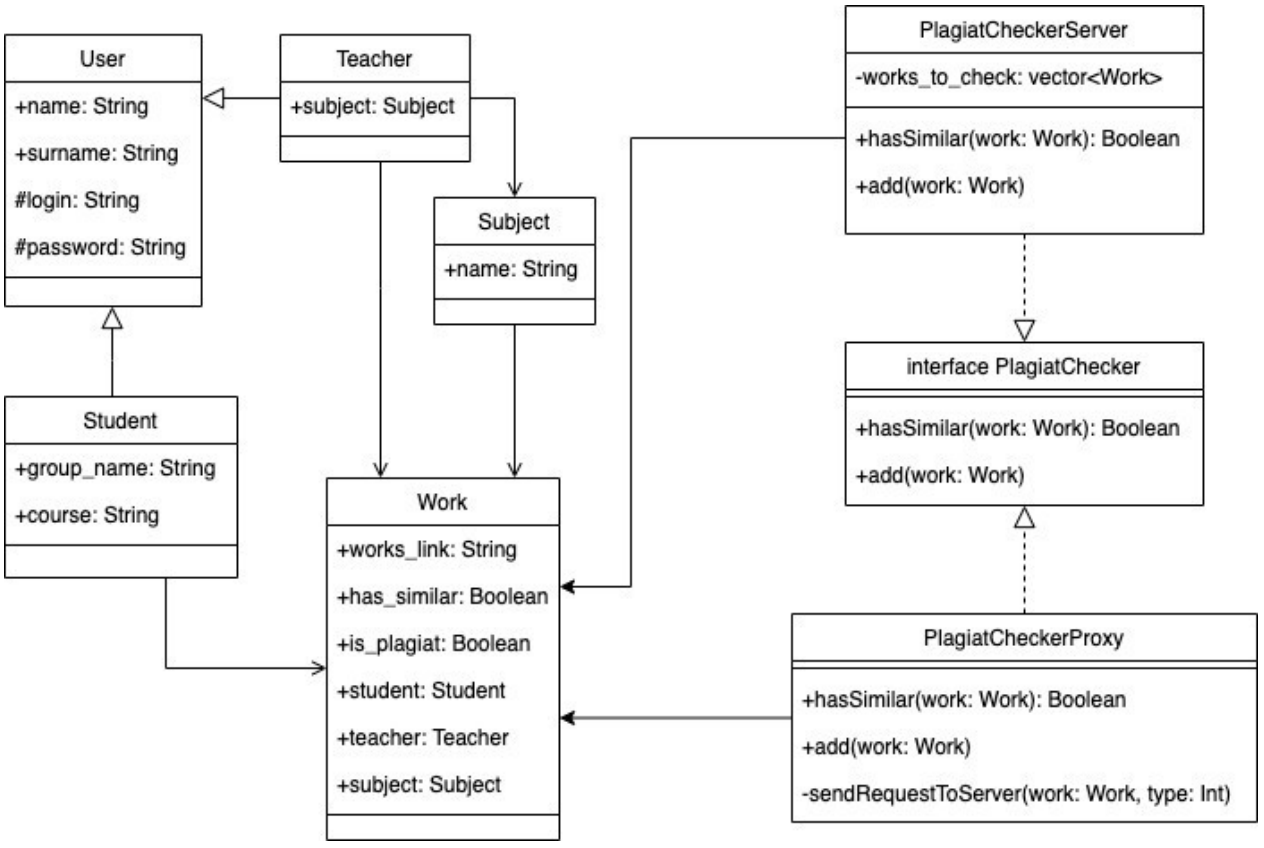

Рисунок 3.3 – Діаграма класів

1. User (Користувач)

   Атрибути:

   - name: ім'я користувача (тип String).
   - surname: прізвище користувача (тип String).
   - login: логін користувача (тип String).
   - password: пароль користувача (тип String).

   Зв'язки:

   - спільний батьківський клас з Teacher та Student.

2. Teacher (Викладач)

   Атрибути:

   - subject: предмет, який викладає викладач (тип Subject).

   Зв'язки:

- наслідується від класу User, що означає, що викладач є підтипом користувача;

- має відношення до класу Subject, що представляє предмет, який викладається.

3. Student (Студент)

   Атрибути:

   - group_name: назва групи студента (тип String);
   - course: курс, на якому навчається студент (тип String).

   Зв'язки:

   - наслідується від класу User, що означає, що студент також є підтипом користувача;
   - пов'язаний з класом Work, оскільки кожен студент має роботи.

4. Subject (Предмет)

   Атрибути:

   - name: назва предмета (тип String).

   Зв'язки:

   - пов'язаний з класами Teacher та Work, тому що викладачі викладають певні предмети, а роботи студентів мають відповідний предмет.

5. Work (Робота)

   Атрибути:

   - works_link: посилання на роботу (тип String);
   - has_similar: булеве значення, що показує, чи є схожість роботи з іншими (тип Boolean);
   - is_plagiat: булеве значення, що вказує, чи є плагіат у роботі (тип Boolean).

   Зв'язки:

   - має відношення до класів Student, Teacher, та Subject;
   - це об'єкт, який перевіряється на плагіат.

6. PlagiatCheckerServer (Сервер перевірки плагіату)

   Атрибути:

- works_to_check: список робіт, які потрібно перевірити (тип vector<Work>).

Методи:

- hasSimilar(work: Work): Boolean: метод для перевірки на наявність схожості у роботі.
- add(work: Work): метод для додавання нової роботи для перевірки.

7. PlagiatChecker (Інтерфейс перевірки плагіату)

   Методи:

   - hasSimilar(work: Work): Boolean: метод для перевірки на наявність схожості.
   - add(work: Work): метод для додавання роботи.

8. PlagiatCheckerProxy (Проксі для перевірки плагіату)

   Атрибути:

   - Проксі-об'єкт, що дозволяє здійснювати доступ до серверу перевірки плагіату.

   Методи:

   - hasSimilar(work: Work): Boolean: перевірка схожості.
   - add(work: Work): додавання роботи.
   - sendRequestToServer(work: Work, type: Int): відправлення запиту на сервер для перевірки роботи.

Клас PlagiatCheckerServer реалізує інтерфейс PlagiatChecker, що дозволяє визначати функціональність для перевірки схожості та додавання робіт на сервер.

Клас PlagiatCheckerProxy працює як проміжний клас, що може здійснювати запити на сервер для перевірки плагіату та додавання робіт.

3.4 Проєктування бази даних

Проаналізувавши потреби системи у збереженні даних на основі детального опису системи, опишемо базу даних, яку повинна

використовувати система за допомогою ER-моделі та схеми реляційної БД.

ER-модель – модель або діаграма даних, головною перевагою якої є надання розробнику можливості описувати високорівневі (концептуальні) схеми предметної області, що у майбутньому дає змогу ефективно спроектувати бази даних.

При проектуванні ER-моделі виділяють ключові для системи сутності. Потім між сутностями можуть бути встановлені зв’язки трьох типів:

– один до одного (взаємодія між сутностями User та Teacher, також User та Student);

– один до багатьох (взаємодія між сутностями Student та Work, Work та PlagiatsInfo, Subject та Work, Teacher та Work);

– багато до багатьох (взаємодія між сутностями Subject та Teacher).

На рисунку 3.4 наведено ER-модель та описано кожну сутність розроблюваної системи.

Наведемо детальний опис кожної суності ER-модель розробленої програмної системи:

– User (користувач). Має унікальний ідентифікаційний номер, ім’я, прізвище, логін, пароль. Один користувач може мати 1 роль - викладач чи студент. Від ролі залежать доступні функції системи.

– Student (студент). Має унікальний ідентифікаційний номер, курс та групу. Один студент може завантажити багато робіт у систему.

– Teacher (викладач). Має унікальний ідентифікаційний номер. Багато викладачів можуть викладати багато предметів, один викладач може перевіряти багато робіт студентів та оновлювати в них статус, чи є робота плагіатом.

– Work (робота). Характеризується унікальним ідентифікаційний номером, текстом роботи, висновком системи та викладача стосовно того, чи є робота унікальною.

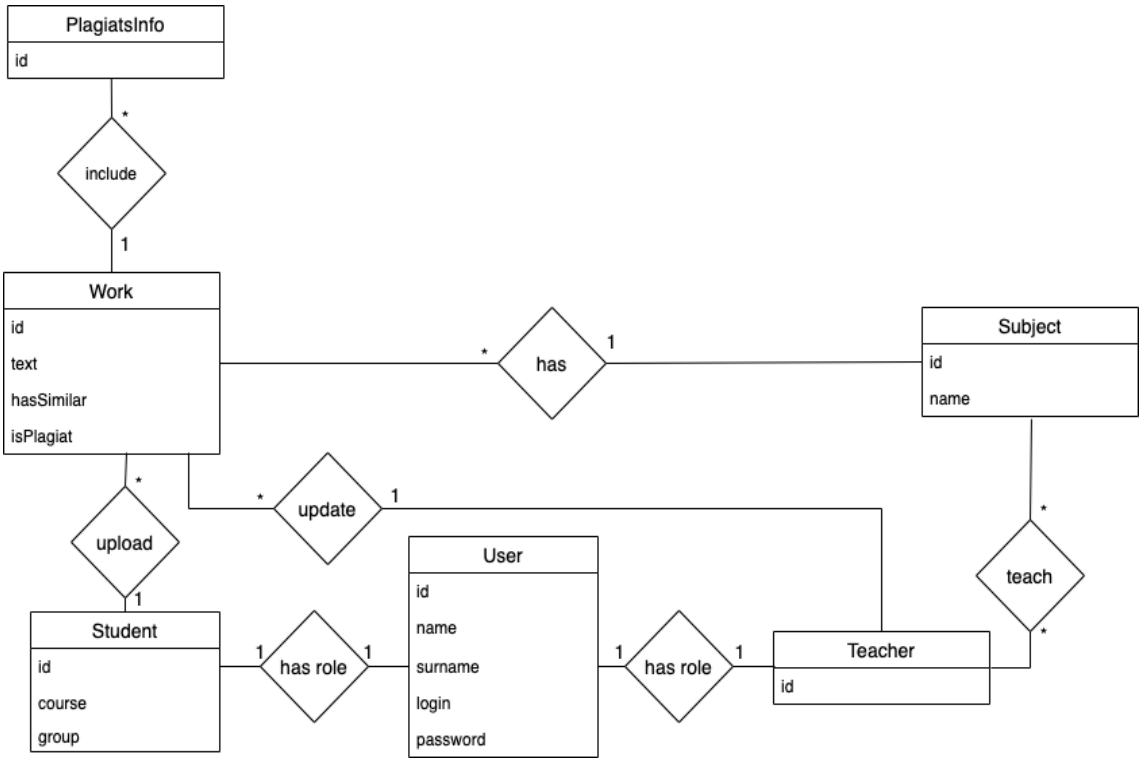

Рисунок 3.4 – ER-модель розроблюваної системи

– Subject (предмет). Характеризується унікальним ідентифікаційний номером та назвою. З одного предмета може бути завантажено багато робіт.

– PlagiatsInfo (інформація про роботи, які визнані плагіатом). Характеризується унікальним ідентифікаційний номером та парами посилань на схожі роботи.

Для логічного проектування було обрано модуль перевірки роботи на плагіат.

Для більш детального опису модуля перевірки на плагіат будемо використовувати наступні об’єкти: User (користувач), Student (студент), Teacher (викладач), Subject (предмет), TeacherSubject(залежність, який вчитель веде той чи інший предмет), Work (робота), PlagiatsInfo (список робіт плагіатів).

Побудуємо схему реляційної БД для вказаних об’єктів з предметної області (рис. 3.5). Завдяки цій моделі можна впорядкувати дані в одну або декілька таблиць, кожна з яких характеризується унікальним ключем. Всі таблиці та відношення в реляційній БД відображають одну сутність. Рядки призначені для відображення екземплярів даного типу сутності, стовпці – для опису значень, які відносяться до цього екземпляра.

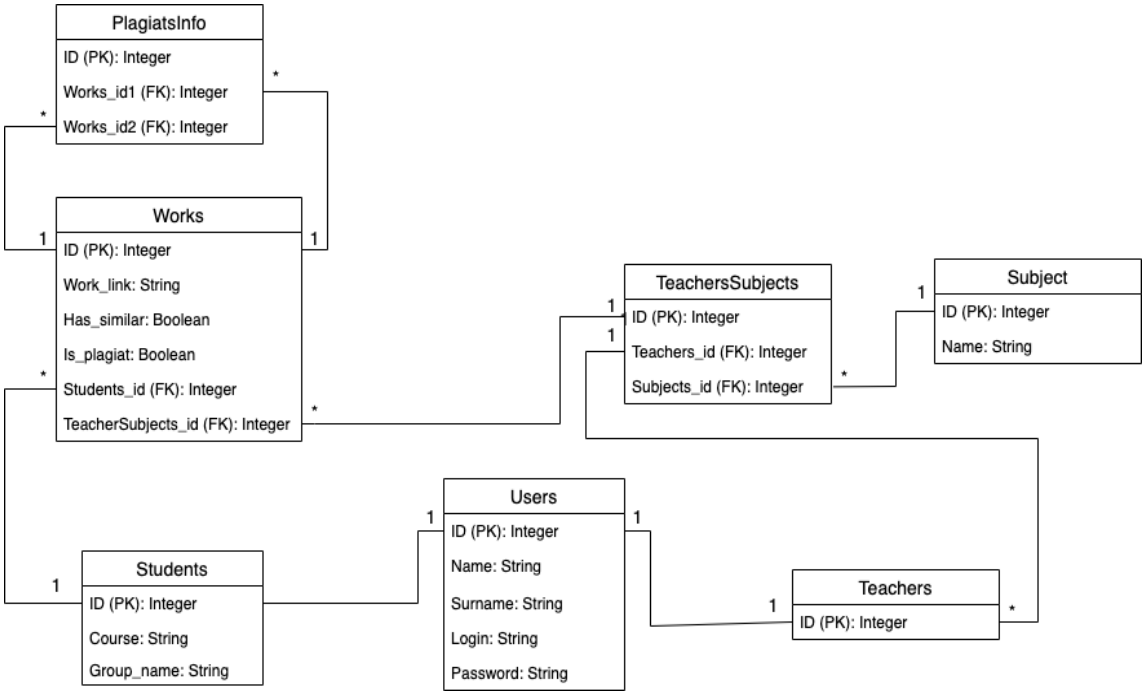

Рисунок 3.5 – Схема реляційної БД

На основі зображення з діаграмою бази даних (ER-діаграмою), розпишемо таблиці з її атрибутами та зв’язками:

1. Користувачі – таблиця інформації про користувачів.

Атрибути:

ID(PK): унікальний ідентифікатор користувача.

Name: ім'я користувача.

Surname: прізвище користувача.

Login: логін для входу в систему.

Password: пароль для входу в систему.

2. Студенти – таблиця з даними про студентів.

Атрибути:

ID(ПК): унікальний ідентифікатор студента.

Course: курс, на якому навчається студент.

Group_name: назва академічної групи студента.

3. Вчителі – таблиця з даними про викладачів.

Атрибути:

ID(ПК): унікальний ідентифікатор викладача.

4. Дисципліна – таблиця інформації про дисципліну.

Атрибути:

ID(ПК): унікальний ідентифікатор дисципліни.

Name: назва дисципліни.

5. ВчителіПредметники – зв'язувальна таблиця між викладачами та дисциплінами.

Атрибути:

ID(PK): унікальний ідентифікатор зв'язку.

6. Праці – таблиця інформації про студентські роботи.

Атрибути:

ID(ПК): унікальний ідентифікатор роботи.

Work_link: посилання на студентську роботу.

Has_similar: логічне значення, яке вказує, чи є такі роботи.

Is_plagiat: логічне значення, яке вказує, чи є робота плагіатом.

7. PlagiatsInfo – таблиця для відстеження інформації про плагіат.

Атрибути:

ID(PK): унікальний запис ідентифікатора.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Опис програмних технологій

Для зберігання інформації у розробленій програмі було обрано реляційну систему керування базами даних SQLite [25].

Основними перевагами обраної бази даних є:

– зручне зберігання. Вся база даних зберігається у єдиному файлі, який може бути відкритий на будь-якій платформі;

– можливість зберігати великі об’єми даних. Бази даних розміром у терабайт та дані розміром у гігабайт;

– швидкість найпоширеніших операцій;

– простота. Легкість у використанні;

– автономність. Обрана база даних не має зовнішніх залежностей;

– кросплатформність. Можливість легкого перенесення на такі системи, як Linux, Mac OS, Windows.

Таким чином, для зберігання основних даних, таких як інформація про користувачів, роботи студентів та результати перевірок на плагіат, було обрано реляційну систему керування базами даних SQLite. Усі класи програми реалізовані за допомогою мови програмування C++, яка є компільованою, об’єктно-орієнтованою мовою загального призначення зі статичною типізацією [26].

Оскільки платформа повинна бути зручною у використанні та задовольняти потреби користувачів незалежно від типу пристрою, з якого здійснюється доступ, програму було розроблено з адаптивним дизайном. Для створення інтерфейсу було використано середовище розробки Qt Creator, яке дозволяє створювати багатоплатформні рішення. Qt Creator – це інтегроване середовище розробки, що використовує стандартні бібліотеки Qt для створення сучасних, інтуїтивно зрозумілих інтерфейсів [27].

Платформа підтримує розробку такими мовами програмування, як: C,

C++, QML та блоки CSS для написання інтерфейсу. Основними перевагами середовища є:

1. Кросплатформність. Можливість легкого перенесення розробленого додатку на такі системи, як Linux, Mac OS, Windows.
2. Вбудований дизайнер інтерфейсу. Можна скласти та налаштувати віджети або діалоги та протестувати їх, використовуючи різні стилі та роздільну здатність безпосередньо в редакторі.
3. Підтримка CSS при написанні інтерфейсу. Дає змогу розробити інтерфейс легким та зрозумілим для користувача.
4. Вбудований редактор для HTML. Дає змогу редагувати зовнішний вигляд інтерфейсу та відкривати тексти за необхідністю в браузері.

Для генерації звіту з порівнянням всіх схожих робіт для викладача був написаний скрипт на мові Python 3. Python 3 [28] – це інтерпретована мова програмування високого рівня. Ця мова об’єктно-орієнтована та має строгу динамічну типізацію. Написаний скрипт самостійно генерує файл з розширенням .html, який автоматично відкриється для викладача у браузері. Програмний код цього скрипта наведений у Додатку А під назвою diff.py. Також викладач має можливість зберегти звіт на локальний комп’ютер у форматі .pdf засобами свого браузеру.

Локальний сервер [29] був створений за допомогою класів з бібліотеки Qt – QTcpServer та QTcpSocket, передача даних виконується на основі протоколу TCP.

TCP [30] – це протокол управління передачею даних. Протокол забезпечує надійну, упорядковану та перевірену помилками передачу потоків даних між програмами, що працюють на хостах та спілкуються через IP-мережу.

QTcpSocket – це підклас QAbstractSocket з бібліотеки Qt. Він надає можливість встановлювати TCP-з'єднання між сервером та клієнтом, після чого передавати потоки даних.

Використаний клас QTcpServer забезпечує розгортання сервера на

локальному комп'ютері на основі TCP. Він надає можливість приймати TCP-з'єднання, надіслані зі сторони клієнтів. Сервер реалізований окремою системою, яка запускається на локальному комп’ютері. Основна система не може бути запущеною без з’єднання із сервером.

4.2 Інтерфейс користувача

На рис. 4.1 представлено форму входу в систему, яка з'являється перед користувачем одразу після запуску програми. Інтерфейс форми вирізняється простотою та зрозумілістю, що забезпечує швидкий доступ до системи.

Користувач повинен ввести свої облікові дані, зокрема ім'я користувача та пароль, щоб отримати доступ до функціоналу системи. Для зручності передбачено можливість відновлення пароля у випадку його втрати, а також кнопка для переходу до реєстрації, якщо користувач ще не має облікового запису. Такий підхід забезпечує не лише безпеку, але й комфортність використання системи.

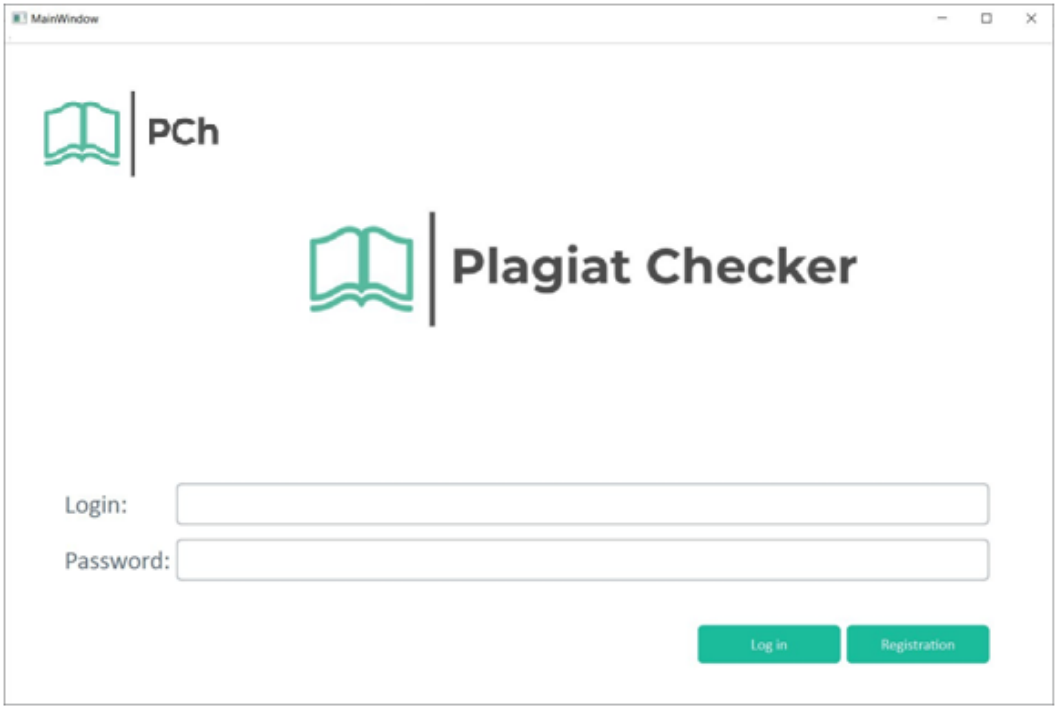

Рисунок 4.1 – Форма входу в систему

Форма входу до розробленої системи включає такі ключові компоненти:

Поле Login: поле для введення логіна користувача. Це поле є обов’язковим для заповнення, оскільки без нього ідентифікація користувача неможлива.

Поле Password: поле для введення паролю користувача. Також є обов’язковим для заповнення, щоб забезпечити безпеку доступу до системи.

Кнопка Log in: кнопка, що ініціює процес входу до системи. Після її натискання відбувається перевірка введених облікових даних.

Кнопка Registration: ця кнопка дозволяє перейти до форми реєстрації для створення нового облікового запису.

Щоб отримати доступ до системи, користувач має пройти обов'язкову процедуру реєстрації, після якої його облікові дані зберігаються у базі даних на сервері.

На рис. 4.2 представлено форму реєстрації користувача, усі поля якої є обов’язковими для заповнення. Реєстрація передбачає введення особистих даних, які будуть використовуватися для ідентифікації та забезпечення доступу до системи. Такий підхід сприяє організації надійного контролю доступу та захисту інформації.

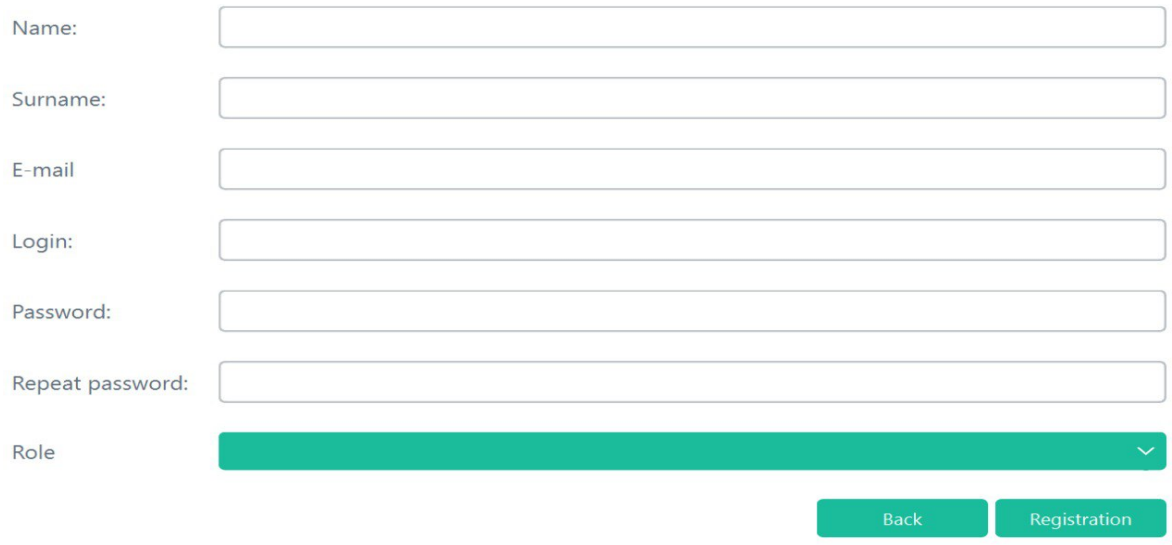

Рисунок 4.2 – Форма реєстрації

Форма реєстрації користувача містить такі основні компоненти:

- Поле Name: поле для введення імені користувача.
- Поле Surname: поле для введення прізвища користувача.
- Поле E-mail: поле для введення електронної адреси користувача. Використовується для комунікації та можливого відновлення доступу до системи.
- Поле Login: поле для створення унікального логіна користувача.
- Поле Password: поле для введення паролю користувача, який забезпечує доступ до системи.
- Поле Repeat Password: поле для повторного введення паролю. Введені значення у полях Password та Repeat Password мають збігатися, щоб уникнути помилок при створенні облікового запису.
- Поле Role: поле для вибору ролі користувача. Доступні варіанти: Студент або Викладач. Обраний варіант впливає на подальше заповнення додаткових форм:

Якщо обрана роль Студент, користувач заповнює форму, представлену на рис. 4.3.

Якщо обрана роль Викладач, користувач заповнює форму, представлену на рис. 4.4.

Кнопка Back: Кнопка для повернення на сторінку входу в систему (рис. 4.1).

Кнопка Registration: Кнопка для завершення процесу реєстрації. Після її натискання система перевіряє правильність заповнення всіх полів і зберігає дані у базу.

Форма для заповнення при реєстрації Студента, наведена на рис. 4.3, забезпечує введення додаткових даних, таких як група, номер студентського квитка та інші параметри, необхідні для ідентифікації студента в системі.

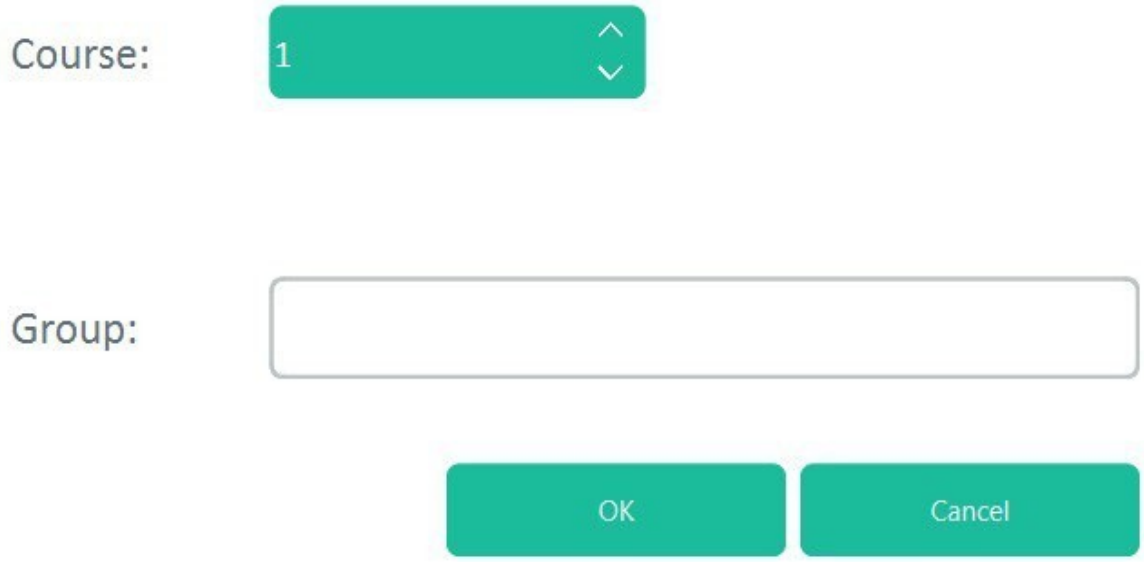

Рисунок 4.3 – Форма для заповнення студентом при реєстрації

Форма реєстрації для студентів включає такі компоненти:

– Поле Course: поле для введення курсу студента. Це поле визначає поточний рівень навчання користувача.

– Поле Group: поле для введення групи студента, що дозволяє ідентифікувати його в межах навчального процесу.

– Кнопка OK: кнопка для підтвердження та збереження введених даних. Після натискання система перевіряє правильність заповнення форми.

– Кнопка Cancel: кнопка для скасування введення даних і повернення до попереднього етапу.

У разі коректного заповнення всіх полів, дані студента зберігаються у базі даних, і користувач може перейти до основного інтерфейсу системи.

Форма для реєстрації викладача, представлена на рис. 4.4, містить специфічні для цієї ролі поля, які дозволяють враховувати особливості викладацької діяльності.

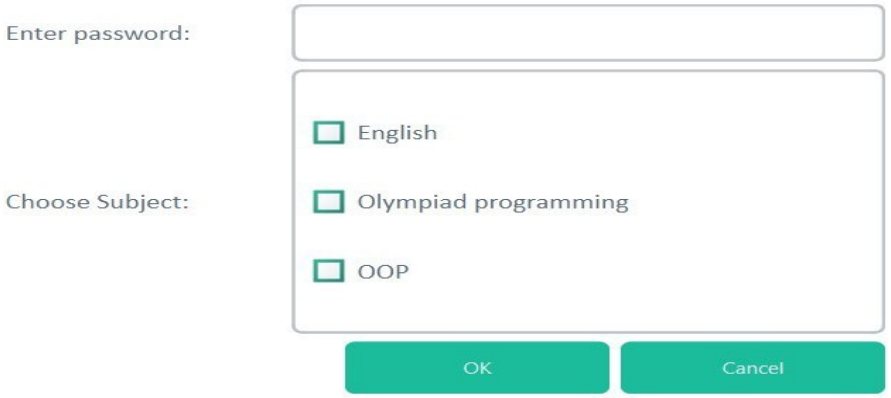

Рисунок 4.4 – Форма для заповнення викладачем при реєстрації

Форма для реєстрації викладача включає такі компоненти:

- Поле Enter Password: поле для введення пароля викладача. Пароль має відповідати встановленим критеріям безпеки.
- Поле Choose Subject: поле для вибору одного або кількох предметів, які викладач буде вести. Це поле дає змогу створити зв'язок між викладачем і навчальними дисциплінами.
- Кнопка ОК: кнопка для підтвердження введених даних. Натискання на кнопку ініціює перевірку коректності заповнення форми та збереження даних у базі.
- Кнопка Cancel: кнопка для скасування введених даних та повернення до попередньої сторінки.

Після успішної реєстрації викладач отримує доступ до функціоналу системи, який відповідає його ролі.

Форма для завантаження роботи студентом, представлена на рис. 4.5, розроблена для забезпечення процесу подання навчальних матеріалів студентами.

Форма для перегляду викладачем усіх робіт, надісланих студентами, включає наступні функціональні можливості:

– Вибір предмета: якщо викладач веде кілька дисциплін, він може обрати потрібний предмет зі списку або за допомогою випадаючого меню. Це спрощує перегляд робіт для кожної дисципліни окремо.

– Список робіт: таблиця або список, у якому відображаються роботи, надіслані студентами. У ньому можуть бути такі стовпці, як ім'я студента, назва роботи, дата подання та статус перевірки.

– Функції сортування та фільтрації: для зручності користування форма може мати інструменти для сортування робіт за різними критеріями (дата, статус, студент) або фільтрації, наприклад, за групою студентів.

– Кнопка перегляду роботи: кожна робота має бути доступною для перегляду в один клік, наприклад, натискаючи на відповідну кнопку або посилання в списку.

– Кнопка оцінки роботи: викладач може швидко перейти до виставлення оцінки або додавання коментарів для кожної конкретної роботи.

Ця форма забезпечує викладачу зручний інтерфейс для організації перевірки та оцінювання студентських робіт.

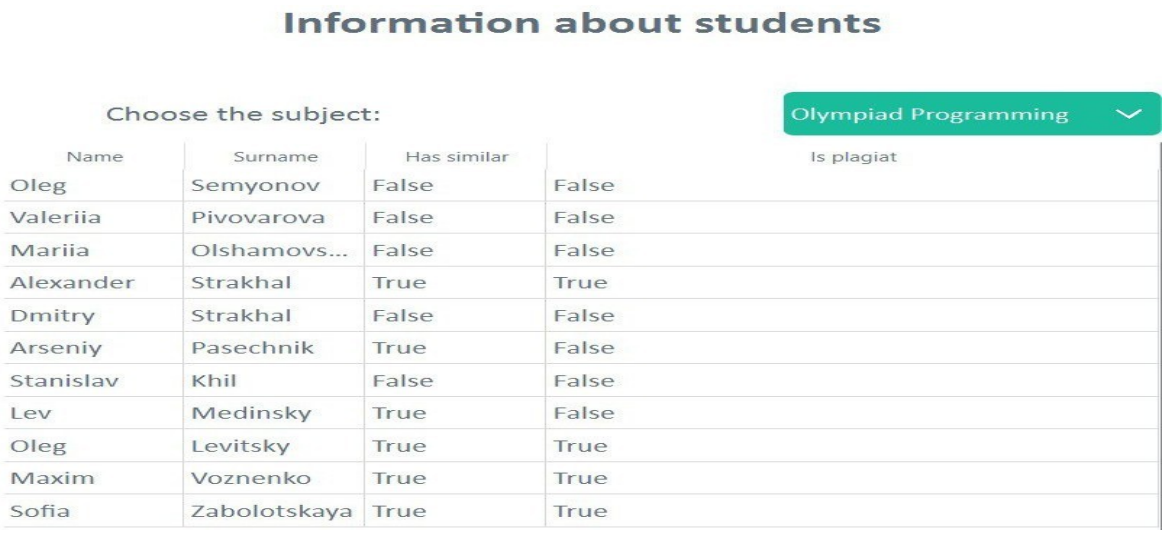

Рисунок 4.5 – Форма для перегляду викладачем всіх робіт, відправлених студентами

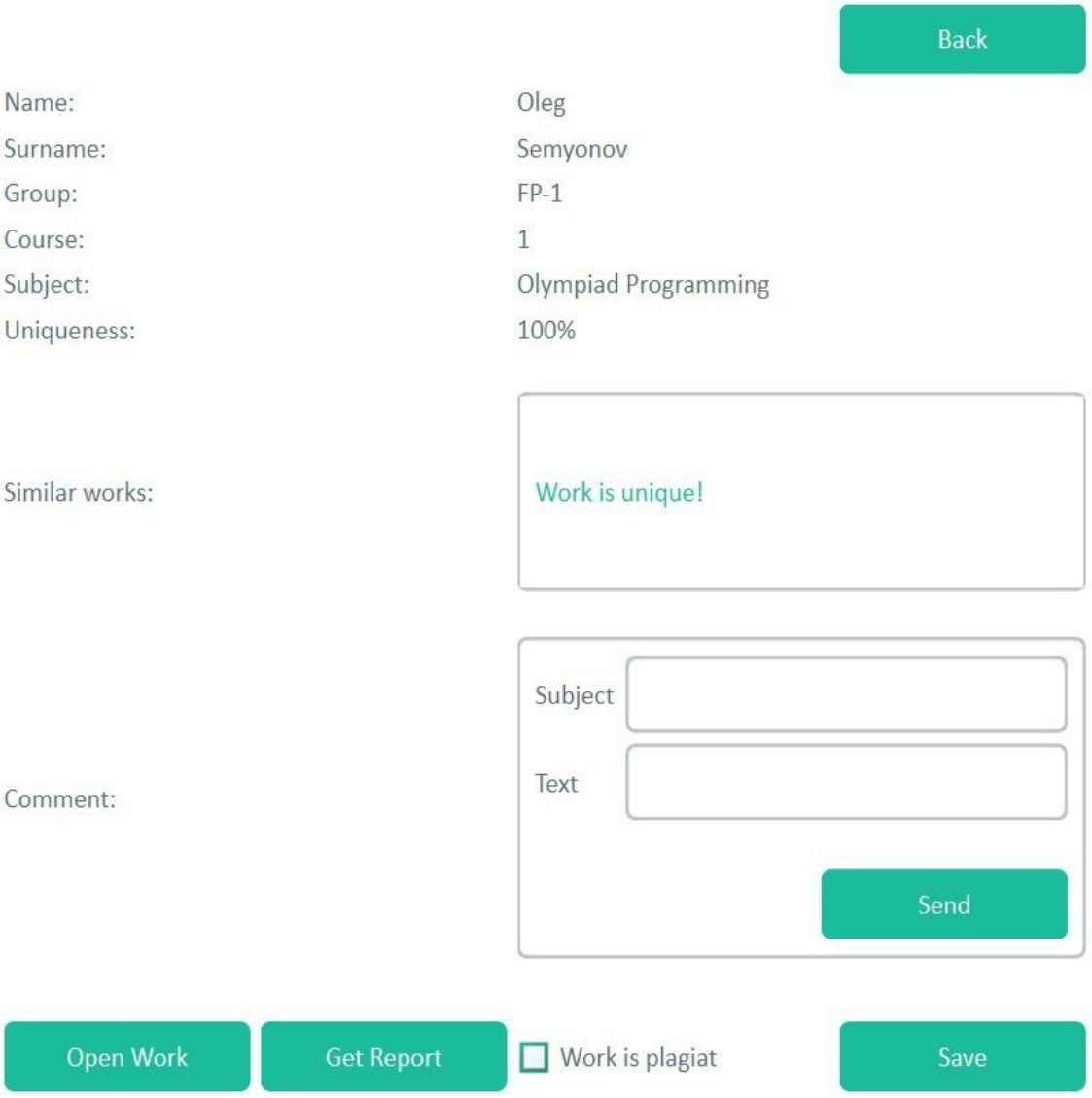

Рисунок 4.6 – Форма перегляду та аналізу роботи студента (унікальна робота)

Форма для перегляду викладачем усіх робіт, надісланих студентами, включає наступні функціональні можливості:

– Вибір предмета: якщо викладач веде кілька дисциплін, він може обрати потрібний предмет зі списку або за допомогою випадаючого меню. Це спрощує перегляд робіт для кожної дисципліни окремо.

– Список робіт: таблиця або список, у якому відображаються роботи, надіслані студентами. У ньому можуть бути такі стовпці, як ім'я студента, назва роботи, дата подання та статус перевірки.

– Функції сортування та фільтрації: для зручності користування форма може мати інструменти для сортування робіт за різними критеріями (дата, статус, студент) або фільтрації, наприклад, за групою студентів.

– Кнопка перегляду роботи: кожна робота має бути доступною для перегляду в один клік, наприклад, натискаючи на відповідну кнопку або посилання в списку.

– Кнопка оцінки роботи: викладач може швидко перейти до виставлення оцінки або додавання коментарів для кожної конкретної роботи.

Ця форма забезпечує викладачу зручний інтерфейс для організації перевірки та оцінювання студентських робіт.

Звіти проаналізованих робіт студентів автоматично формуються у файли з розширенням .html. Вони відкриваються у браузері, який налаштований на комп’ютері як браузер за замовчуванням.

Основні можливості роботи зі звітами:

Автоматичне відкриття: Після генерації звіт автоматично відкривається у браузері, що забезпечує зручність перегляду.

Збереження звіту: Викладач має можливість зберегти звіт для подальшого аналізу або використання на локальному комп’ютері. Для цього можна скористатися стандартними функціями браузера, які дозволяють зберігати файл у таких форматах:

– .html: Збереження вихідного файлу для подальшого відкриття у будь-якому веб-браузері.

– .pdf: Експорт у формат для друку або передачі іншим особам.

Переваги формату звіту:

1. Кросплатформенність: Звіт у форматі .html підтримується всіма сучасними браузерами.

2. Можливість редагування: У форматі .html звіт може бути адаптований до потреб викладача, наприклад, додавання коментарів чи зміна структури.

3. Зручність передачі: Формат .pdf забезпечує зручність передачі звіту між користувачами без ризику зміни вмісту.

Цей підхід підвищує ефективність роботи викладача, забезпечуючи гнучкість у збереженні та подальшому використанні звітів.

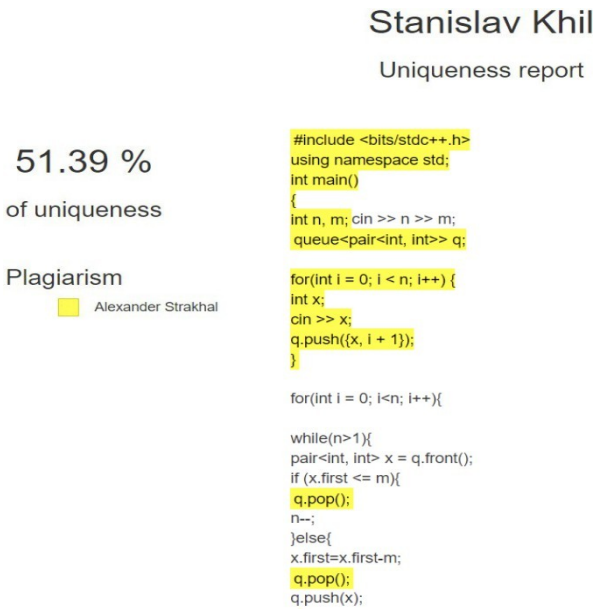

Рисунок 4.7 – Приклад звіту роботи студента з відсотком унікальності 51.39%

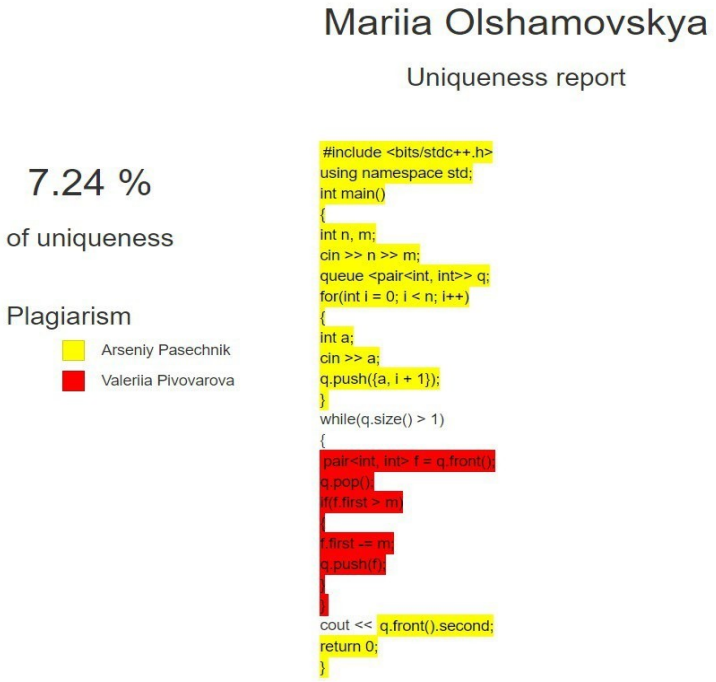

Рисунок 4.8 – Приклад звіту роботи студента з відсотком унікальності 7.24%

Форма Звіт про перевірку роботи студента містить такі ключові елементи:

1. Ідентифікаційні дані студента:

– Ім'я та прізвище студента, чию роботу переглядає викладач.

2. Аналіз тексту студентської роботи:

– Повний текст роботи з виділенням частин, які система визначила як запозичення з інших джерел.

3. Відсоток унікальності:

– Показник, що відображає ступінь оригінальності роботи.

4. Список робіт із схожими фрагментами:

– Перелік студентів, роботи яких містять фрагменти, схожі на текст даної роботи. Для кожного зі збігів визначено відповідний колір, яким позначено запозичені частини.

5. Колірна схема для відображення запозичень:

– Система використовує 20 фіксованих кольорів для позначення збігів із різними джерелами.

– У випадках, коли кількість джерел перевищує 20, для запозичень, пов’язаних із джерелами під номерами 21 і більше, застосовується жовтий колір.

Такий підхід забезпечує зручність візуального аналізу навіть за умов великої кількості збігів.

Ця структура звіту дозволяє викладачу легко і швидко проаналізувати текст роботи, виявити запозичення та визначити їх джерела, що сприяє ефективному контролю за академічною доброчесністю.

4.3 Функціональне тестування

Тестування програмного забезпечення – це процес перевірки та оцінювання роботи системи для забезпечення її відповідності вимогам, виявлення помилок та забезпечення якості функціонування.

1. Завантаження роботи в систему (Студент).

Мета: Перевірити, чи студент може завантажити свою роботу в систему.

Сценарій 1.1: Завантаження файлу допустимого формату.

Кроки:

Увійти до системи як студент.

Перейти до розділу завантаження роботи.

Завантажити файл у форматі .docx.

Підтвердити дію.

Очікуваний результат: Файл успішно завантажено, з'являється повідомлення про успіх.

Сценарій 1.2: Завантаження файлу недопустимого формату.

Кроки:

1. Завантажити файл у форматі .exe.

Очікуваний результат: Відображається повідомлення про помилку, файл не завантажено.

2. Перевірка роботи на антиплагіат.

Мета: Перевірити коректність перевірки на антиплагіат.

Сценарій 2.1: Робота не містить плагіату.

Кроки:

Завантажити унікальний текстовий документ.

Запустити перевірку.

Очікуваний результат: Система вказує, що плагіат не виявлено.

Сценарій 2.2: Робота містить частковий плагіат.

Кроки:

1. Завантажити документ, який містить скопійовані частини.

2. Запустити перевірку.

Очікуваний результат: Система показує відсоток збігів і виділяє частини тексту, що містять плагіат.

3. Додавання плагіатчиків до списку (Система).

Мета: Перевірити автоматичне додавання студентів до списку плагіатчиків.

Сценарій 3.1: Робота визначена як плагіат.

Кроки:

Завантажити файл із високим рівнем збігів (>70%).

Система проводить перевірку на антиплагіат.

Система визначає роботу як плагіат.

Автоматично додає студента до списку плагіатчиків.

Очікуваний результат:

Робота визначена як плагіат.

Студента внесено до списку плагіатчиків із відповідною позначкою в базі даних.

4. Перегляд робіт студентів (Викладач).

Мета: Забезпечити можливість викладачеві переглядати роботи, завантажені студентами.

Сценарій 4.1: Викладач переглядає всі роботи.

Кроки:

Увійти до системи як викладач.

Перейти до розділу "Роботи студентів".

Переглянути список завантажених робіт із деталями (студент, дисципліна, статус перевірки).

Очікуваний результат: Система відображає список робіт із фільтрацією за студентами, дисциплінами, статусами.

Сценарій 4.2: Викладач відкриває конкретну роботу.

Кроки:

Обрати одну роботу зі списку.

Натиснути «Переглянути».

Очікуваний результат: Відображається зміст роботи з індикатором перевірки на плагіат.

5. Аналіз робіт студентів (Викладач)

Мета: Перевірити можливість викладача аналізувати результати перевірки.

Сценарій 5.1: Аналіз роботи без плагіату.

Кроки:

Викладач відкриває роботу, яка не містить плагіат.

Перевіряє результат антиплагіатного тесту.

Очікуваний результат: Система показує роботу без виділених плагіатних частин.

Сценарій 5.2: Аналіз роботи з плагіатом.

Кроки:

Викладач відкриває роботу з плагіатними збігами.

Перевіряє виділені плагіатні фрагменти та відсоток збігів.

Очікуваний результат: Виділено текстові частини зі збігами, надано відсоток подібності.

6. Оновлення кінцевого списку плагіатчиків (Викладач/Система)

Мета: Переконатися, що список плагіатчиків оновлюється після внесення нових даних.

Сценарій 6.1: Система автоматично оновлює список.

Кроки:

Студент завантажує роботу з плагіатом.

Система автоматично додає студента до списку.

Викладач переглядає оновлений список.

Очікуваний результат: У списку плагіатчиків з'являється новий запис із даними студента.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи були виконані наступні завдання:

1. Модифікація алгоритму локально-чутливого хешування. Було додано попередню стандартизацію текстів програм, що дозволило значно покращити ефективність алгоритму для перевірки текстів на плагіат. Завдяки цій модифікації алгоритм набув асимптотичної складності O(n), де n – сумарна довжина текстів у корпусі.

2. Проєктування та розробка системи для перевірки робіт студентів на плагіат. Система базується на модифікованому алгоритмі локально-чутливого хешування та має такі ключові функції:

- клієнт-серверна архітектура: сервер виконує основні обчислення та зберігання даних, тоді як клієнт забезпечує взаємодію з користувачами через зручний інтерфейс. Використання технології тонкого клієнта дозволяє знизити навантаження на комп’ютери користувачів;
- перевірка робіт на плагіат: студент може завантажити свою роботу для перевірки на плагіат;
- доступ для викладачів: викладач може переглядати роботи всіх студентів, а також детально аналізувати роботу конкретного студента, включаючи відсоток її унікальності;
- надсилання електронних листів: викладач має можливість відправляти повідомлення на електронні пошти студентів;
- генерація звітів: викладач може отримувати детальні звіти з порівнянням роботи студента з іншими схожими роботами.

3. Тестування системи. Були проведені тести для оцінки ефективності роботи системи, її здатності обробляти запити та надавати необхідні результати з перевірки на плагіат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Великий тлумачний словник сучасної української мови. – К.; Ірпінь; ВТФ «Перун», 2005. – 977 с.

2. Як перевірити текст на унікальність на плагіат [Електронне видання]: – Режим доступу: https://ag.marketing/yak-pereviryty-tekst-na-unikalnist/.

3. Куперштейн Л. М. Аналіз методів перевірки тексту на плагіат / Л. М. Куперштейн, М. Я. Мельник. – Вінниця, 2014. – 4 с.

4. Gorenko Y. Common Ways Students Avoid Plagiarism [Електронне видання]: – Режим доступу: https://www.teachthought.com/technology/.

5. Ванюшкін А.С., Гращенко Л.А. Оцінка алгоритмів вилучення ключових слів: інструментарій та ресурси // Нові інформаційні технології в автоматизованих системах, 2017. –Вип. 20.

6. Similarity Percentage Computation in SIM / Dick Grune, 2016. [Електронне видання]: – Режим доступу: https://dickgrune.com/Programs/similarity_tester/.

7. JPlag – Detecting Software Plagiarism [Електронне видання]: – Режим доступу: https://github.com/jplag/JPlag/wiki.

8. Efficient Source Code Plagiarism Identification Based on Greedy String Tilling / Khurram Zeeshan Haider, Tabassam Nawaz, Sami ud Din, Ali Javed. – Taxila, Pakistan, 2010.

9. Chen X., Francia B., Li M., Mckonnon B., Seker A. Shared information and program plagiarism detection. – University of California, Santa Barbara, December 13, 2003.

10. Зельцер Н. Г. Пошук фрагментів вихідного коду, що повторюються, при автоматичному рефакторингу // Праці Інституту системного програмування РАН, 2013. – Т. 25.

11. Патерни проектування / Е. Фрімен, Е. Робсон, Б. Бейтс, К. Сієрр; Видавництво “Фабула” – К., 2020 – 672 с.