НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

_____Ігор БОЛБОТ
(підпис)

«___» _____ 2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: "Програмне забезпечення системи перевірки плагіату"

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент **_____ Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
науковий ступінь та вчене звання **_____ С.В. ВАСИЛЮК-ЗАЙЦЕВА**
(підпис)

Виконав **_____ Владислав БІЛИК**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Білику Владиславу Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення системи перевірки плагіату»
затверджена наказом від «19».12.2025р. № 3196«С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту предметне оформлення текстових запозичень; вимоги до системи перевірки плагіату; алгоритми аналізу подібності текстів; структура БД; стек Python, FastAPI, PostgreSQL, HTML, CSS, JavaScript.
Перелік питань, що підлягають дослідженню:

1. аналіз предметної області та існуючих систем перевірки плагіату;
2. Формування вимог та постановка завдання;
3. Проектування UML-моделей і архітектури системи;
4. Розробка БД, серверної та клієнтської частин системи;
5. Реалізація, тестування системи та підготовка рекомендацій.Перелік графічних документів (за необхідності).

ER-діаграма БД; UML-діаграми варіантів використання, класів, компонентів; скріншоти інтерфейсу.
Дата видачі завдання «___»_____202 р.

Керівник бакалаврської кваліфікаційної роботи _____ **(підпис)** **Світлана ВАСИЛЮК-ЗАЙЦЕВА**

Завдання взяв до виконання _____ **(підпис)** **Владислав БІЛИК**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД - система управління базами даних

API - Application Programming Interface — інтерфейс прикладного програмування

CASE - Computer-Aided Software Engineering — автоматизоване проектування програмного забезпечення

ER - Entity-Relationship — сутність-зв'язок

GCC - GNU Compiler Collection — колекція компіляторів GNU

GNU - GNU's Not Unix — рекурсивна абревіатура; основа відкритого програмного середовища

HTTP - HyperText Transfer Protocol — протокол передавання гіпертексту

IP - Internet Protocol — інтернет-протокол

JSON - JavaScript Object Notation — текстовий формат обміну даними

JWT - JSON Web Token — токен авторизації на основі JSON

MIME - Multipurpose Internet Mail Extensions — багатоцільові розширення пошти Інтернет

RFC - Request for Comments — офіційний технічний документ стандартів Інтернет

SOA - Service-Oriented Architecture — сервісно-орієнтована архітектура

SQL - Structured Query Language — мова структурованих запитів

SSL - Secure Sockets Layer — рівень захищених сокетів

SMTP - Simple Mail Transfer Protocol — протокол простої передачі пошти

SMTPS - Simple Mail Transfer Protocol Secure — захищений протокол передачі пошти

SSD - Solid State Drive — твердотільний накопичувач

TCP - Transmission Control Protocol — протокол керування передаванням

TLS - Transport Layer Security — безпека транспортного рівня

UML - Unified Modeling Language — уніфікована мова моделювання

URL - Uniform Resource Locator — уніфікований локатор ресурсу

ВСТУП

Сьогодні розвиток інформаційних технологій характеризується активним впровадженням цифрових ресурсів у навчальну та наукову сферу. Значна частина освітніх процесів так чи інакше базується на використанні електронних документів, онлайн-бібліотек і відкритих джерел інформації. Це значно спрощує доступ до матеріалів і їх опрацювання, проте одночасно підвищує ризик некоректного використання чужих текстів. У практиці закладів освіти це часто проявляється у вигляді некоректно оформлених запозичень або частково змінених фрагментів, що ускладнює перевірку їх оригінальності.

У зв’язку з цим питання академічної доброчесності стає особливо актуальним і потребує ефективних засобів контролю унікальності текстових документів. Це особливо важливо в умовах високого навантаження на викладачів, коли необхідно перевіряти велику кількість робіт у короткі строки. За таких обставин автоматизовані системи перевірки плагіату розглядаються як необхідний інструмент, що допомагає підвищити об’єктивність оцінювання та зменшити суб’єктивний вплив під час перевірки.

Метою даної дипломної роботи є розробка програмної системи перевірки академічного плагіату, яка забезпечує автоматизований аналіз текстових документів, визначення ступеня їх схожості та формування результатів у зручному для подальшого використання вигляді. Практична цінність такої системи полягає у можливості її застосування в навчальному процесі для підвищення якості контролю студентських робіт і оптимізації часу перевірки.

Об'єкт дослідження: Процес автоматизованого виявлення текстових запозичень.

Предмет дослідження: Програмні засоби та алгоритми pre-processing і порівняння текстів на наявність плагіату.

Практичне значення: Створення автономного сервісу для перевірки студентських робіт.

У процесі реалізації програмного продукту застосовуються сучасні методи обробки текстової інформації. Для виявлення структурних збігів використовуються алгоритми shingling та fingerprinting, які дозволяють порівнювати документи на основі їх фрагментів. Додатково застосовуються підходи обробки природної мови (NLP), що дають можливість враховувати не лише формальні збіги, а й часткову семантичну подібність текстів. Реалізація системи виконана мовою програмування Python, яка є зручною для роботи з текстовими даними та має широкий набір бібліотек для їх аналізу. Для зберігання інформації використовується реляційна база даних SQL, у якій фіксуються документи, результати перевірок і дані користувачів. Взаємодія між компонентами системи реалізована через REST API, що забезпечує обмін даними між клієнтською та серверною частинами.

Структура дипломної роботи визначається логікою виконання дослідження і складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 86 сторінок, містить 32 джерела, 10 рисунків, 6 таблиць і 4 додатки.

У першому розділі виконано аналіз теоретичних аспектів академічної доброчесності, розглянуто різні підходи до класифікації плагіату та основні методи виявлення текстових запозичень. У другому розділі основну увагу приділено проєктуванню програмної системи, зокрема визначенню її архітектури та побудові структури бази даних. Третій розділ містить опис реалізації програмного забезпечення, а також результати проведеного тестування. У четвертому розділі розглянуто питання охорони праці, організації робочого місця розробника та забезпечення інформаційної безпеки.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМ ПЕРЕВІРКИ ПЛАГІАТУ

1.1. Поняття плагіату та академічної доброчесності

Проблема плагіату в освітньому середовищі вже давно не обмежується лише формальними питаннями правильного оформлення джерел у навчальних роботах. Насправді вона є значно глибшою, оскільки напряму пов’язана з формуванням самостійності мислення студентів та з тим, наскільки об’єктивно можна оцінювати результати їхньої навчальної і наукової діяльності. У загальному розумінні плагіат визначається як використання або привласнення результатів інтелектуальної праці іншої особи без належного зазначення авторства [1]. При цьому важливо розуміти, що до таких результатів належать не лише текстові матеріали, а й ідеї, методичні підходи, логіка побудови дослідження та інші форми інтелектуальної діяльності, які можуть бути запозичені без посилання на першоджерело.

На практиці плагіат досить рідко проявляється у вигляді повного копіювання великих фрагментів тексту, частіше зустрічаються випадки часткового перефразування, заміни окремих слів або зміни структури речень при збереженні загального змісту. Саме такі ситуації є найбільш складними для однозначного виявлення, оскільки формально текст може виглядати оригінальним, хоча фактично він не є результатом самостійної інтелектуальної роботи. З розвитком цифрових технологій і масовим переходом до використання онлайн-джерел ситуація ще більше ускладнилася, оскільки доступ до інформації став практично миттєвим і необмеженим, що з одного боку спрощує процес навчання, але з іншого створює ризик поверхневого використання матеріалів без належного осмислення або коректного цитування. У таких умовах межа між допустимим використанням джерел і порушенням академічних правил стає менш чіткою, що і призводить до зростання кількості некоректних запозичень [2].

У межах академічного середовища плагіат розглядається не лише як технічне порушення вимог до оформлення, а як більш серйозне відхилення від

принципів академічної доброчесності, яка передбачає чесність у виконанні робіт, самостійність дослідження та коректне використання джерел [3]. Ці принципи фактично формують основу довіри до освітнього процесу, оскільки без їх дотримання складно говорити про об’єктивність оцінювання знань або реальний рівень підготовки студентів. Водночас важливо враховувати, що академічна доброчесність у багатьох країнах, включно з Україною, має нормативне закріплення і регулюється відповідними законодавчими актами [4], тому порушення у вигляді плагіату можуть мати не лише внутрішні навчальні наслідки, такі як повернення роботи на доопрацювання або зниження оцінки, але й більш серйозні форми відповідальності у випадку систематичних порушень, через що питання доброчесності виходить за межі внутрішніх правил закладів освіти і стає частиною ширшої освітньої та правової системи.

Якщо розглядати види плагіату більш детально, можна виділити кілька основних форм, які суттєво відрізняються за складністю виявлення. Найпростішою є пряма форма, коли текст копіюється без змін і без посилання на джерело, і такі випадки зазвичай досить легко виявляються автоматизованими системами, оскільки базуються на точному збігу фрагментів. Більш складним варіантом є так званий мозаїчний плагіат, коли текст формується шляхом поєднання фрагментів із різних джерел, і на перший погляд може виглядати як цілісний авторський матеріал, хоча фактично він складається із запозичених частин. Ще складнішим є перефразований плагіат, при якому зберігається зміст першоджерела, але повністю змінюється формулювання, і саме цей тип є найбільш проблемним для автоматичного виявлення, оскільки потребує не лише аналізу текстових збігів, а й оцінки семантичної подібності між фрагментами [5]. Окремо слід згадати явище самоплагіату, яке полягає у повторному використанні власних раніше опублікованих матеріалів без відповідного посилання, і хоча в цьому випадку не порушуються авторські права інших осіб, у межах академічної доброчесності це також вважається проблемним, оскільки може створювати ілюзію новизни результатів та викривляти реальний обсяг дослідження [3].

З погляду інформаційних технологій плагіат можна розглядати як задачу визначення ступеня схожості між текстовими документами, де складність полягає в тому, що різні типи запозичень потребують різних підходів до аналізу. У простих випадках достатньо пошуку точних збігів, тоді як у складніших необхідно застосовувати методи обробки природної мови та семантичного аналізу, що дозволяють враховувати змістову подібність текстів. Саме тому сучасні системи перевірки унікальності зазвичай будуються як комбіновані рішення, у яких кілька алгоритмічних підходів працюють разом і компенсують обмеження один одного [2]. Окрему роль відіграють також сучасні методи аналізу, що базуються на машинному навчанні, які дозволяють виявляти глибші зв’язки між текстами, однак при цьому потребують більших обчислювальних ресурсів і складнішої архітектури систем.

У підсумку можна сказати, що проблема плагіату виходить далеко за межі суто технічного завдання, оскільки вона безпосередньо пов’язана з етичними принципами освіти та рівнем довіри до результатів навчання. Саме тому питання академічної доброчесності поступово стає невід’ємною частиною освітнього процесу, а не лише формальною вимогою [3]. Автоматизовані системи в цьому контексті виконують допоміжну функцію, оскільки дозволяють швидко виявляти потенційні збіги в текстах, але не можуть повністю замінити експертну оцінку, яка враховує зміст, контекст і загальну логіку роботи, тому остаточне рішення завжди залишається за людиною.

1.2. Класифікація видів плагіату та методи його виявлення

Плагіат як явище не має однозначного або універсального визначення, оскільки в реальних умовах він проявляється у різних формах і на різних рівнях опрацювання тексту. У науковій та освітній практиці його зазвичай описують через поєднання двох основних ознак: ступінь зміни первинного матеріалу та спосіб приховування факту запозичення [1]. Саме ці характеристики фактично і визначають, наскільки складно виявити той чи інший тип плагіату автоматизованими засобами, а також які методи аналізу будуть ефективними у конкретному випадку.

Якщо подивитися на проблему з боку інформаційних систем, то стає зрозуміло, що складність виявлення напряму залежить від того, як саме був змінений вихідний текст. У найпростішому випадку, коли йдеться про пряме копіювання, достатньо базових алгоритмів порівняння. Але коли текст переробляється, змінюється структура або замінюються слова, задача ускладнюється і потребує більш глибокого аналізу. Через це більшість сучасних систем перевірки унікальності будуються не як один алгоритм, а як поєднання кількох рівнів обробки, де кожен рівень відповідає за свій тип запозичень.

Найпростішим варіантом є прямий плагіат, який фактично зводиться до дослівного копіювання тексту без змін. У такій ситуації система може працювати з точними збігами фрагментів і визначати подібність на основі порівняння послідовностей символів або слів. Подібні випадки зазвичай виявляються досить швидко, оскільки структура тексту повністю повторюється. Проте навіть тут іноді виникають дрібні ускладнення, пов’язані з форматуванням або незначними вставками, які можуть частково приховувати збіги, хоча суттєво на результат це не впливає.

Більш складною формою є мозаїчний плагіат, коли текст збирається з різних джерел і при цьому окремі фрагменти можуть бути трохи змінені або переставлені місцями. У такому випадку вже недостатньо дивитися лише на локальні збіги, оскільки необхідно брати до уваги загальну структуру документа. Часто саме логіка побудови тексту видає запозичення, навіть якщо окремі частини виглядають достатньо відмінними. Тому для таких випадків застосовуються методи, які аналізують не тільки самі фрагменти, а й їх послідовність та взаємозв’язки в межах усього документа.

Ще більш складним є перефразований плагіат, коли зберігається зміст, але суттєво змінюється форма подачі. Саме цей тип найважче виявляється класичними методами, оскільки прямі збіги можуть бути відсутні. Тут вже необхідно враховувати семантичну схожість текстів, тобто не тільки те, які слова використані, а й те, яке значення вони передають у контексті. Як показує практика

досліджень, саме цей варіант є найбільш проблемним для автоматизованих систем, оскільки вимагає більш складних моделей аналізу [2].

Серед іншого виділяється самоплагіат, який полягає у повторному використанні власних раніше опублікованих матеріалів без відповідного посилання. Хоча з формальної точки зору автор не порушує права інших осіб, у межах академічної доброчесності така практика розглядається як некоректна, оскільки створюється ілюзія новизни там, де її фактично немає. Для систем перевірки це також може бути проблемним випадком, адже тексти одного автора часто мають високу схожість навіть без прямого копіювання.

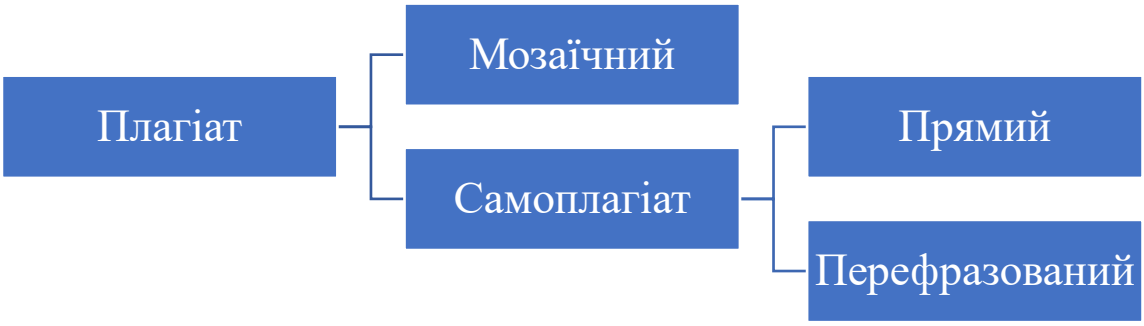


Рисунок 1.1 – Узагальнена класифікація видів плагіату

На рисунку 1.1 представлено узагальнену класифікацію видів плагіату, яка відображає основні підходи до його поділу залежно від складності та характеру запозичення. Загальна логіка цієї схеми полягає в тому, що всі види розташовані умовно від найпростішого до найбільш складного для виявлення. Такий поділ дозволяє краще зрозуміти, чому різні випадки плагіату потребують різних методів аналізу: якщо пряме копіювання можна виявити досить швидко, то перефразовані або комбіновані варіанти вимагають більш глибокої перевірки змісту та структури тексту. Фактично ця схема задає основу для вибору алгоритмів, які використовуються в системах перевірки унікальності.

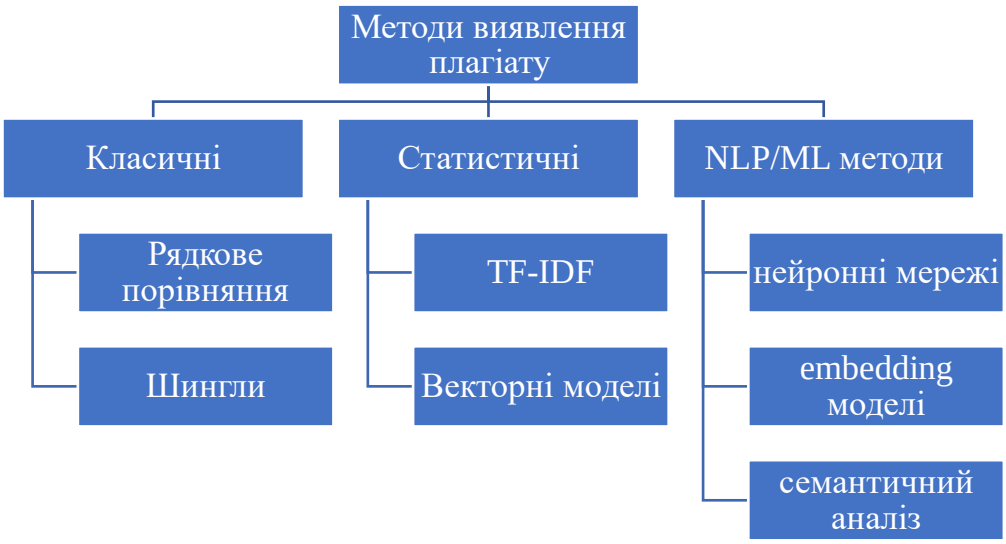


Рисунок 1.2 – Класифікація методів виявлення плагіату

На рисунку 1.2 наведено класифікацію методів виявлення плагіату, яка показує, як еволюціонували підходи до аналізу текстів. У нижній частині схеми зазвичай розташовані прості методи прямого порівняння, які працюють із точними збігами і не враховують зміст тексту. Вище знаходяться методи шинглів, які вже дозволяють аналізувати текст через перекривні послідовності слів, що робить їх більш чутливими до часткових змін. Окрему групу становлять статистичні методи, зокрема TF-IDF підхід, який оцінює важливість слів у документі відносно всієї колекції текстів. На верхньому рівні знаходяться більш складні інтелектуальні підходи, які вже можуть враховувати семантику і контекст, але потребують значно більших обчислювальних ресурсів. Така ієрархія дозволяє зрозуміти, чому на практиці часто використовується комбінація різних методів, а не один універсальний алгоритм.

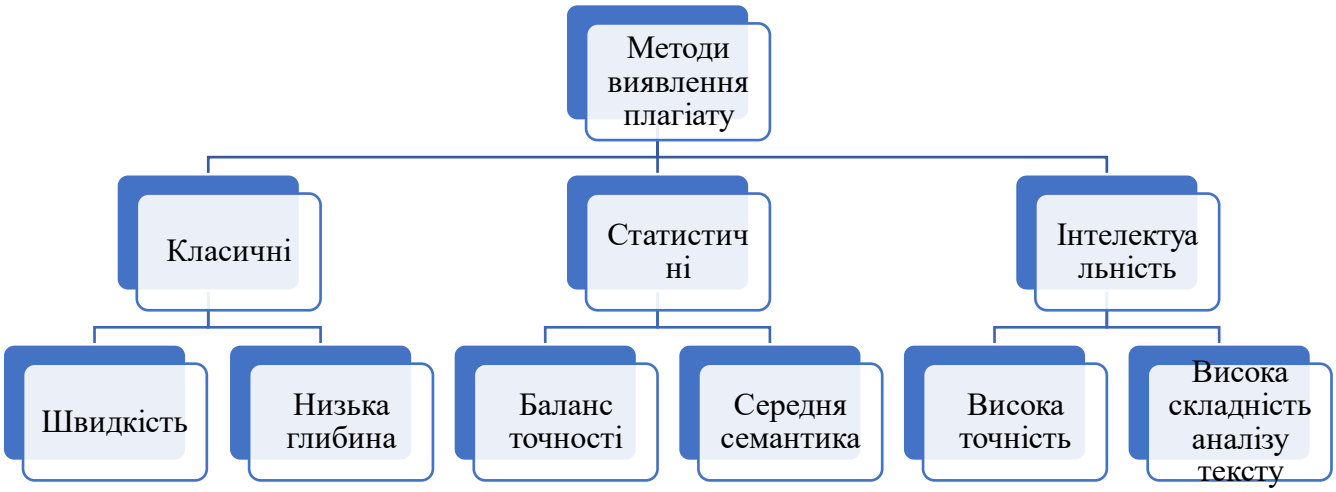


Рисунок 1.3 – Порівняльна характеристика основних підходів до виявлення плагіату

На рисунку 1.3 наведено порівняльну характеристику основних підходів до виявлення плагіату, де зіставляються класичні, статистичні та інтелектуальні методи за ключовими параметрами. У загальному вигляді видно, що прості методи мають перевагу в швидкості обробки та простоті реалізації, але поступаються в точності при складних типах запозичень. Статистичні методи займають проміжне положення, оскільки дозволяють отримати більш збалансований результат між точністю та продуктивністю. Інтелектуальні методи, у свою чергу, демонструють найкращу якість аналізу, проте є найбільш ресурсомісткими і складними у впровадженні. Саме тому ця схема добре ілюструє практичний компроміс між швидкістю роботи системи та глибиною аналізу тексту, який завжди враховується при розробці подібних програмних рішень.

Узагальнюючи, можна сказати, що класифікація плагіату та методи його виявлення формують єдину взаємопов’язану систему, яка визначає принципи побудови сучасних інформаційних рішень у сфері контролю унікальності текстів. Ефективна реалізація таких систем зазвичай передбачає поєднання різних підходів, оскільки це дозволяє враховувати як структурні особливості тексту, так і його семантичний зміст.

1.3. Огляд існуючих програмних систем перевірки плагіату

Сучасні системи перевірки плагіату фактично є досить складними програмними комплексами, які поєднують обробку тексту, великі бази даних і алгоритми порівняння документів. Їх основне завдання полягає у тому, щоб знайти схожі фрагменти між перевірюваним текстом і вже існуючими джерелами – це можуть бути наукові статті, веб-ресурси, студентські роботи або внутрішні архіви навчальних закладів. Загальна ідея проста, але реалізація вже значно складніша, оскільки сучасні тексти можуть бути сильно зміненими або перефразованими, що ускладнює пряме порівняння.

Якщо подивитися на розвиток таких систем у динаміці, то видно, що вони пройшли шлях від простого пошуку збігів рядків до багаторівневих алгоритмів аналізу. У більш нових рішеннях вже використовується сегментація тексту, побудова шинглів, різні хеш-представлення та частково методи семантичного аналізу [10]. Тобто система працює не лише з “буквальними” збігами, а намагається аналізувати структуру тексту більш глибоко, хоча ця глибина все одно обмежена.

У межах цього розділу розглядаються кілька найбільш відомих систем: Turnitin, Unicheck, PlagScan та iThenticate. Вони часто згадуються як базові приклади сучасних рішень у сфері перевірки унікальності текстів, але при цьому кожна з них має свою специфіку, як у технічній реалізації, так і у сфері застосування [13].

Turnitin можна вважати однією з найпоширеніших систем у сфері освіти. Вона використовується у багатьох університетах і працює на основі хмарної архітектури, де кожен документ проходить послідовну обробку: спочатку відбувається очищення тексту і його нормалізація, потім поділ на фрагменти, після чого формується набір шинглів, які вже порівнюються з великою базою даних [10]. Така база включає інтернет-джерела, наукові публікації і внутрішні студентські роботи, що дозволяє знаходити як зовнішні, так і внутрішні збіги.

Turnitin має велику внутрішню базу студентських робіт, яка постійно поповнюється новими матеріалами. Завдяки цьому система здатна виявляти

випадки повторного використання робіт навіть тоді, коли вони були виконані в різний час або в різних навчальних закладах. З технічної точки зору в основі роботи лежить поєднання методів аналізу шинглів і статистичного порівняння текстових фрагментів, що дозволяє ефективно знаходити схожі ділянки тексту. Водночас можливості системи щодо глибокого розуміння змісту залишаються обмеженими, оскільки основний акцент робиться на структурній і лексичній подібності, а не на семантичному аналізі.

У використанні система доволі зручна, оскільки інтегрується з навчальними платформами. Викладач може просто завантажити роботу або отримати її автоматично через LMS, після чого формується звіт із виділеними фрагментами та посиланнями на джерела. Це значно економить час, особливо при великій кількості студентських робіт. Водночас слід враховувати, що система є комерційною і закритою, тому її внутрішні алгоритми не є повністю прозорими, а результати іноді можуть включати спірні збіги через стандартні формулювання.

Unicheck за принципом роботи досить близький до Turnitin, але більше орієнтований на інтеграцію в освітні платформи і автоматизацію перевірки. Він також працює у хмарному середовищі, де документ проходить етапи обробки, розбиття на фрагменти і подальшого порівняння з базами даних та онлайн-джерелами [11]. Основний акцент тут робиться на швидкість і зручність використання у навчальному процесі.

Особливістю Unicheck є те, що результати подаються у вигляді інтерактивного звіту. У ньому користувач бачить не тільки відсоток унікальності, а й конкретні фрагменти тексту, які збігаються з джерелами. Це робить аналіз більш наочним і зрозумілим. При цьому алгоритмічна частина базується переважно на виявленні локальних збігів і подальшому їх узагальненні, тому складні перефразовані тексти можуть визначатися не повністю.

PlagScan відрізняється тим, що надає більше гнучкості у налаштуванні перевірки. Його використовують не тільки в освіті, але й у корпоративному середовищі, де важливо контролювати унікальність внутрішніх документів. Після завантаження текст проходить стандартну обробку: очищення,

нормалізацію, розбиття на фрагменти і порівняння з великими базами даних, які включають як веб-ресурси, так і наукові публікації [12].

Головна особливість PlagScan полягає в можливості змінювати чутливість перевірки. Це означає, що користувач може сам визначати, наскільки “суворо” система буде шукати збіги. З одного боку, це дає гнучкість, але з іншого – може впливати на стабільність результатів, оскільки одна і та сама робота при різних налаштуваннях може давати різні показники унікальності.

iThenticate більше орієнтований на наукову сферу і використовується переважно видавництвами та дослідницькими організаціями. Його основне завдання – перевірка рукописів перед публікацією, тому база даних тут складається в основному з наукових статей, конференційних матеріалів і дисертацій [10]. Це дозволяє системі краще працювати саме з академічними текстами.

Архітектурно iThenticate також базується на хмарній обробці і використовує багаторівневий аналіз тексту. Спочатку документ розбивається на структурні частини, потім формуються індексовані фрагменти, після чого вони порівнюються з великими академічними базами. Особливість полягає в тому, що система краще враховує академічний контекст і може точніше відрізняти цитування від потенційного плагіату.

Водночас ця система теж не позбавлена обмежень. Вона більше орієнтована на наукові тексти, тому у загальних навчальних завданнях її ефективність може бути менш універсальною. Також, як і інші подібні системи, вона не завжди коректно працює з глибоко перефразованими фрагментами, де змінена структура речення, але збережений зміст.

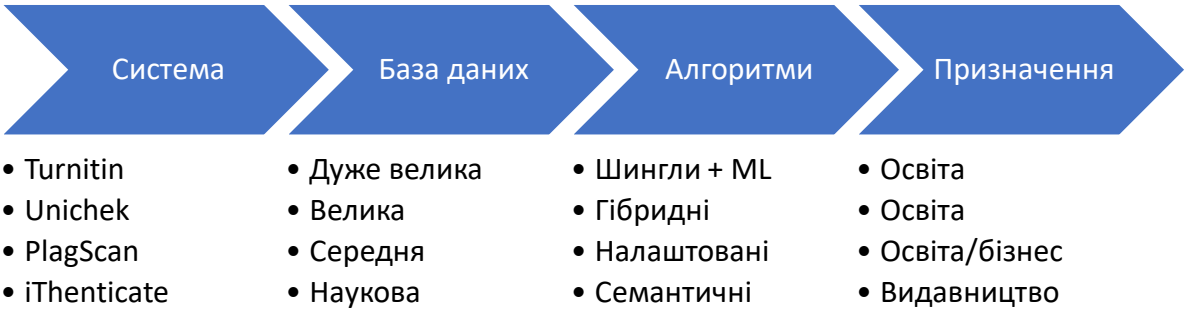


Рисунок 1.3 – Порівняльна характеристика сучасних систем перевірки плагіату

Якщо подивитися на розглянуті системи в цілому, то між ними можна побачити кілька основних відмінностей, які стосуються обсягу їхніх баз даних, складності алгоритмів обробки тексту, глибини семантичного аналізу та сфер, де вони застосовуються. При цьому всі ці рішення працюють за схожою ідеєю – вони порівнюють текстові фрагменти з великими масивами вже наявної інформації, але роблять це з різним рівнем деталізації і за різними внутрішніми механізмами.

Turnitin та Unicheck переважно орієнтовані на освітнє середовище, де основною вимогою є швидка перевірка великої кількості студентських робіт і формування стандартизованих звітів про унікальність. Вони оптимізовані для інтеграції з навчальними платформами та забезпечують автоматизацію процесу перевірки, що робить їх ефективними у масовому академічному використанні. Натомість PlagScan займає проміжну позицію, поєднуючи можливості освітнього та корпоративного застосування завдяки гнучким налаштуванням алгоритмів перевірки та рівня чутливості аналізу. У свою чергу iThenticate є найбільш спеціалізованим рішенням, орієнтованим на наукові видавництва та

рецензування академічних публікацій, де критично важливим є контроль оригінальності дослідницьких текстів перед їх публікацією.

Представлені системи також відрізняються підходами до реалізації алгоритмів аналізу тексту. У більшості випадків використовується комбінація методів шинглів, хешування текстових фрагментів та статистичного аналізу частоти термінів. Однак рівень врахування семантичної подібності суттєво варіюється: від переважно структурного порівняння у більшості освітніх систем до частково семантичного аналізу у спеціалізованих наукових рішеннях. Це свідчить про поступовий перехід від поверхневих методів порівняння до більш інтелектуальних підходів обробки природної мови.

Попри загальний високий рівень розвитку сучасних систем перевірки плагіату, вони мають низку спільних обмежень, які визначають перспективні напрями їх подальшого вдосконалення. Найбільш суттєвою проблемою залишається складність виявлення глибоко перефразованих текстів, які не містять прямих лексичних збігів, але зберігають семантичну ідентичність. Такі випадки особливо складні для алгоритмів, що базуються на поверхневому аналізі тексту, оскільки вони не враховують глибинні смислові зв’язки між елементами тексту [10].

Додатковим обмеженням є залежність ефективності систем від повноти та актуальності індексованих баз даних. Якщо джерело інформації відсутнє у базі, система не здатна виявити запозичення навіть у випадку повного копіювання тексту. Це створює фундаментальну проблему, пов’язану з неповнотою інформаційного простору, в якому працюють подібні системи.

Ще одним важливим аспектом є проблема стандартизації результатів перевірки. Різні системи можуть демонструвати різні значення рівня унікальності для одного й того самого документа через відмінності у алгоритмах аналізу та налаштуваннях чутливості. Це ускладнює порівняння результатів між різними платформами та створює додаткові труднощі для уніфікованої оцінки академічних робіт.

Якщо узагальнити, усі розглянуті системи працюють за схожим принципом – вони порівнюють текстові фрагменти з великими індексованими базами даних. Різниця між ними полягає у масштабі цих баз, рівні інтеграції в освітні або наукові процеси та глибині аналізу тексту. При цьому спільною проблемою залишається те, що складні випадки перефразування і зміни структури тексту все ще важко визначаються автоматично.

Таблиця 1.2 – Порівняльна характеристика аналогів систем перевірки плагіату

Система	**Сфера застосування**	**Швидкість**	**Точність**	**Відкритий API**	**Хмарна інтеграція**
Turnitin	Освіта, наука	Середня	Висока	Ні	LMS, Moodle
Unicheck	Освіта	Висока	Середня	Ні	Google Classroom, Moodle
Antiplagiat	Освіта (РФ, Україна)	Висока	Середня	Так	Хмарна
PlagScan	Освіта, бізнес	Середня	Середня	Так	Хмарна
iThenticate	Наука, видавництво	Середня	Висока	Ні	Хмарна
Розроблена система	Освіта (автономний сервіс)	Висока	Середня/Висока	Так	REST API

Згідно з таблицею 1.2, розроблена система вигідно відрізняється за рахунок відкритого API, автономної роботи без хмарної інфраструктури та високої швидкості обробки документів.

1.4. Аналіз алгоритмів порівняння текстів (shingling, fingerprinting, NLP-методи)

Сучасні системи виявлення плагіату вже давно не працюють на рівні простого порівняння тексту «рядок до рядка», бо в реальних умовах це майже не дає результату. Тексти можуть бути перефразовані, перебудовані, розбиті на частини або навіть переписані так, що зовні виглядають повністю унікальними, хоча зміст фактично той самий. Тому замість одного алгоритму використовується цілий набір підходів, які працюють на різних рівнях аналізу тексту. У класичному варіанті всі ці методи умовно ділять на три великі групи: shingling, fingerprinting і NLP-підходи, які базуються на обробці природної мови [16]. І важливо, що це не просто «різні алгоритми», а фактично різні рівні глибини аналізу тексту, які доповнюють один одного.

Метод shingling зазвичай розглядають як найпростішу і найбільш базову модель, але при цьому він дуже часто використовується як стартовий етап у великих системах. Його суть виглядає доволі просто: текст розбивається на невеликі перекривні фрагменти однакової довжини – шингли, і ці фрагменти далі вже порівнюються між документами. На практиці це означає, що система не «читає» текст як людина, а дивиться на нього як на набір послідовностей. Потім кожен шингл перетворюється у хеш, і вже ці хеші легше порівнювати між собою, ніж сам текст. В результаті отримуємо множини значень, і подібність рахується через кількість збігів. Це добре працює у випадках, коли текст просто скопійований або мінімально змінений, і саме тому такі підходи часто використовуються як первинна фільтрація у великих системах перевірки.

Якщо трохи спростити, то shingling фактично «ловить» структуру тексту, але дуже чутливий до будь-яких змін. Навіть якщо просто переставити слова або замінити частину фраз синонімами, кількість збігів може різко впасти, і система вже не побачить очевидну схожість. Тобто він добре працює на рівні прямого тексту, але майже не розуміє змісту, і це його головне обмеження. Саме тому його майже ніколи не використовують окремо, без додаткових методів.

Fingerprinting-методи можна вважати наступним кроком розвитку цієї ідеї. Тут вже не аналізується весь набір шинглів, а відбирається лише частина найбільш характерних елементів, які фактично формують «відбиток» документа. Ідея в тому, щоб не порівнювати все підряд, бо це занадто довго, а залишити тільки ключові фрагменти, які найбільше описують текст [7]. Вибір цих елементів може відбуватись по-різному: через мінімальні значення хешів, через позиційні правила або через статистичну значущість.

Сам процес виглядає як скорочений варіант shingling: текст розбивається, хешується, але далі система «викидає зайве» і залишає тільки частину даних. Це дуже сильно пришвидшує роботу, особливо коли потрібно перевіряти величезні бази документів. Але тут виникає очевидний компроміс – чим менше даних залишаєш, тим швидше працює система, але тим більше деталей ти втрачаєш. Через це складні випадки плагіату, особливо мозаїчні або перефразовані, можуть визначатися гірше, ніж при повному аналізі.

Найбільш сучасний і складний напрям – це NLP-методи, які вже працюють не з формою тексту, а з його змістом. Тут текст перетворюється у векторне представлення, і фактично кожне речення або документ описується як точка у багатовимірному просторі. Це дозволяє порівнювати тексти не за словами, а за тим, наскільки близький їхній зміст. Саме тому такі методи значно краще справляються з перефразуванням і складними трансформаціями тексту.

Для порівняння документів зазвичай використовується косинусна міра подібності, яка дозволяє оцінити кут між векторами текстів. Якщо він малий – тексти схожі, якщо великий – ні. І це вже зовсім інший рівень аналізу порівняно з shingling або fingerprinting [15]. Але проблема в тому, що такі моделі потребують значно більше ресурсів і якісних даних для навчання. Плюс є ще одна складність – природна мова сама по собі неоднозначна, і одне слово може змінювати значення залежно від контексту, що інколи ускладнює точність результатів.

Якщо подивитися на всі три підходи разом, то вони фактично утворюють не конкуренцію, а певну ієрархію. Shingling працює на рівні структури і швидкого порівняння, fingerprinting – на рівні оптимізації і скорочення даних, а

NLP – на рівні змісту. Саме тому в реальних системах їх зазвичай комбінують, а не використовують окремо. Це добре видно в сучасних рішеннях, де прості алгоритми спочатку відсіюють очевидні збіги, а більш складні моделі вже аналізують спірні або перефразовані частини тексту [15; 16].

Таблиця 1.1.

Порівняльна характеристика основних методів аналізу текстової подібності

Метод	Основний принцип	Сильні сторони	Слабкі сторони	Тип подібності
Shingling	Порівняння перекривних фрагментів	Висока точність для прямих збігів	Чутливість до перефразування	Структурна
Fingerprinting	Відбір репрезентативних фрагментів	Висока швидкість, масштабованість	Втрата частини контексту	Структурна
NLP-методи	Семантичне представлення тексту	Виявлення перефразування і змістовної схожості	Висока складність і ресурсоємність	Семантична

Узагальнення всіх підходів наведено у таблиці 1.1, де показано порівняння основних методів аналізу текстової подібності. У ній видно, що кожен метод має свою «зону ефективності»: shingling найкраще працює зі структурними збігами, fingerprinting – зі швидкою обробкою великих обсягів даних, а NLP – із семантичним аналізом, але з високою обчислювальною складністю. Саме це порівняння і показує, що універсального методу фактично не існує, тому системи змушені комбінувати різні підходи залежно від задачі.

По суті, не існує одного методу, який би однаково добре працював у всіх випадках виявлення плагіату. Саме тому в реальних системах, зокрема в таких як Turnitin, зазвичай поєднують кілька різних підходів до аналізу тексту. Така комбінація дає змогу краще адаптуватися до різних типів запозичень і одночасно підтримувати прийнятну швидкість обробки та достатній рівень точності [7].

1.5. Висновки до розділу 1

У межах першого розділу розглянуто предметну область систем перевірки текстів на плагіат, а також ті підходи, які сьогодні найчастіше використовуються для виявлення текстових запозичень. По суті, це дало можливість краще зрозуміти, як такі системи працюють у реальних умовах, з яких компонентів вони зазвичай складаються і чому на практиці майже ніколи не обмежуються одним алгоритмом. Окрему увагу було приділено загальній логіці їх побудови, оскільки саме архітектура визначає, як обробляються великі обсяги текстових даних і наскільки швидко система може видавати результат.

Аналізуючи існуючі рішення, зокрема Turnitin, Unicheck, PlagScan та iThenticate, стало зрозуміло, що різниця між ними не стільки в базовому принципі перевірки, скільки в організації роботи та масштабі використання. Усі вони так чи інакше спираються на великі індексовані сховища документів і виконують порівняння завантаженого тексту з уже наявними джерелами. Водночас кожна система має свою специфіку: десь акцент зроблений на освітньому процесі, десь на наукових публікаціях або інтеграції з навчальними платформами, але загальна логіка залишається подібною.

В роботі розглядалися основні підходи до аналізу текстової схожості, такі як shingling, fingerprinting та методи, пов’язані з обробкою природної мови. У процесі аналізу стало помітно, що ці підходи фактично працюють на різних рівнях: одні більше орієнтовані на швидке виявлення збігів за структурою тексту, інші – на оптимізацію зберігання та порівняння великих обсягів даних, а більш складні вже намагаються враховувати зміст. При цьому на практиці добре видно різницю між ними: структурні методи дають швидкий результат, але слабко справляються з перефразуванням, тоді як семантичні підходи потенційно точніші, але складніші в реалізації та потребують більше ресурсів.

Якщо підсумувати, то жоден із розглянутих підходів не можна вважати повністю універсальним. Причина тут не лише в обмеженнях самих алгоритмів, а й у тому, що текст як об’єкт аналізу може змінюватися дуже по-різному – іноді це прості заміни слів, а іноді повністю переписаний зміст. Через це на практиці системи перевірки плагіату майже завжди будуються як поєднання кількох

методів, де кожен бере на себе свою частину роботи і частково компенсує слабкі місця інших.

Як підсумок, перший розділ дав змогу сформувати більш цілісне уявлення про предметну область і побачити, як саме працюють сучасні системи перевірки плагіату на практиці. Також стало більш зрозуміло, де саме виникають їх основні обмеження, особливо у випадках складного перефразування тексту. Отримані результати можна вважати базою для подальших розділів, де вже буде розглядатися проєктування та реалізація власної системи перевірки текстових запозичень.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПЕРЕВІРКИ ПЛАГІАТУ

2.1. Постановка задачі та формування вимог до системи

Розробка програмної системи зазвичай починається не з реалізації коду або вибору конкретних технологій, а з більш базового етапу – розуміння того, яку саме задачу потрібно вирішити і в яких умовах система буде використовуватись. Це особливо помітно у випадку систем, що працюють з текстовою інформацією, де важливими є не тільки алгоритми як такі, а й стабільність роботи, коректність результатів і зручність їх інтерпретації. Система перевірки академічного плагіату в цьому сенсі є типовим прикладом рішення, яке потребує чітко визначених вимог ще до початку проєктування. Саме на цьому етапі задаються основні межі майбутньої системи, визначаються її можливості, обмеження та принципи взаємодії між компонентами [14].

Актуальність подібних систем напряму пов’язана зі змінами в освітньому та науковому середовищі, які відбулися внаслідок масової цифровізації. Обсяг доступної інформації суттєво зріс, а доступ до неї став майже необмеженим завдяки Інтернету. З одного боку, це значно спростило процес підготовки навчальних і наукових матеріалів, оскільки необхідні джерела можна знайти швидко і без суттєвих обмежень. Але з іншого боку, така відкритість призвела до того, що процес копіювання і повторного використання текстів також став значно простішим. Через це традиційні методи перевірки, які базуються на ручному аналізі, вже не справляються з великими обсягами даних, і виникає потреба у програмних засобах, здатних автоматично виявляти як прямі, так і приховані форми текстових запозичень.

У процесі аналізу предметної області стає зрозуміло, що обмеження лише на пошук однакових фрагментів тексту є недостатнім. У реальних умовах тексти дуже часто змінюються: можуть перефразовуватись речення, замінюватися окремі слова, змінюватися порядок подачі матеріалу, але при цьому загальний зміст залишається тим самим. Через це система, яка орієнтується тільки на

буквальні збіги, втрачає значну частину потенційних випадків запозичення. Саме тому у вимогах до системи одразу закладається необхідність поєднання різних підходів до аналізу тексту, включаючи як поверхневе порівняння, так і більш складні методи оцінювання семантичної подібності.

Формування вимог значною мірою залежить від середовища, у якому буде використовуватись система. У закладах освіти, як правило, основний акцент робиться на можливості обробки великої кількості студентських робіт, підтримці власних баз документів і інтеграції з навчальними платформами. У таких умовах особливо важливими стають швидкість перевірки та автоматизація процесів, оскільки система повинна справлятися з регулярним і масовим навантаженням. У науковому середовищі пріоритети дещо інші: тут більша увага приділяється точності аналізу, роботі з великими масивами публікацій і здатності виявляти навіть частково змінені запозичення [6]. Через це система має бути спроєктована так, щоб її можна було адаптувати під різні сценарії використання без необхідності повної перебудови архітектури.

Однією з базових функціональних вимог є підтримка різних форматів документів, оскільки в реальному використанні найчастіше зустрічаються файли DOCX, PDF та TXT. При цьому важливо не просто зчитати текст, а зробити це коректно, без втрати структури або змістових фрагментів, які можуть впливати на результати аналізу. Документи часто містять складне форматування, таблиці або вставні елементи, тому система повинна враховувати ці особливості під час обробки [14].

Після завантаження документа запускається етап попередньої обробки тексту, який хоч і не є видимим для користувача, але має критичне значення для якості подальшого аналізу. На цьому етапі відбувається очищення тексту від службових символів, зайвого форматування, іноді виконується нормалізація регістру та приведення тексту до більш уніфікованого вигляду. У деяких випадках також застосовується видалення стоп-слів або додаткове структурування тексту. Якщо цей етап виконано некоректно, навіть найточніші

алгоритми порівняння можуть давати спотворені результати, тому він фактично є одним із ключових у всьому процесі роботи системи.

Основним компонентом системи є модуль аналізу тексту. Враховуючи результати попереднього розділу, доцільним є використання комбінованого підходу, коли різні методи застосовуються послідовно. Спочатку виконується швидке порівняння текстів за допомогою алгоритмів shingling або fingerprinting, які дозволяють виявити потенційні збіги на рівні фрагментів. Далі, у разі необхідності, може застосовуватися більш глибокий аналіз із використанням методів обробки природної мови (NLP), що дозволяє оцінювати не лише форму, а й зміст тексту [17]. Така багаторівнева структура дає змогу збалансувати швидкість роботи та точність результатів.

Не менш важливою вимогою є формування зрозумілого результату перевірки. Користувачеві недостатньо отримати лише загальний відсоток схожості, оскільки це не дає повного уявлення про характер запозичень. Значно інформативнішим є представлення конкретних фрагментів тексту, їх розташування в документі та джерел, з якими вони збігаються. У багатьох випадках доцільно також використовувати візуальне виділення таких фрагментів, оскільки це спрощує аналіз і робить результати більш зрозумілими [6].

На рисунку 2.1 наведено узагальнену схему функціонування системи. Процес починається із завантаження документа користувачем, після чого виконується його попередня обробка та формування внутрішнього представлення тексту. Далі система переходить до етапу пошуку збігів у базі даних або зовнішніх джерелах, а завершальним кроком є формування звіту. Така послідовність дозволяє організувати процес перевірки логічно і забезпечує баланс між швидкістю та точністю аналізу.

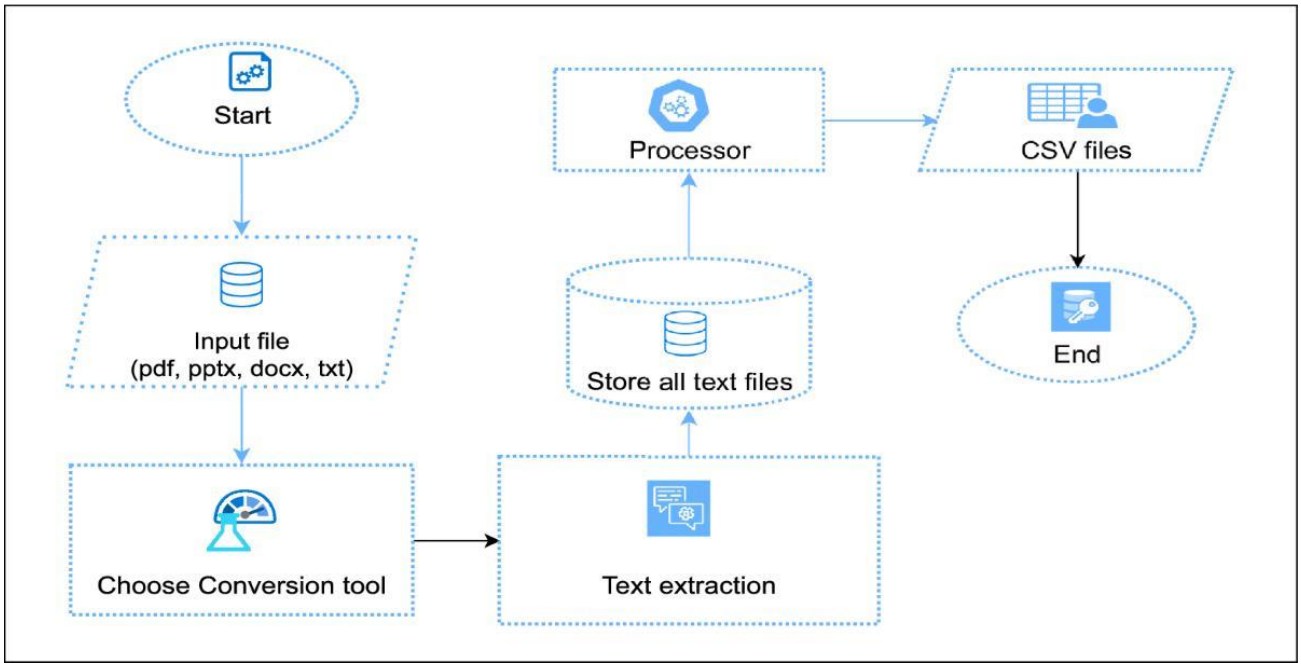


Рисунок 2.1 – Узагальнена схема функціонування системи перевірки плагіату

Сучасна система перевірки академічного плагіату є складним комплексним програмним рішенням, яке поєднує алгоритми аналізу тексту, механізми обробки документів, засоби забезпечення безпеки та інструменти взаємодії з користувачем. Ефективність такої системи визначається не лише точністю виявлення текстових збігів, а й швидкодією, масштабованістю, надійністю та зручністю використання. Саме тому під час її розробки важливо враховувати як алгоритмічні особливості виявлення плагіату, так і практичні аспекти подальшої експлуатації в умовах реального освітнього середовища.

2.2. Загальна архітектура програмного забезпечення

Після визначення основних функцій системи необхідно обрати підхід до її побудови. Архітектура програмного забезпечення визначає, як саме будуть взаємодіяти між собою окремі компоненти системи, яким чином оброблятимуться дані та наскільки зручно буде підтримувати програму в процесі її подальшого розвитку. Для систем перевірки плагіату це має особливе значення, оскільки такі системи працюють із великими текстовими масивами, виконують складний аналіз документів та повинні забезпечувати достатню швидкість роботи навіть при значному навантаженні [17].

Для реалізації систем аналізу тексту найчастіше використовується клієнт-серверна архітектура. Такий підхід дозволяє розділити систему на дві основні частини: клієнтську та серверну. Клієнтська частина відповідає за взаємодію користувача із системою через веб-інтерфейс, тоді як серверна виконує основну обробку даних. Крім цього, система будується за модульним принципом, оскільки процес перевірки плагіату складається з кількох незалежних етапів. До них належать завантаження документа, попередня обробка тексту, аналіз подібності, семантичний аналіз, збереження результатів та формування звіту [20]. Поділ на окремі модулі дозволяє змінювати або вдосконалювати певні компоненти без необхідності змінювати всю систему повністю.

На рисунку 2.2 наведено загальну архітектуру системи перевірки плагіату. Веб-інтерфейс виступає основною точкою взаємодії користувача із системою. Через нього здійснюється завантаження документів, запуск перевірки та перегляд результатів аналізу. Після отримання документа серверна частина системи виконує його обробку, проводить аналіз подібності та формує підсумковий звіт. Отримані результати зберігаються у базі даних і можуть бути використані під час наступних перевірок.

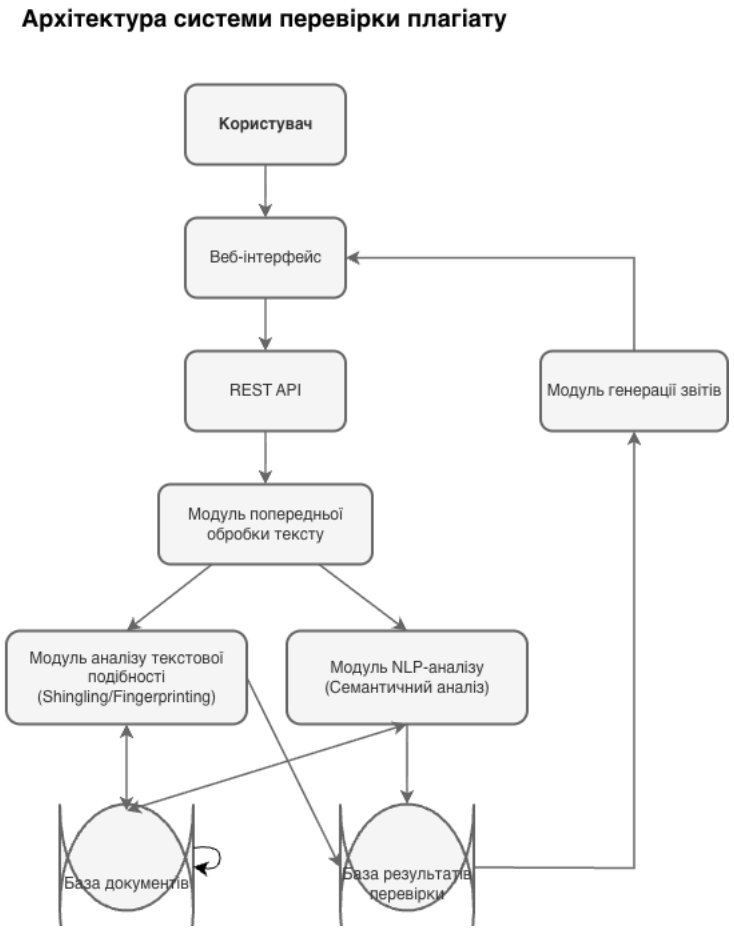


Рисунок 2.2 – Загальна архітектура системи перевірки плагіату

Одним із ключових компонентів системи є модуль попередньої обробки тексту (preprocessing). Саме на цьому етапі текст приводиться до вигляду, придатного для подальшого аналізу. Виконується очищення документа від HTML-тегів, службових символів та елементів форматування. Далі система проводить токенізацію тексту, тобто поділ його на окремі слова або фрагменти, а також нормалізацію регістру. Також здійснюється видалення стоп-слів – слів, які часто зустрічаються у тексті, але практично не впливають на його зміст. До них належать прийменники, сполучники та інші службові частини мови. Попередня обробка є необхідною, оскільки навіть однакові за змістом тексти можуть містити різне форматування або незначні відмінності, що ускладнює їх порівняння.

Для додаткової нормалізації тексту використовуються stemming та lemmatization. Stemming виконує механічне скорочення слова до його основи, тоді як lemmatization визначає правильну початкову форму слова з урахуванням контексту. Наприклад, слова «перевірка», «перевірки» та «перевіркою» система

буде розглядати як одну лексичну форму. Лематизація вважається більш складним методом, проте вона забезпечує вищу точність аналізу тексту, особливо при роботі з природною мовою.

Після завершення попередньої обробки документ передається до модуля аналізу подібності, який реалізує основну логіку виявлення плагіату. Одним із найпоширеніших методів є shingling. Його суть полягає у розбитті тексту на послідовності слів однакової довжини. Наприклад, документ може бути поділений на фрагменти по три або п’ять слів. Для кожного такого фрагмента формується хеш-значення, що дозволяє значно прискорити процес порівняння документів між собою. Якщо два тексти мають велику кількість однакових шинглів, це свідчить про високий рівень їх подібності. Перевагою такого підходу є швидкість роботи та відносна простота реалізації, однак метод має і певні обмеження. Зокрема, його ефективність помітно знижується у випадках значного перефразування тексту.

Для зменшення обчислювального навантаження додатково використовується підхід fingerprinting. Він дозволяє формувати скорочене представлення документа у вигляді набору найбільш характерних фрагментів тексту. Завдяки цьому система працює не з повним текстом документа, а лише з його ключовими частинами, що суттєво прискорює перевірку та зменшує кількість необхідних обчислень [22]. Це особливо важливо у випадках, коли одночасно обробляється велика кількість документів.

Структурні методи добре працюють для пошуку прямих текстових збігів, проте вони не завжди здатні виявляти семантично схожі тексти після перефразування. Саме тому у сучасних системах перевірки плагіату додатково використовуються NLP-компоненти. Вони дозволяють аналізувати не лише окремі слова, а й зміст тексту загалом. Для цього текст перетворюється у векторне представлення (embeddings), після чого система визначає ступінь семантичної близькості між документами.

Сучасні трансформерні моделі, зокрема архітектури типу BERT, враховують контекст слова залежно від його розташування у реченні, що

дозволяє значно підвищити точність семантичного аналізу [23]. Завдяки цьому система може виявляти приховані текстові запозичення навіть у тих випадках, коли структура речень або окремі слова були змінені. Разом із тим використання таких моделей потребує більшої кількості обчислювальних ресурсів, тому найчастіше вони застосовуються як додатковий рівень перевірки для складніших випадків аналізу.

База даних у системі використовується для зберігання документів, результатів перевірки та інформації про користувачів. Крім цього, вона виступає джерелом для порівняння нових документів із текстами, які вже були завантажені раніше. Від структури бази даних та якості індексування безпосередньо залежить швидкість роботи системи. Для реалізації можуть використовуватися як реляційні, так і нереляційні бази даних. Реляційні СУБД забезпечують надійність та підтримку складних запитів, тоді як NoSQL-рішення краще підходять для роботи з великими обсягами текстових даних. Для прискорення пошуку сучасні системи перевірки плагіату використовують технології повнотекстового пошуку та попередньо обчислені представлення документів [17]. Без індексування текстових полів система була б змушена виконувати повний перегляд усіх документів під час кожної перевірки, що значно уповільнювало б її роботу при збільшенні кількості даних.

Взаємодія між клієнтською та серверною частинами реалізується через REST API. Через нього виконуються всі основні операції: завантаження документів, запуск перевірки, отримання результатів та взаємодія з базою даних. Такий підхід дозволяє забезпечити незалежність окремих компонентів системи та спрощує інтеграцію з іншими платформами, наприклад Moodle або Google Classroom.

Окрему увагу необхідно приділити питанням безпеки. Система повинна забезпечувати захист даних користувачів та результатів перевірки. Для цього використовується автентифікація користувачів, шифрування мережевого трафіку за допомогою HTTPS та розмежування прав доступу [24]. Завдяки цьому

користувачі отримують доступ лише до тих функцій і документів, які відповідають їхній ролі у системі.

При збільшенні кількості користувачів або документів система повинна зберігати стабільну продуктивність, тому важливим аспектом є масштабованість. Для цього використовується горизонтальне масштабування, коли до системи можуть додаватися нові серверні екземпляри, а навантаження між ними розподіляється за допомогою балансувальника. Додатково може застосовуватися кешування результатів, що дозволяє зменшити навантаження на базу даних та прискорити повторну обробку запитів.

Використання клієнт-серверної архітектури з модульним принципом побудови є доцільним рішенням для системи перевірки плагіату. Такий підхід забезпечує зручну взаємодію між компонентами системи, дозволяє підтримувати стабільну продуктивність і водночас створює можливість для подальшого розширення функціональності без необхідності суттєво змінювати вже існуючу структуру програмного забезпечення.

2.3. Моделювання системи (UML-діаграми: варіантів використання, класів, послідовностей)

Перед тим як переходити до програмної реалізації системи перевірки плагіату, важливо спочатку зрозуміти її загальну структуру та принципи роботи. У ході проєктування це дозволяє сформувати цілісне уявлення про те, як саме система буде функціонувати, як будуть взаємодіяти її окремі компоненти та які дані між ними передаватимуться. Для цього використовується UML-моделювання, яке дає змогу описати систему у вигляді стандартизованих діаграм різного рівня деталізації – від загальних сценаріїв використання до внутрішньої структури класів і послідовності виконання процесів. У даній роботі система реалізується з використанням мови програмування Python, оскільки вона зручна для роботи з текстовими даними та алгоритмами аналізу, а для зберігання інформації використовується реляційна база даних SQL, що забезпечує структуроване збереження документів і результатів перевірок.

Рисунок 2.3 – Діаграма варіантів використання системи перевірки академічного плагіату

На рисунку 2.3 наведено діаграму варіантів використання системи перевірки академічного плагіату, яка дозволяє описати загальну логіку взаємодії користувачів із системою без заглиблення в технічні деталі реалізації. Вона показує, які дії доступні користувачам і як система на них реагує, формуючи базове розуміння функціональності ще до розгляду внутрішньої архітектури. У межах моделі виділяються два основні типи взаємодії: робота звичайного користувача та адміністратора. Звичайний користувач взаємодіє із системою через завантаження документів, запуск перевірки та отримання результатів у вигляді звіту, який містить рівень унікальності та інформацію про знайдені текстові збіги. Окрім цього, у системі передбачено збереження історії перевірок, що дозволяє повертатися до попередніх результатів, аналізувати різні версії одного документа та відстежувати зміни рівня унікальності після редагування тексту, що є особливо корисним у процесі доопрацювання навчальних або наукових робіт. Інший сценарій роботи пов’язаний з адміністратором, який має доступ до функцій керування системою, зокрема до налаштування параметрів алгоритмів перевірки, управління обліковими записами користувачів та перегляду статистики використання системи, наприклад кількості перевірених документів або загального навантаження. Такий поділ ролей дозволяє чітко

розмежувати звичайне використання системи та її адміністративне керування, що позитивно впливає як на безпеку, так і на зручність експлуатації.

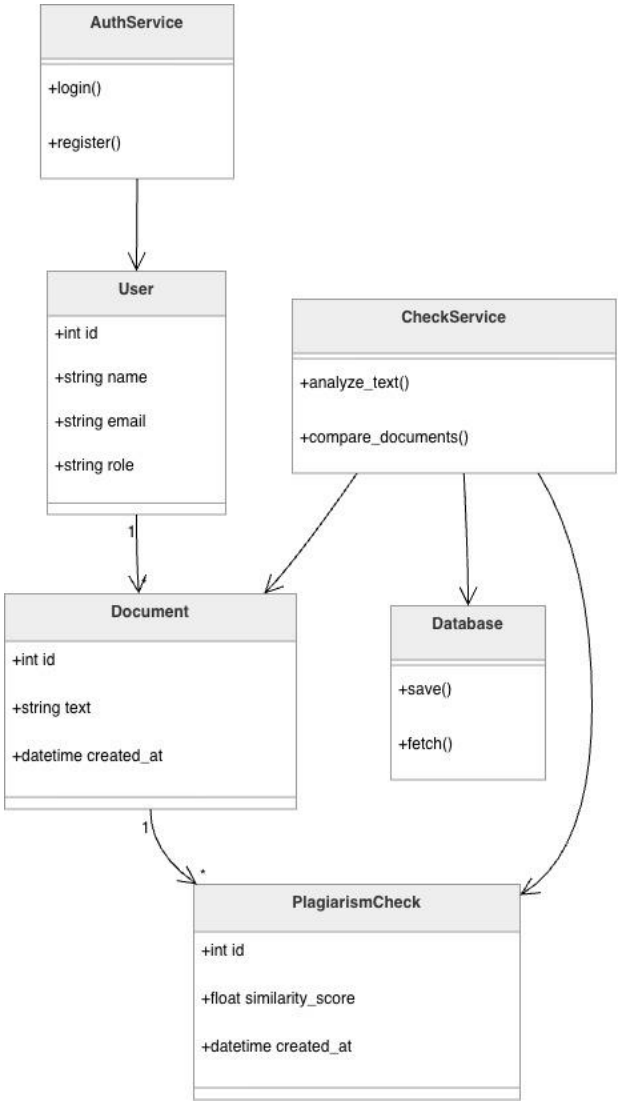


Рисунок 2.4 – Діаграма класів системи перевірки академічного плагіату

На рисунку 2.4 представлено діаграму класів системи перевірки академічного плагіату, яка відображає внутрішню структуру програмної реалізації та взаємозв’язки між основними елементами системи. Вона показує, як організовані дані, які сутності використовуються та як між ними вибудувана логіка взаємодії. У межах моделі можна виділити основні сутності предметної області, до яких належать користувач, документ і результат перевірки. Користувацькі дані використовуються для авторизації в системі та містять інформацію, необхідну для входу, електронну пошту та роль доступу, яка

визначає набір дозволених дій. Документ містить основний текст, назву файлу, дату завантаження та прив’язку до користувача, який його додав у систему, що дозволяє відстежувати походження кожного завантаженого матеріалу. Результат перевірки, у свою чергу, зберігає показник унікальності, перелік знайдених збігів і дату проведення аналізу, що дає можливість формувати історію перевірок і аналізувати зміни результатів у часі.

Окрім основних сутностей, у системі передбачено окремі сервісні компоненти, які відповідають за реалізацію логіки обробки даних. Сервіс авторизації забезпечує перевірку користувачів і контроль доступу до функцій системи, тоді як сервіс аналізу тексту виконує операції, пов’язані з обробкою документів: очищення тексту, токенізацію, нормалізацію та подальше порівняння з іншими документами в базі. У даній роботі виділено шар роботи з базою даних, який відповідає за збереження, оновлення та отримання інформації, що дозволяє відокремити бізнес-логіку від рівня зберігання даних і зробити систему більш гнучкою до змін у майбутньому. Логіка зв’язків між сутностями побудована так, що один користувач може додавати кілька документів, а кожен документ може проходити не одну перевірку, що дає змогу зберігати повну історію аналізу та відстежувати зміни результатів з часом.

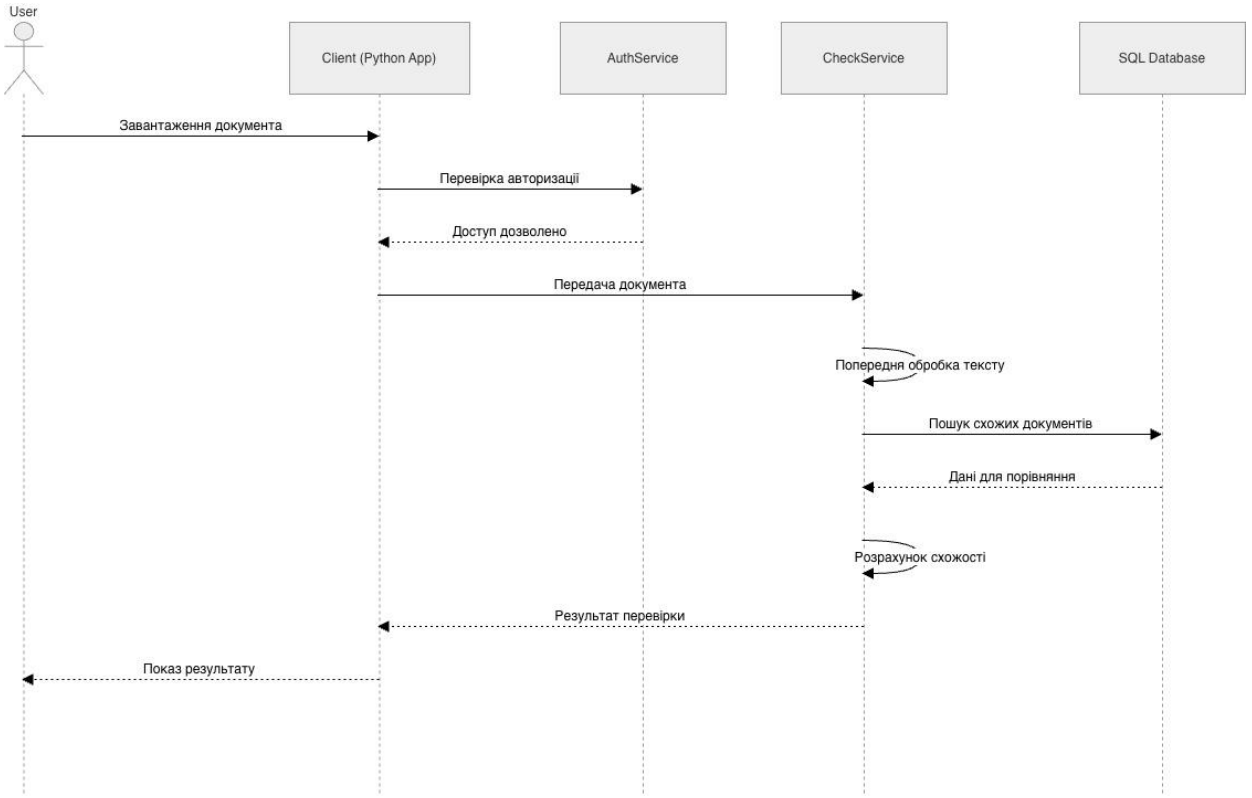


Рисунок 2.5 – Діаграма послідовностей процесу перевірки документа

На рисунку 2.5 наведено діаграму послідовностей процесу перевірки документа, яка дозволяє побачити динаміку роботи системи та порядок взаємодії між її компонентами під час виконання основного сценарію. Вона показує, як саме передаються дані між користувачем, клієнтською частиною, сервером і базою даних у процесі перевірки. Робота системи починається з авторизації користувача, після чого перевіряються облікові дані і у разі успішного входу відкривається доступ до функціоналу системи. Далі користувач завантажує документ через інтерфейс, і цей файл передається на сервер для подальшої обробки. На серверному рівні спочатку виконується попередня обробка тексту, яка включає очищення від зайвих символів, приведення тексту до єдиного формату, токенізацію та нормалізацію слів, що дозволяє підготувати дані до подальшого аналізу. Після цього система звертається до бази даних SQL для отримання раніше збережених документів, які використовуються як основа для порівняння, після чого виконується аналіз подібності текстів і розрахунок рівня збігу. Отриманий результат повертається на клієнтську частину системи, де відображається користувачу у вигляді звіту, а також одночасно зберігається в базі

даних для подальшого використання, що дозволяє формувати історію перевірок і проводити додатковий аналіз у майбутньому.

У цілому UML-моделювання дозволяє сформувати цілісне уявлення про систему ще до початку її реалізації, оскільки показує як загальну логіку роботи, так і внутрішню структуру компонентів та порядок їх взаємодії. Завдяки цьому стає можливим більш чітко розділити систему на модулі, які можна розвивати незалежно один від одного, а також спростити подальше масштабування та інтеграцію нових алгоритмів аналізу або зовнішніх сервісів без необхідності повної перебудови архітектури.

2.4. Проєктування бази даних для зберігання текстів та результатів перевірки

База даних у системі перевірки плагіату є одним із ключових компонентів, оскільки саме вона відповідає за збереження всієї інформації, з якою працює система. У ній зберігаються користувацькі облікові записи, завантажені документи, результати перевірок, а також деталі знайдених збігів. Від того, наскільки правильно спроєктована структура бази даних, залежить не лише коректність зберігання інформації, а й загальна продуктивність системи при збільшенні кількості документів і запитів. У даній роботі використовується реляційна модель даних на основі SQL, оскільки вона дозволяє чітко організувати зв’язки між сутностями, забезпечує цілісність даних і підтримує транзакційність, що особливо важливо при одночасній роботі багатьох користувачів. Окрему увагу під час проєктування приділено збереженню історії перевірок, оскільки один і той самий документ може змінюватися та перевірятися кілька разів, і система повинна зберігати результати кожного такого аналізу окремо, щоб можна було відстежувати динаміку змін рівня унікальності.

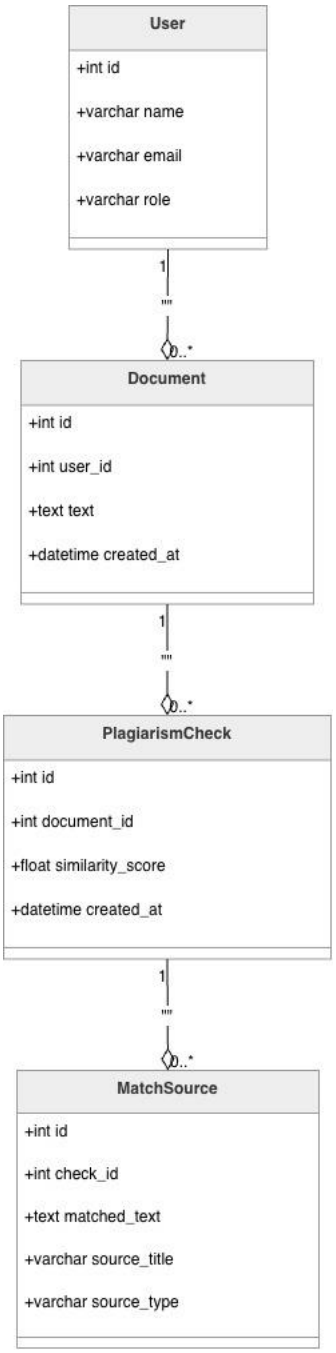


Рисунок 2.6 – ER-діаграма бази даних системи перевірки плагіату

На рисунку 2.6 наведено ER-діаграму бази даних системи перевірки плагіату, яка відображає основні сутності та зв’язки між ними. У структурі бази даних виділяється сутність користувача, яка містить базову інформацію про обліковий запис, зокрема електронну пошту, зашифрований пароль та роль доступу, що визначає можливості роботи в системі. Один користувач може створювати та зберігати кілька документів, тому між сутностями користувача і документа реалізовано зв’язок один-до-багатьох. Сутність документа містить

безпосередньо текстові дані, назву файлу та інформацію про час завантаження, а також зв’язується з користувачем через зовнішній ключ, який дозволяє однозначно визначити автора кожного документа. Наголошується, що сутність результату перевірки, яка зберігає значення рівня схожості, дату виконання аналізу та додаткові дані, що описують результат роботи алгоритму. Один документ може мати декілька результатів перевірки, оскільки він може редагуватися і повторно перевірятися, що дозволяє формувати повну історію змін. Для збереження деталей знайдених збігів використовується окрема сутність, яка містить інформацію про фрагменти тексту, що були визначені як потенційно запозичені, а також дані про їх походження; вона пов’язується з конкретним результатом перевірки через зовнішній ключ, що дозволяє деталізувати кожен випадок виявленого плагіату і не обмежуватися лише загальним відсотком схожості.

Листинг 2.1 – SQL-скрипт створення бази даних

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role VARCHAR(20) DEFAULT 'student',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    filename VARCHAR(255),
    original_text TEXT,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE results (
    id SERIAL PRIMARY KEY,
    document_id INTEGER REFERENCES documents(id) ON DELETE CASCADE,
    similarity_score FLOAT,
    matched_fragments TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Фізична реалізація бази даних представлена у вигляді SQL-таблиць, які відповідають логічній моделі. Таблиця користувачів містить основні дані облікових записів, при цьому поле електронної пошти є унікальним, що запобігає

дублюванню користувачів у системі, а пароль зберігається у вигляді хешу, що відповідає базовим вимогам безпеки. Таблиця документів містить текстові дані та інформацію про завантаження, а також зовнішній ключ, який пов’язує документ із користувачем; при цьому використання каскадного видалення дозволяє автоматично видаляти всі документи користувача у випадку його видалення з системи, що запобігає накопиченню некоректних записів. Таблиця результатів перевірки зберігає числовий показник подібності та інформацію про знайдені збіги, а також пов’язана з документом через зовнішній ключ, що дозволяє у будь-який момент відновити повний контекст виконаної перевірки.

Зі збільшенням обсягу даних особливого значення набуває питання продуктивності, оскільки без індексування таблиць система змушена виконувати повний перегляд даних при кожному запиті, що значно уповільнює роботу. Саме тому в базі даних передбачено використання індексів на зовнішніх ключах та полях, які часто використовуються для пошуку і фільтрації. Окрім цього, для текстових даних може застосовуватися повнотекстовий пошук, що дозволяє швидше знаходити необхідні фрагменти документів. Також доцільним є збереження попередньо обчислених представлень текстів, наприклад у вигляді векторів, що дозволяє уникати повторних обчислень під час кожної перевірки. Завдяки такій структурі бази даних система зберігає стабільну роботу навіть при значному зростанні кількості документів і запитів, а також залишається достатньо гнучкою для подальшого розширення функціональності.

2.5. Вибір технологій та інструментів розробки

Вибір технологічного стеку для системи перевірки академічного плагіату є одним із найважливіших етапів проєктування, оскільки саме від нього залежить подальша стабільність роботи системи, можливість її масштабування та складність підтримки в майбутньому. Неправильно обрані технології можуть призвести до проблем із продуктивністю, труднощів при розширенні функціоналу або навіть необхідності часткового переписування системи через певний час. Саме тому під час вибору інструментів враховувалися не лише поточні потреби проєкту, а й можливість подальшого розвитку системи,

збільшення кількості користувачів та обсягів текстових даних, які будуть аналізуватися. Оскільки система працює з текстами та алгоритмами аналізу природної мови, основною мовою програмування було обрано Python, який сьогодні широко використовується для створення серверних застосунків і систем обробки текстових даних. Однією з головних причин такого вибору стала велика кількість готових бібліотек для NLP та машинного навчання, що значно спрощує реалізацію алгоритмів перевірки плагіату. Крім цього, Python має зрозумілий синтаксис і високу читабельність коду, що особливо важливо для навчальних та дослідницьких проєктів, де система може поступово розширюватися та доповнюватися новими функціями.

Для реалізації серверної частини використовується FastAPI – сучасний Python-фреймворк, орієнтований на створення швидких вебзастосунків та API. Його використання дозволяє ефективно працювати з асинхронними запитами та забезпечує високу продуктивність навіть при одночасній роботі великої кількості користувачів. Ще однією перевагою FastAPI є автоматичне формування документації API, що спрощує тестування та подальшу інтеграцію системи з іншими сервісами. У порівнянні з Flask FastAPI краще підходить для навантажених систем, де важлива швидкість обробки паралельних запитів і зручна робота з типами даних. Для перевірки коректності отриманих даних використовується бібліотека Pydantic, яка дозволяє автоматично виконувати валідацію та уникати багатьох помилок під час роботи із запитами користувачів.

Архітектура системи побудована за багаторівневим принципом, що дозволяє логічно розділити різні частини застосунку. Перший рівень відповідає за взаємодію з користувачем через вебінтерфейс і реалізується за допомогою HTML, CSS та JavaScript. Через цей інтерфейс користувач завантажує документи, запускає перевірку та переглядає результати аналізу. Другий рівень складається з бізнес-логіки, реалізованої на FastAPI, де відбувається обробка HTTP-запитів, маршрутизація та взаємодія між основними модулями системи. Саме тут запускаються алгоритми аналізу тексту, формується результат перевірки та виконується взаємодія з базою даних. Третій рівень відповідає за доступ до даних

і включає роботу з SQL-запитами або ORM, що дозволяє відокремити логіку обробки від механізмів зберігання інформації. Такий підхід робить систему більш гнучкою та полегшує її подальше масштабування.

Серверна частина організована за модульним принципом, завдяки чому окремі компоненти можуть розвиватися незалежно один від одного. Наприклад, модуль авторизації відповідає за перевірку користувачів і керування доступом до функцій системи, модуль обробки документів займається завантаженням та підготовкою тексту, а модуль аналізу схожості виконує безпосереднє порівняння документів. Такий поділ дозволяє змінювати або вдосконалювати окремі частини системи без необхідності повністю перебудовувати всю архітектуру. Це особливо важливо для систем, які в майбутньому можуть доповнюватися новими алгоритмами або інтеграціями із зовнішніми платформами.

Для зберігання інформації обрано PostgreSQL – реляційну систему керування базами даних з відкритим кодом, яка добре підходить для роботи з великими обсягами структурованих даних. PostgreSQL підтримує складні SQL-запити, механізми індексування та повнотекстовий пошук, що є важливим для систем аналізу тексту. Крім цього, вона забезпечує надійну роботу з транзакціями та дозволяє зберігати цілісність даних навіть при одночасній роботі багатьох користувачів. Для адміністрування бази даних використовується pgAdmin, який надає зручний графічний інтерфейс для створення таблиць, виконання SQL-запитів та перегляду структури бази. Під час розробки програмного коду застосовуються середовища PyCharm або Visual Studio Code, які підтримують Python, мають вбудовані інструменти налагодження та інтеграцію з Git для контролю версій проєкту.

Оскільки система в перспективі може працювати з великою кількістю документів і складними алгоритмами семантичного аналізу, доцільно передбачити можливість використання векторних представлень текстів. Для цього можуть використовуватися спеціалізовані рішення на кшталт pgvector або Chroma, які дозволяють ефективно зберігати embeddings і виконувати швидкий семантичний пошук. Це особливо важливо для виявлення прихованого плагіату,

коли тексти можуть бути суттєво перефразовані, але залишатися схожими за змістом.

Для реалізації алгоритмічної частини системи використовується кілька спеціалізованих бібліотек. Бібліотека scikit-learn застосовується для TF-IDF-векторизації та обчислення косинусної подібності між текстами. Для попередньої обробки тексту використовуються spaCy або NLTK, які дозволяють виконувати токенізацію, лематизацію та видалення стоп-слів. Отримання семантичних embeddings реалізується за допомогою бібліотеки sentence-transformers, яка працює на основі трансформерних моделей. Для зчитування документів форматів DOCX і PDF використовуються python-docx та PyPDF2 відповідно, а формування PDF-звітів виконується за допомогою бібліотеки ReportLab.

Вибір технологічного стеку дозволяє створити систему, яка поєднує гнучкість, продуктивність та можливість подальшого розвитку. Поєднання Python, FastAPI та PostgreSQL забезпечує стабільну основу для реалізації алгоритмів перевірки плагіату, а модульна архітектура спрощує подальше масштабування системи, інтеграцію нових технологій та вдосконалення методів аналізу тексту без необхідності кардинальної перебудови всієї структури застосунку.

2.6. Висновки до розділу 2

У другому розділі виконано проєктування системи перевірки академічного плагіату, починаючи з формування вимог і закінчуючи вибором технологій та загальних принципів реалізації. На цьому етапі важливо було не просто описати набір функцій, які повинна підтримувати система, а й зрозуміти, як вона буде поводитися в умовах реального використання, коли зростає кількість користувачів, документів і одночасних перевірок. Індивідуально досліджувалось питання того, як зробити структуру системи такою, щоб її можна було розширювати без суттєвих змін уже реалізованих компонентів.

Під час визначення вимог система була розділена на функціональні та нефункціональні складові. До функціональних віднесено роботу з різними форматами документів, виконання перевірки текстів на наявність як прямих збігів,

так і змінених або перефразованих фрагментів, а також формування результатів аналізу у зручному для користувача вигляді. Нефункціональні вимоги стосувалися швидкості обробки даних, можливості масштабування, стабільності роботи при навантаженні та загальної зручності взаємодії з системою. При розгляді сценаріїв використання стало очевидно, що система може застосовуватися як для перевірки окремих робіт, так і для обробки великих масивів документів, наприклад у межах навчального закладу, де перевірка відбувається регулярно і у великих обсягах.

Архітектурне проєктування було виконано з урахуванням поділу системи на окремі логічні рівні. Було обрано клієнт-серверну модель, у якій інтерфейс користувача, бізнес-логіка та робота з даними розділені між собою. Такий підхід дозволяє уникнути жорсткої залежності між компонентами та спрощує подальші зміни в окремих частинах системи без необхідності переробляти її повністю. Для організації взаємодії між частинами системи використано підхід REST API, що дає змогу стандартизувати обмін даними та потенційно інтегрувати систему з іншими сервісами. Питання безпеки, навантаження та масштабування також розглядалися на цьому етапі, оскільки вони безпосередньо впливають на практичну придатність системи.

Для опису структури та логіки роботи системи використовувалися UML-діаграми, які дозволили відобразити взаємодію користувачів із системою, основні компоненти та послідовність виконання процесу перевірки документа. Таке моделювання дало можливість ще до початку програмної реалізації побачити загальну логіку роботи системи та уточнити окремі моменти, пов’язані з обробкою тексту і передачею даних між модулями.

Окрім того опрацьовувалося проєктування бази даних, оскільки саме вона відповідає за збереження документів, результатів перевірок і супутньої інформації. Було сформовано реляційну структуру з виділеними основними сутностями та встановленими між ними логічними зв’язками. Такий підхід дозволяє уникати дублювання даних і забезпечує коректне збереження історії перевірок у системі. Також приділялася увага тому, як система працює з

великими обсягами текстів, зокрема через індексування та оптимізацію запитів до бази.

На стадії вибору технологій було зупиненося на Python як основній мові реалізації системи, оскільки вона добре підходить для задач обробки тексту та дозволяє швидко інтегрувати необхідні бібліотеки. Для серверної частини обрано FastAPI, що забезпечує зручну організацію API та достатню продуктивність при роботі з запитами, а для збереження даних використано PostgreSQL як надійну реляційну систему керування базами даних. В тому числі сформовано перелік бібліотек, які відповідають за обробку тексту, токенізацію та оцінювання схожості між документами. Такий набір технологій дозволяє реалізувати систему без зайвих ускладнень на рівні архітектури, зберігаючи при цьому можливість її масштабування та поступового розширення функціоналу в майбутньому. У підсумку другий розділ фактично закладає повну технічну основу для подальшої практичної реалізації проєкту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Опис реалізації основних модулів системи

Система перевірки академічного плагіату побудована за класичною клієнт-серверною архітектурою, де серверна частина відповідає за обробку даних і реалізацію бізнес-логіки, а клієнтська – за відображення результатів та взаємодію з користувачем. Такий поділ забезпечує незалежність компонентів та спрощує подальшу підтримку і розширення системи.

Серверна частина реалізована на основі веб-фреймворку FastAPI. Його вибір зумовлений кількома практичними перевагами: висока продуктивність завдяки асинхронній обробці запитів, вбудована підтримка автоматичної документації API та зручна інтеграція з базами даних і бібліотеками обробки тексту. Клієнтська частина побудована на HTML, CSS та JavaScript, що дозволяє реалізувати динамічний веб-інтерфейс без перезавантаження сторінки під час роботи з результатами.

Основна взаємодія користувача із системою починається з веб-інтерфейсу, де можна завантажити текстовий документ для перевірки. Після вибору файлу він передається на сервер, де виконується його збереження та початкова обробка.

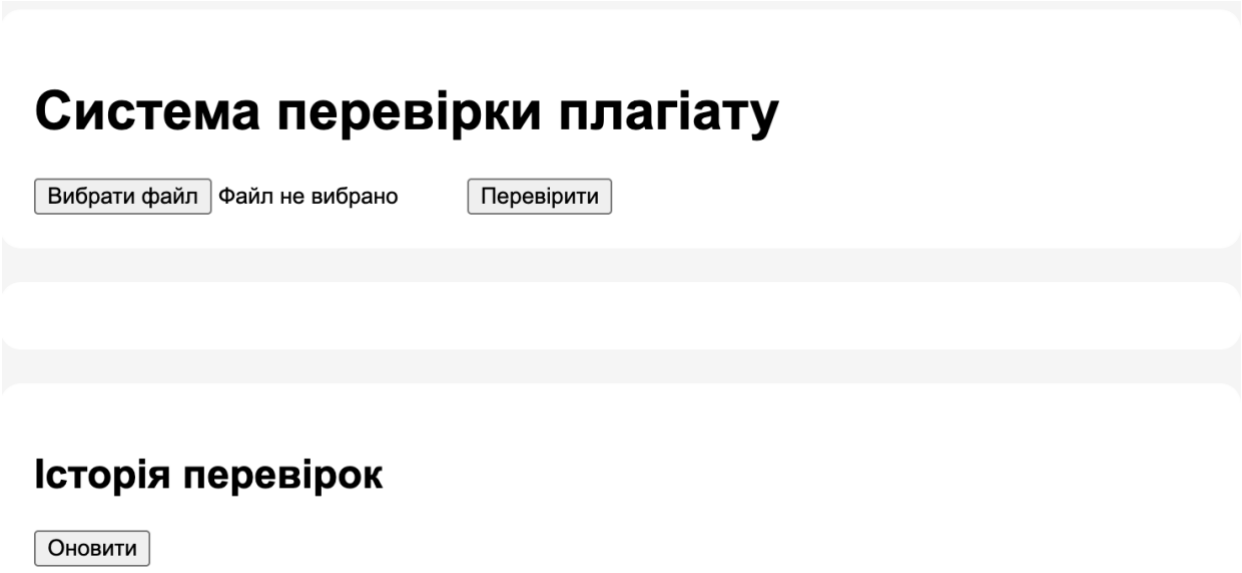


Рис. 3.1 – Головна сторінка системи перевірки плагіату (завантаження документа та запуск аналізу)

Після завершення аналізу користувач бачить результат: числовий відсоток текстової подібності та один з трьох рівнів плагіату – low, medium або high. Така градація дозволяє швидко зрозуміти загальну картину навіть без заглиблення в числові деталі.

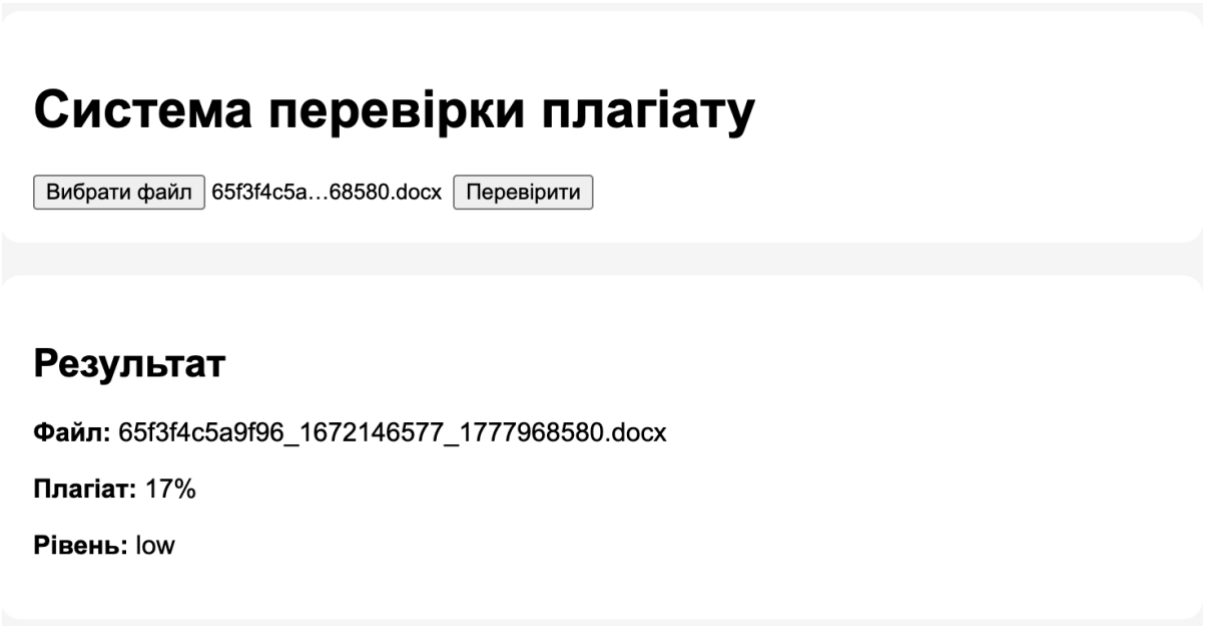


Рис. 3.2 – Відображення результату перевірки документа (відсоток подібності та рівень плагіату)

Після отримання файлу серверна частина виконує його обробку у кілька послідовних кроків. Спочатку документ зберігається на сервері, після чого з

нього витягується текстовий вміст – за допомогою відповідних парсерів залежно від формату файлу: TXT, DOCX або PDF. Такий підхід забезпечує уніфіковане представлення вхідних даних незалежно від того, в якому форматі надійшов документ.

Витягнутий текст проходить етап попередньої обробки: очищення від службових символів і форматувальних елементів, нормалізація регістру та приведення до уніфікованого вигляду. Цей крок критично важливий – без нього технічні особливості форматування могли б спотворювати результати порівняння. Наприклад, ідентичні за змістом фрагменти у різних форматах або з різним регістром без нормалізації можуть не бути розпізнані як схожі.

Центральним компонентом системи є модуль аналізу текстової подібності, який реалізує основну логіку виявлення плагіату. Він порівнює текстові ознаки нового документа з уже збереженими роботами у базі даних. Система обчислює коефіцієнт подібності між текстами, а на основі отриманого значення визначає рівень плагіату за трирівневою градацією: low – низький, medium – середній, high – високий. Такий підхід забезпечує інтуїтивну інтерпретацію результатів без необхідності розбиратися у числових метриках.

Після завершення аналізу результати зберігаються у базі даних. Кожен запис містить назву файлу, відсоток подібності, визначений рівень плагіату та оброблений текст документа. База даних у цьому випадку виконує не лише функцію збереження результатів, але й використовується як джерело для порівняння нових документів із вже наявними. Завдяки цьому стає можливим виявлення внутрішніх запозичень між роботами різних користувачів, що особливо важливо для навчальних закладів, де обсяг студентських робіт поступово накопичується.

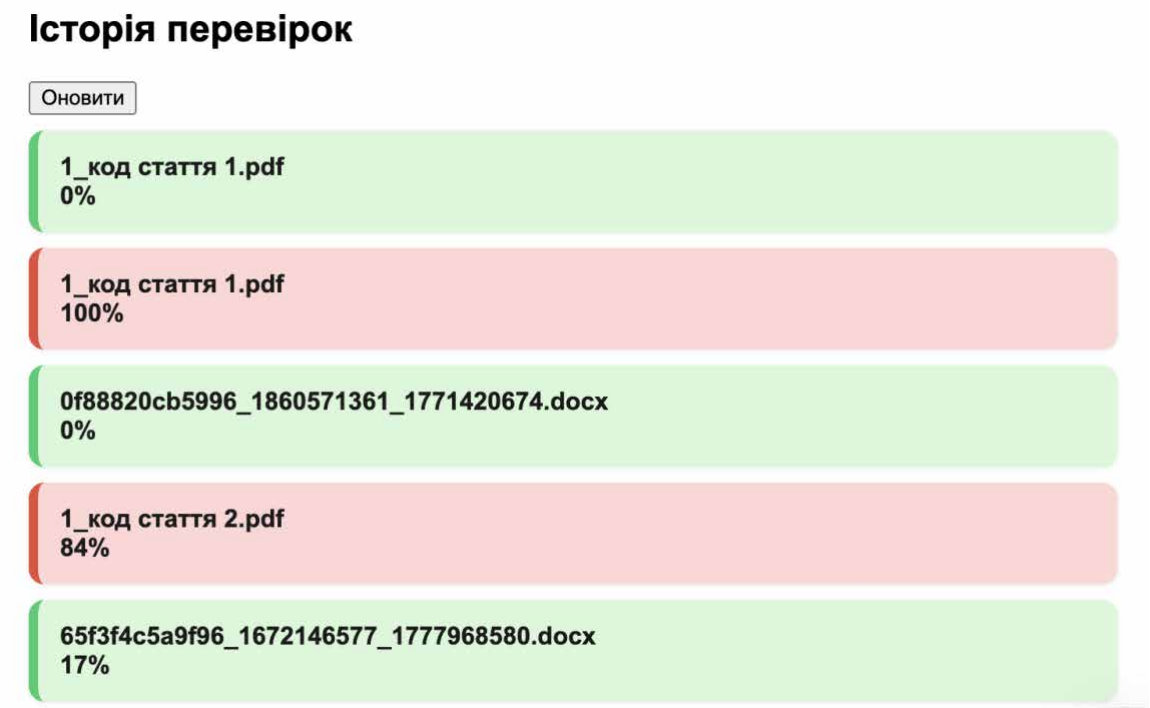
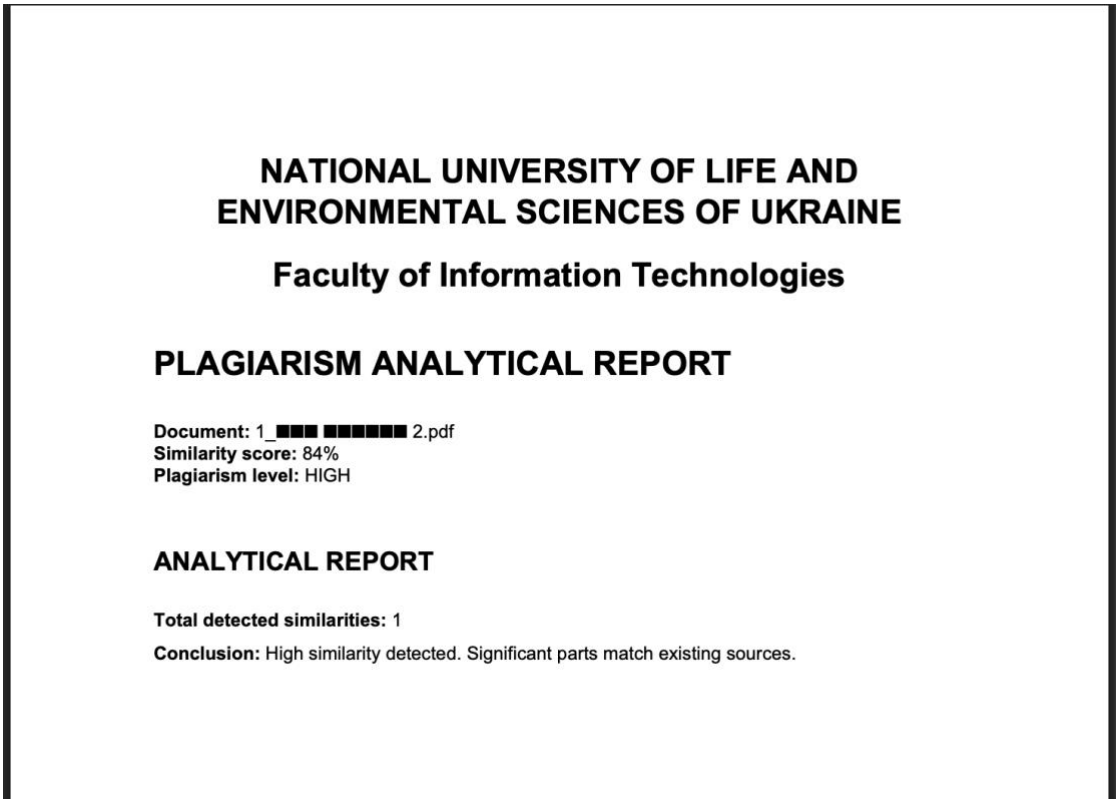


Рис. 3.3 – Історія перевірок документів із відображенням рівня плагіату

Клієнтська частина забезпечує динамічну взаємодію з користувачем: результати перевірки відображаються та оновлюються без перезавантаження сторінки завдяки асинхронним запитам на JavaScript. Для підвищення зручності сприйняття реалізовано кольорову індикацію рівня плагіату: низький рівень позначається зеленим кольором, середній – жовтим, високий – червоним. Це дозволяє оцінити результат ще до того, як читати числові значення.

Окремим функціональним елементом є можливість генерації PDF-звіту за результатами перевірки. Для кожного проаналізованого документа може бути автоматично сформований звіт із відсотком подібності, визначеним рівнем плагіату та іншими параметрами аналізу. Формування звіту реалізовано за допомогою бібліотеки ReportLab, що дозволяє створювати структуровані PDF-документи без звернення до зовнішніх сервісів. Такий підхід забезпечує автономність системи і дає можливість використовувати результати перевірки як офіційний документальний матеріал – наприклад, для долучення до справи студента або архіву кафедри.



NATIONAL UNIVERSITY OF LIFE AND ENVIRONMENTAL SCIENCES OF UKRAINE

Faculty of Information Technologies

PLAGIARISM ANALYTICAL REPORT

Document: 1_■■■ ■■■■■■ 2.pdf
Similarity score: 84%
Plagiarism level: HIGH

ANALYTICAL REPORT

Total detected similarities: 1

Conclusion: High similarity detected. Significant parts match existing sources.

Рис. 3.4 – Автоматично згенерований PDF-звіт результатів перевірки

Архітектура системи побудована за принципом слабкого зв'язування компонентів, що забезпечує незалежність модулів і спрощує їх подальшу модернізацію. Серверна частина відповідає за обробку даних та реалізацію алгоритмів аналізу, тоді як клієнтська – лише за відображення результатів та взаємодію з користувачем. Такий поділ дозволяє, наприклад, замінити або вдосконалити алгоритм аналізу без будь-яких змін у інтерфейсі.

Реалізована система є повноцінним веб-рішенням, яке поєднує алгоритмічний аналіз текстової подібності, роботу з базою даних та інтерактивний інтерфейс користувача. Додатково можливість формування PDF-звітів підвищує практичну цінність системи, оскільки дозволяє зберігати результати перевірки у зручному для подальшого використання та документального підтвердження вигляді.

3.2. Розробка алгоритму перевірки плагіату

Алгоритм перевірки плагіату – це той компонент системи, від якого залежить усе інше: точність виявлення запозичень, швидкість обробки документів і якість фінального результату. Його реалізовано як багаторівневу

обчислювальну процедуру, що послідовно застосовує методи попередньої обробки тексту, структурного аналізу, статистичного порівняння та семантичної оцінки подібності.

Загальна логіка алгоритму побудована за принципом послідовного уточнення: від грубого відбору потенційно схожих документів до більш точного визначення ступеня подібності. Це дозволяє зменшити обчислювальне навантаження та забезпечити баланс між швидкістю і точністю аналізу – адже детальна семантична перевірка кожного документа проти всієї бази даних була б надто ресурсомісткою. Загальна схема алгоритму перевірки плагіату наведена у Додатку А.

Першим етапом є прийом вхідного документа та його первинна обробка. Система зчитує файл незалежно від його формату і перетворює вміст у текстовий вигляд. Далі виконується очищення тексту від службових символів, форматувальних елементів і шумових даних, що не несуть смислового навантаження. Нормалізація регістру забезпечує уніфіковане представлення тексту – без неї одне й те саме слово, написане по-різному, могло б не розпізнаватися як ідентичне. Етапи попередньої обробки тексту детально наведені у Додатку Б.

Наступний крок – токенізація: розбиття тексту на окремі одиниці (слова або речення), які далі використовуються для аналізу. За необхідності виконується також видалення стоп-слів – частотних слів (прийменників, сполучників, артиклів тощо), які не несуть самостійного смислу і лише збільшують розмірність даних без реального внеску у порівняння. Структурне представлення тексту та метод shingling описані у Додатку В.

Після попередньої обробки новий документ порівнюється з уже наявними у базі даних. На цьому етапі за допомогою індексованих структур зберігання виконується швидкий відбір потенційно схожих документів – кандидатів для детальнішого аналізу. Це ключовий крок для підтримки прийнятної швидкодії при великій кількості збережених документів.

Основним інструментом кількісного порівняння є метод TF-IDF у поєднанні з косинусною мірою подібності (cosine similarity). TF-IDF дозволяє зважити важливість кожного слова в контексті конкретного документа відносно всієї колекції – завдяки цьому часто вживані, але малоінформативні слова мають менший вплив на результат. Косинусна міра, у свою чергу, обчислює кут між векторними представленнями текстів: чим менший кут – тим більша подібність. Реалізація алгоритму обчислення подібності (TF-IDF та cosine similarity) наведена у Додатку Г.

Структурні методи добре справляються з прямими текстовими збігами, але погано виявляють плагіат у випадках перефразування. Щоб подолати цю обмеженість, алгоритм доповнено методами обробки природної мови (NLP): тексти перетворюються у векторні представлення (embeddings), після чого обчислюється їхня семантична близькість. Такий підхід дозволяє виявляти схожість навіть там, де лексика майже повністю змінена, але зміст залишився тим самим.

Поєднання структурного і семантичного аналізу дає синергетичний ефект: перший швидко і точно знаходить прямі запозичення, другий – вловлює приховані. Разом вони перекривають більшість сценаріїв академічного плагіату.

Фінальним етапом алгоритму є агрегація результатів. Отримані значення структурної та семантичної подібності комбінуються у єдиний показник унікальності документа. На основі цього значення система визначає рівень плагіату (low, medium, high) та формує підсумковий результат перевірки.

Вибір порогових значень для класифікації є окремим методичним питанням. Занадто низький поріг для high-рівня призведе до хибнопозитивних спрацювань, занадто високий – до пропуску реальних запозичень. У поточній реалізації порогові значення підібрані емпірично на основі тестових наборів документів і можуть бути скоректовані залежно від специфіки використання системи. Отримані результати передаються до модуля формування звітів та зберігаються у базі даних, що дозволяє використовувати їх для подальшого аналізу та формування історії перевірок.

Розроблений алгоритм можна розглядати як багаторівневий підхід до обробки тексту, який поєднує структурний аналіз, статистичне порівняння та елементи семантичної оцінки. Таке поєднання дозволяє підвищити точність виявлення текстових запозичень, водночас зберігаючи ефективність роботи при обробці великих обсягів даних.

3.3. Тестування програмного забезпечення (функціональне, навантажувальне)

Тестування є обов’язковим етапом розробки програмного забезпечення, оскільки саме воно дозволяє перевірити, чи відповідає система заявленій логіці роботи. У випадку системи перевірки плагіату це має особливе значення, адже будь-які помилки в алгоритмах або взаємодії між модулями можуть призводити до некоректних результатів аналізу, що в підсумку знижує довіру до системи. Оскільки розроблена система реалізована у вигляді веб-застосунку, процес тестування проводився на декількох рівнях і охоплював як роботу API, так і взаємодію між клієнтською та серверною частинами, а також коректність виконання алгоритмів аналізу тексту і загальну поведінку системи при різних умовах використання.

Функціональне тестування було зосереджене на основних сценаріях роботи користувача, таких як завантаження документів, запуск перевірки на плагіат, отримання та перегляд результатів аналізу, а також доступ до історії перевірок. Окремо перевірявся модуль автентифікації, який відповідає за розмежування прав доступу залежно від ролі користувача в системі. Під час тестування завантаження файлів було підтверджено коректну обробку документів усіх підтримуваних форматів, а саме TXT, DOCX і PDF, при цьому система стабільно витягує текстовий вміст незалежно від внутрішньої структури файлу та виконує його подальшу попередню обробку перед аналізом.

Окрему увагу приділено перевірці модуля аналізу текстової подібності, для чого використовувалися контрольні тексти з заздалегідь відомими фрагментами запозичень. Це дозволило більш об’єктивно оцінити якість роботи алгоритму. Отримані результати показали, що система здатна виявляти як прямі збіги тексту,

так і частково перефразовані фрагменти, що свідчить про ефективність використаного комбінованого підходу. Також перевірявся модуль формування результатів, де система коректно розраховує відсоток подібності, визначає рівень плагіату у форматі low, medium або high та зберігає всі дані в базі для подальшого використання в історії перевірок.

Навантажувальне тестування виконувалося з метою оцінки стабільності системи в умовах одночасної роботи кількох користувачів. Для цього моделювалися сценарії паралельного завантаження документів, що дозволило перевірити, як серверна частина поводиться при підвищеному навантаженні. Додатково аналізувалася обробка великих за обсягом текстових файлів, оскільки саме розмір документа суттєво впливає на час виконання алгоритмів аналізу. Отримані результати тестування наведено в таблиці 3.1.

Таблиця 3.1 – Результати тестування програмного забезпечення

№	**Тип тестування**	**Опис сценарію**	**Очікуваний результат**	**Фактичний результат**	**Результат**
1	Функціональне	Завантаження TXT/DOCX/PDF файлів	Коректне зчитування тексту	Відповідає	Пройдено
2	Функціональне	Перевірка автентифікації	Розмежування доступу за ролями	Відповідає	Пройдено
3	Функціональне	Обчислення рівня плагіату	Визначення low/medium/high	Відповідає	Пройдено
4	Функціональне	Збереження результатів у БД	Коректне збереження даних	Відповідає	Пройдено
5	Навантажувальне	Одночасна перевірка декількох документів	Стабільна робота системи	Відповідає	Пройдено
6	Навантажувальне	Обробка великих текстових файлів	Коректний результат без помилок	Відповідає	Пройдено

Таблиця 3.2 – Тест-кейси системи перевірки плагіату (plagiarism-system.onrender.com)

ID	Назва тесту	Формат файлу	Кроки виконання	Очікуваний результат
TC-01	Завантаження .docx	.docx	POST /api/v1/documents/check-plagiarism з валідним .docx-файлом	HTTP 200, JSON id, filename max_similarity, level, matches
TC-02	Завантаження .pdf	.pdf	POST /api/v1/documents/check-plagiarism з валідним .pdf-файлом	HTTP 200, текст витягнуто, max_similarity розраховано
TC-03	Виявлення 100% плагіату	.docx / .pdf	Завантажити файл, ідентичний документу з БД;POST /check-plagiarism	max_similarity=1.0 level="red", matches зі збігами
TC-04	Заміна символів (омограф)	.docx	Завантажити текст з кирилиць-лат: 'с'→'c', 'а'→'a', 'е'→'e'; POST /check-plagiarism	Система нормалізує текст схожість визначена коректно
TC-05	Непідтр. формат (.xlsx)	.xlsx	Відправити .xlsx-файл до POST /check-plagiarism	HTTP 422 повідомлення про непідтр. формат

Результати тестування підтверджують коректність роботи системи, її відповідність функціональним вимогам та здатність стабільно функціонувати в умовах реального навантаження. Жоден із протестованих сценаріїв не виявив критичних помилок чи некоректної поведінки.

3.4. Аналіз результатів роботи системи

Після завершення тестування постає питання – наскільки система справляється зі своїм основним завданням? Тестування підтверджує, що модулі працюють коректно, але аналіз результатів дозволяє оцінити глибше: наскільки точним є алгоритм, де є обмеження і як система поводиться в нетипових сценаріях.

Основним критерієм оцінювання є точність визначення рівня текстової подібності між документами. Проведений аналіз підтвердив, що комбінований підхід – поєднання структурних методів (shingling та fingerprinting) із семантичним аналізом на основі NLP – дає стабільніші та точніші результати порівняно з будь-яким із цих методів окремо.

Структурні методи демонструють високу ефективність при виявленні прямих текстових збігів. Семантичний аналіз, у свою чергу, знаходить схожість навіть при перефразуванні або частковій зміні структури тексту. Разом вони перекривають більшість типових сценаріїв академічного плагіату – від грубого копіювання до замаскованих запозичень. Найстабільніші результати досягаються при багаторівневій оцінці, коли підсумковий показник унікальності формується шляхом поєднання результатів різних методів. Такий підхід зменшує вплив похибок окремих алгоритмів і підвищує загальну надійність системи. Водночас збільшення ваги семантичного аналізу покращує точність у складних випадках, але дещо знижує швидкодію – це природний компроміс, який може бути налаштований залежно від пріоритетів конкретного застосування.

Зі збільшенням розміру документа час обробки зростає, однак попередня обробка тексту, токенізація та fingerprinting суттєво зменшують навантаження на систему порівняно з наївним підходом повного попарного порівняння. Найбільш ресурсомістким етапом залишається семантичний аналіз: обчислення embeddings вимагає більше ресурсів порівняно зі структурними методами, тому при масштабуванні системи саме цей компонент потребуватиме найбільшої уваги з точки зору оптимізації. Ефективність роботи бази даних як джерела для порівняння також підтверджена: використання індексації та оптимізованих

запитів забезпечує швидкий доступ до даних навіть при значній кількості збережених записів. Це підтверджує доцільність використання PostgreSQL та правильність обраної схеми зберігання даних.

Серед іншого було оцінено ефективність формату подання результатів. Поєднання загального відсотка унікальності з трирівневою класифікацією виявилося найбільш інформативним: користувач може швидко зрозуміти загальний рівень ризику і при потребі заглибитися в числові деталі. Кольорова індикація рівнів додатково знижує когнітивне навантаження при перегляді великої кількості результатів.

Система демонструє стабільну роботу при одночасній обробці кількох запитів. Незначне збільшення часу відповіді спостерігається лише при пікових навантаженнях, що є очікуваною поведінкою для систем із інтенсивними обчислювальними операціями.

Узагальнений аналіз показує, що розроблена система в цілому є завершеною з функціональної точки зору та може застосовуватися в реальних умовах. Реалізований багаторівневий алгоритм дозволяє досягати прийнятного балансу між точністю перевірки та швидкістю обробки, а модульна структура системи робить її достатньо гнучкою для подальшого розвитку без необхідності суттєвих змін у вже реалізованій архітектурі. Отримані результати також підтверджують, що вибір Python для серверної частини та PostgreSQL для зберігання даних був обґрунтованим, оскільки ці технології забезпечили потрібний рівень продуктивності, гнучкості та можливості масштабування системи.

3.6. Висновки до розділу 3

У третьому розділі виконано практичну реалізацію системи перевірки академічного плагіату відповідно до раніше розробленої архітектури. Реалізація охоплює повний цикл роботи з текстовими документами: від їх завантаження та первинної обробки до виконання аналізу текстової подібності, формування результатів перевірки та подальшого їх збереження і відображення користувачу. Основна увага на цьому етапі приділялася тому, щоб логіка роботи системи

відповідала закладеним на стадії проєктування принципам і не потребувала суттєвих змін у структурі при розширенні функціоналу.

У процесі реалізації було створено та об’єднано основні програмні модулі системи, серед яких модуль автентифікації користувачів, модуль обробки документів, модуль аналізу текстової подібності, модуль взаємодії з базою даних і модуль формування результатів перевірки. Кожен із цих компонентів виконує свою окрему функцію, але при цьому всі вони працюють у єдиному процесі обробки документа. Такий поділ дозволив уникнути надмірної залежності між частинами системи і зробив її структуру більш зрозумілою для подальшого супроводу та розвитку.

В тому числі реалізовано алгоритмічну частину перевірки плагіату, яка є центральним елементом усієї системи. У роботі використано поєднання структурних методів аналізу тексту, зокрема shingling і fingerprinting, із підходами на основі обробки природної мови. Така комбінація дозволяє працювати не лише з прямими збігами тексту, але й із випадками, коли змінюється форма викладу або структура речень. На практиці це особливо важливо, оскільки саме перефразовані фрагменти найчастіше складно виявити за допомогою лише простих методів порівняння.

Для роботи з користувачем зроблено веб-інтерфейс, через який можна виконувати основні операції без встановлення додаткових програм. Під час перевірки результати показуються одразу в зрозумілому вигляді, зокрема через кольорове виділення фрагментів тексту, де є ознаки запозичення. Окрім того додано можливість формувати PDF-звіти, що зручно для подальшого використання в навчальному процесі або для збереження результатів перевірок.

Проведене тестування показало, що основні модулі системи працюють коректно та взаємодіють між собою без помітних збоїв. Між іншим перевірялася робота системи при різних обсягах вхідних даних, щоб оцінити її поведінку при збільшенні навантаження. Отримані результати свідчать про те, що система зберігає стабільність і передбачувану роботу навіть у більш складних умовах, а

обрані архітектурні рішення загалом дозволяють поєднати прийнятну швидкість обробки з достатнім рівнем точності аналізу текстів.

Третій розділ підтверджує практичну реалізованість розробленого підходу та демонструє, що система відповідає поставленим вимогам. Подальший розвиток може бути пов’язаний із покращенням методів семантичного аналізу, оптимізацією продуктивності при роботі з великими обсягами даних, а також розширенням можливостей системи для виявлення більш складних випадків текстових запозичень, зокрема міжмовного характеру.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В ІНФОРМАЦІЙНИХ СИСТЕМАХ

4.1. Аналіз умов праці при роботі з комп'ютерною технікою

Робота над програмною системою перевірки плагіату фактично повністю пов’язана з тривалим використанням комп’ютера, і саме це визначає специфіку умов праці розробника. На перший погляд може здаватися, що така діяльність є «комфортною», однак на практиці вона пов’язана з постійним статичним навантаженням, високою концентрацією уваги та значним навантаженням на зорову систему. Якщо не звертати увагу на організацію робочого місця і режим роботи, то поступово накопичується втома, яка впливає як на продуктивність, так і на загальний стан здоров’я.

Основними факторами, які найбільше впливають на самопочуття під час роботи за комп’ютером, є тривале сидіння в одній позі, перенапруження м’язів шиї та спини, а також зорове навантаження через постійну роботу з текстом і кодом. В тому числі варто враховувати й те, що сучасні монітори хоч і відповідають санітарним нормам, але все одно не виключають ризику розвитку синдрому «сухого ока» або загальної втоми зору при багатогодинній безперервній роботі. За таких умов навіть невеликі порушення режиму праці можуть призводити до головного болю, зниження концентрації та загального відчуття перевантаження [25].

Важливу роль у цьому контексті відіграє ергономіка робочого місця, оскільки саме вона визначає, наскільки природно людина тримає тіло під час роботи. Правильно підібрані стіл і крісло дозволяють зменшити навантаження на хребет і уникнути постійного напруження в плечовому поясі. Зазвичай рекомендується, щоб стопи повністю стояли на підлозі, спина мала опору в області попереку, а руки під час набору тексту знаходилися приблизно під прямим кутом у ліктях. Такі умови здаються дрібницею, але саме вони визначають, чи буде робота комфортною при тривалому навантаженні.

Не менш важливим є правильне розташування монітора і відстань до нього. Якщо екран розташований занадто близько або занадто високо, це одразу створює додаткове навантаження як на очі, так і на шию. У середньому комфортною вважається відстань приблизно 50–70 см, при цьому верхня частина екрана повинна бути на рівні очей або трохи нижче. Таке положення дозволяє зменшити напруження м’язів шиї та робить роботу більш стабільною з врахуванням тривалого навантаження [26].

Особливо слід згадати освітлення, оскільки воно часто недооцінюється, хоча напряму впливає на зорову втому. Якщо світло занадто різке або, навпаки, недостатнє, очі швидко перенапружуються, особливо при роботі з текстом і кодом. Найкращим варіантом вважається поєднання природного освітлення та м’якого штучного світла без прямих відблисків на екран. Разом з тим враховувати розташування ламп, щоб уникати контрасту між яскравим екраном і темним фоном навколо, оскільки це додатково навантажує зір.

Мікроклімат приміщення також має суттєве значення, навіть якщо це не завжди помітно одразу. Температура, вологість і якість повітря безпосередньо впливають на здатність концентруватися. У занадто теплому або задушливому приміщенні швидше настає втома, тоді як занадто холодні умови викликають дискомфорт і напруження м’язів. Оптимальні умови передбачають помірну температуру та нормальну вологість повітря, а також регулярне провітрювання, яке запобігає накопиченню вуглекислого газу [25].

Доцільно враховувати шумове навантаження, яке хоча і не є критичним у більшості випадків, але впливає на концентрацію. Постійні фонові звуки, розмови або технічний шум можуть поступово знижувати ефективність роботи, особливо під час вирішення складних логічних задач. У таких випадках допомагають або організаційні заходи (зонування простору), або використання засобів шумозаглушення.

Важливим елементом є режим праці та відпочинку, оскільки саме він дозволяє зменшити накопичення втоми. При тривалій роботі за комп’ютером доцільно робити регулярні перерви, щоб зняти навантаження із зору та м’язів.

Навіть короткі паузи дозволяють суттєво знизити рівень втоми, особливо якщо під час них виконувати прості вправи для шиї, спини або очей. Такий підхід дозволяє підтримувати стабільну продуктивність протягом усього робочого дня [27].

Окрім того слід звернути увагу на психофізіологічні аспекти роботи, оскільки програмування і аналіз текстових алгоритмів потребують високої концентрації. При тривалому навантаженні може виникати відчуття «перевантаження задачами», коли складно підтримувати увагу і зростає кількість дрібних помилок. У таких випадках важливо чергувати різні типи завдань і не працювати над складними алгоритмами без перерви протягом тривалого часу.

Крім цього, значну роль відіграє електробезпека, оскільки робоче місце розробника передбачає використання великої кількості електронних пристроїв. Важливо, щоб усе обладнання було справним, а кабелі та підключення відповідали технічним вимогам. Додатково рекомендується використання заземлення та пристроїв безперебійного живлення, які допомагають уникнути втрати даних і захищають обладнання від перепадів напруги. Неправильна організація кабелів або перевантаження розеток може створювати не лише незручності, але й потенційні ризики для безпеки [28].

4.2. Вимоги до організації робочого місця програміста

Коли розглядається робота програміста в контексті створення складних інформаційних систем, таких як система перевірки академічного плагіату, стає очевидно, що робоче місце – це не просто стіл із комп’ютером, а певним чином організоване середовище, яке напряму впливає і на швидкість роботи, і на її якість, і навіть на те, наскільки довго людина може працювати без перевтоми. У реальних умовах більшість завдань виконується за екраном монітора, тому будь-які дрібні незручності – від неправильної висоти стільця до погано організованого освітлення – з часом накопичуються і починають впливати на загальний стан і концентрацію.

У нормативних документах, зокрема в ДСанПіН 3.3.2.007-98, визначено основні вимоги до організації роботи з візуальними дисплейними терміналами,

які на практиці фактично задають базовий рівень безпеки під час роботи за комп’ютером [29]. Ці вимоги стосуються не тільки технічних характеристик обладнання, а й того, як організоване робоче місце загалом: де розташований монітор, яка відстань до очей, як падає світло в приміщенні, а також як розміщені документи чи додаткові пристрої на столі. Важливо, що такі правила не є суто формальними, оскільки вони сформовані на основі практичних спостережень за тим, як тривала робота за комп’ютером впливає на зір, поставу та загальне самопочуття користувача.

Окрему увагу в організації робочого місця слід приділяти режиму праці та відпочинку, оскільки навіть правильно налаштоване робоче місце не компенсує постійного безперервного навантаження. У міжгалузевих нормах часу на відпочинок при роботі з комп’ютерною технікою зазначається необхідність регулярних перерв, які дозволяють знизити накопичену втому та відновити працездатність [31]. На практиці це виглядає не як формальна вимога, а як цілком логічна необхідність, адже при тривалій роботі з кодом або великими текстовими масивами увага поступово розсіюється, а кількість помилок зростає навіть у досвідчених розробників. Тому організація робочого процесу за принципом чергування інтенсивної роботи та короткого відпочинку фактично є частиною правильно організованого робочого місця, а не окремим фактором.

Також важливим є правове та організаційне забезпечення безпеки праці. Закон України «Про охорону праці» встановлює загальні вимоги до роботодавця і працівника щодо створення безпечних умов роботи, і хоча програмування часто сприймається як відносно безпечний вид діяльності, воно все одно підпадає під дію цих норм [32]. Це означає, що робоче місце має бути організоване так, щоб зменшувати можливі ризики – від питань електробезпеки до правильного розміщення обладнання. На практиці це проявляється у використанні справної техніки, якісних кабелів, коректному підключенні пристроїв і недопущенні перевантаження електромережі, що насправді досить часто ігнорується в реальних умовах роботи.

Якщо розглядати робоче місце програміста у більш сучасному контексті розробки програмного забезпечення, то воно включає не лише фізичне середовище, а й програмну частину. Наприклад, у задачах, пов’язаних з обробкою текстів і порівнянням документів, велике значення має стабільність середовища розробки, правильна ізоляція залежностей і відтворюваність результатів. У дослідженнях, присвячених виявленню дублікатів даних, підкреслюється, що навіть незначні відмінності в середовищі виконання можуть впливати на результати порівняння великих обсягів інформації [8]. Тому робоче місце програміста сьогодні фактично включає ще й інструменти контейнеризації, віртуальні середовища та системи контролю версій, які забезпечують стабільність процесу розробки.

Узагальнюючи, вимоги до організації робочого місця програміста не можна зводити лише до ергономіки або техніки безпеки. Це більш комплексне поняття, яке включає фізичні умови роботи, режим навантаження, нормативні вимоги та навіть особливості програмного середовища. І саме поєднання цих факторів дозволяє створити умови, за яких розробник може працювати стабільно, без надмірного навантаження та з меншим ризиком помилок у довготривалій перспективі.

4.3. Інформаційна безпека та захист даних у системі

Питання інформаційної безпеки в системі перевірки академічного плагіату є досить чутливим, оскільки така система працює з великим обсягом текстових документів, які користувачі завантажують для перевірки. У цих документах можуть міститися як навчальні роботи, так і більш серйозні матеріали – наприклад, курсові, дипломні чи навіть фрагменти наукових досліджень, які ще не опубліковані. Відповідно, сама система фактично стає сховищем потенційно конфіденційної інформації, і це автоматично підвищує вимоги до її захисту на всіх рівнях.

Як показано в сучасних дослідженнях, присвячених методам виявлення академічного плагіату, особливу увагу приділяють не лише алгоритмам порівняння текстів, але й питанням безпеки зберігання та обробки даних,

оскільки великі системи перевірки працюють із централізованими базами документів і повинні гарантувати їх цілісність і недоступність для сторонніх осіб [14]. У таких умовах безпека розглядається не як додатковий модуль, а як невід'ємна частина загальної архітектури системи, яка супроводжує всі етапи роботи – від завантаження документа до формування результату перевірки.

У практичних реалізаціях подібних систем, наприклад у комерційних платформах на кшталт Turnitin, використовується багаторівневий підхід до захисту інформації, де кожен компонент системи має власні механізми контролю доступу та обробки даних [10]. Це означає, що навіть у разі потенційного порушення одного рівня захисту система не втрачає повністю контроль над даними, оскільки інші рівні продовжують виконувати функції захисту. Такий підхід є типовим для сучасних веб-сервісів, які працюють із великими масивами користувацької інформації та повинні одночасно забезпечувати і продуктивність, і безпеку.

З аспекту програмної реалізації, питання інформаційної безпеки тісно пов'язані із загальними принципами побудови програмних систем. У класичній інженерії програмного забезпечення наголошується, що безпека має бути інтегрована в систему ще на етапі проєктування, а не додаватися як додатковий шар після реалізації основної функціональності [16]. Це означає, що архітектура системи перевірки плагіату повинна одразу враховувати механізми автентифікації, авторизації, захищеної взаємодії між компонентами та контроль доступу до даних. Якщо цього не зробити на початкових етапах, у подальшому це призводить до ускладнення структури системи та появи вразливостей, які важко виправити без повної перебудови окремих модулів.

Між іншим слід враховувати й те, що сучасні підходи до розробки програмного забезпечення передбачають використання комплексних моделей життєвого циклу системи, у яких безпека розглядається як один із ключових критеріїв якості поряд із продуктивністю, масштабованістю та зручністю використання [17]. У цьому контексті інформаційна безпека включає не лише захист від зовнішніх атак, але й контроль коректності обробки даних всередині

системи, запобігання втраті інформації та забезпечення її відновлення у випадку збоїв.

У межах розроблюваної системи перевірки плагіату це означає необхідність реалізації захищеного зберігання документів, обмеження доступу до результатів перевірки залежно від ролей користувачів, а також використання механізмів захищеної передачі даних між клієнтською та серверною частинами. Важливим також є контроль цілісності інформації, оскільки навіть незначні зміни в текстах або результатах перевірки можуть впливати на підсумкову оцінку унікальності документа.

У підсумку можна сказати, що інформаційна безпека в системі перевірки академічного плагіату не є окремою функцією, а виступає як комплексний набір принципів, які пронизують усю архітектуру програмного забезпечення. Вона охоплює як технічні механізми захисту, так і загальні принципи побудови системи, і саме їх поєднання дозволяє забезпечити надійну та стабільну роботу програмного продукту в умовах реального використання.

4.4. Висновки до розділу 4

У межах четвертого розділу розглянуто питання, пов’язані з умовами безпечної та стабільної роботи над системою перевірки академічного плагіату. Основну увагу зосереджено на трьох напрямах: організація робочих умов розробника, особливості технічного середовища, у якому виконується розробка, а також питання інформаційної безпеки самої програмної системи. Такий поділ дозволяє краще зрозуміти, що на практиці надійність програмного продукту залежить не лише від коду чи архітектури, а й від зовнішніх умов, у яких цей продукт створюється і використовується.

Аналіз умов праці показав, що ефективність роботи розробника значною мірою визначається організацією робочого місця та дотриманням базових ергономічних і санітарно-гігієнічних вимог. Має значення все: від рівня освітлення і шумового навантаження до мікроклімату в приміщенні та правильного розташування обладнання. Якщо ці фактори не враховуються, з часом накопичується втома, знижується концентрація, і це безпосередньо

впливає на якість програмного коду та швидкість прийняття рішень. Серед іншого враховується режим праці та відпочинку, оскільки тривала робота за комп’ютером без перерв поступово знижує продуктивність і може призводити до перевантаження. Також важливою складовою є дотримання вимог електробезпеки, оскільки робота з комп’ютерною технікою завжди пов’язана з певними технічними ризиками, які потрібно мінімізувати як для користувача, так і для збереження даних.

Організація технічного середовища розробки розглядається як сукупність програмних і апаратних засобів, які забезпечують нормальний процес створення та підтримки системи. У практичній розробці це включає використання систем контролю версій, ізоляцію робочих середовищ, контейнеризацію та інструменти автоматизації процесів збирання і тестування. Такі рішення дозволяють уникати ситуацій, коли зміни в одному компоненті непередбачувано впливають на інші частини системи, а також роблять процес розробки більш керованим і відтворюваним. Особливо враховується взаємодія з серверною частиною системи, яка реалізується через захищені канали зв’язку, що є важливим як для стабільності роботи, так і для захисту даних.

Інформаційна безпека системи розглядається як багаторівнева структура, де кожен рівень відповідає за свій напрям захисту. Йдеться не лише про шифрування даних під час передачі, а й про механізми автентифікації користувачів, розмежування прав доступу, контроль операцій у системі та резервне копіювання інформації. Такий підхід дозволяє зменшити ризики втрати або несанкціонованого доступу до даних, що особливо важливо для систем, які працюють із навчальними та науковими матеріалами, де часто зберігається інтелектуально цінна інформація.

Узагальнюючи розглянуті аспекти, можна сказати, що створення надійної програмної системи неможливе без одночасного врахування умов, у яких працює розробник, особливостей технічного середовища та вимог до безпеки даних. Лише поєднання цих складових дає змогу забезпечити стабільну роботу системи в реальних умовах і її подальше практичне використання без суттєвих обмежень.

ВИСНОВКИ

У межах цієї роботи було створено програмну систему, яка дозволяє перевіряти академічні тексти на наявність плагіату. Під час розробки важливо було не тільки реалізувати сам функціонал, а й розібратися, як такі системи працюють у реальному середовищі, де доводиться обробляти великі обсяги документів і при цьому отримувати результат без значних затримок. У підсумку вдалося реалізувати повний процес роботи з текстом – від завантаження документа до формування фінального звіту з результатами перевірки.

На початку роботи довелося трохи глибше зануритися в саму предметну область. Було розглянуто, як сьогодні підходять до виявлення текстових запозичень, і стало зрозуміло, що проблема не обмежується простим пошуком однакових фрагментів. Насправді плагіат може виглядати по-різному: іноді це прямі копії, а іноді – перефразований текст, який складніше виявити. Через це при проєктуванні системи було важливо передбачити не один, а кілька підходів до аналізу тексту, а також приділити увагу попередній обробці даних, без якої подальше порівняння втрачає точність.

Окремо було розглянуто існуючі рішення в цій сфері. Вони загалом працюють за схожим принципом – порівнюють текстові фрагменти з великими базами даних, де вже зібрані індексовані документи. Але при цьому самі системи сильно відрізняються за функціональністю та сферою застосування. Варто відзначити, що більшість із них є закритими, тому зрозуміти їхню внутрішню логіку повністю складно, що, в принципі, і стало додатковою причиною розробляти власне рішення в межах навчального проєкту.

Коли справа дійшла до проєктування, були визначені основні вимоги до системи і продумана її структура. Логіку вирішено розділити на окремі модулі, кожен з яких відповідає за свій етап обробки тексту. Такий підхід виявився більш зручним, оскільки дозволяє уникнути “змішування” функцій і спрощує подальші зміни або розширення системи.

Не менш важливою частиною стала робота з базою даних. Саме вона зберігає всі завантажені документи та результати перевірок, тому від її організації напряму залежить стабільність усього рішення. Також одразу було враховано, що система повинна нормально працювати з різними форматами файлів і залишатися гнучкою, щоб у майбутньому можна було додавати нові можливості без повної перебудови архітектури.

Безпосередня реалізація включала створення основних компонентів: завантаження документів, їх обробку, аналіз схожості текстів і формування результатів перевірки. У процесі використано комбінований підхід до аналізу, який дозволяє враховувати як більш формальні структурні ознаки тексту, так і випадки, коли зміст подано в перефразованому вигляді. Окремо був реалізований інтерфейс користувача, який спрощує роботу із системою і приховує технічну складність внутрішніх процесів.

Після реалізації було проведено тестування на документах різного обсягу та складності. У цілому система показала стабільну роботу і коректне виконання основних функцій навіть при збільшенні навантаження. Разом із тим стало помітно, що якість результатів сильно залежить від попередньої обробки тексту. Фактично саме цей етап визначає, наскільки коректно далі буде виконуватися порівняння, тому його важливість недооцінювати не можна.

Окремим блоком розглядалися питання охорони праці та інформаційної безпеки. Оскільки система передбачає роботу з великими обсягами даних і тривале використання комп’ютера, було описано базові вимоги до організації робочого місця та умови, які допомагають зменшити навантаження на користувача. Також було враховано фактори, що впливають на комфорт і продуктивність під час тривалої роботи, а ще – основні механізми захисту інформації, зокрема контроль доступу та збереження даних.

Як підсумок, можна сказати, що мета роботи досягнута. Розроблена система дозволяє автоматизувати процес перевірки текстів на подібність і може використовуватися в освітньому середовищі для підтримки академічної

доброчесності. Отримані результати показують, що обрана архітектура є робочою і може бути основою для подальшого розвитку системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Maurer H., Kappe F., Zaka B. *Plagiarism – A Survey* // Journal of Universal Computer Science. 2006. Vol. 12, No. 8. P. 1050–1084. URL: Journal of Universal Computer Science.

2. Clough P. *Plagiarism in Natural and Programming Languages: An Overview of Current Tools and Technologies* : Research Memoranda CS-00-05. Sheffield : Department of Computer Science, University of Sheffield, 2000. 31 p. URL: Online version.

3. Pecorari D. *Teaching to Avoid Plagiarism: How to Promote Good Source Use*. Maidenhead : Open University Press, 2013. 192 p. URL: https://www.researchgate.net/publication/262526943_Teaching_to_avoid_plagiarism_How_to_promote_good_source_use_D_Pecorari_Open_University_Press_Berkshire_England_2013.

4. International Center for Academic Integrity. *Fundamental Values of Academic Integrity*. 3rd ed. Clemson : ICAI, 2021. 28 p. URL: https://www.researchgate.net/publication/354410538_THREE_EDITIONS_OF_THE_FUNDAMENTAL_VALUES_OF_ACADEMIC_INTEGRITY_AS_A_RESOURCE_FOR_EDUCATORS

5. Про освіту : Закон України від 05.09.2017 № 2145-VIII // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2145-19.

6. Manning C. D., Schütze H. Foundations of Statistical Natural Language Processing. Cambridge : MIT Press, 1999. 680 p. URL: https://icog-labs.com/wp-content/uploads/2014/07/Christopher_D._Manning_Hinrich_Sch%C3%BCtze_Foundations_Of_Statistical_Natural_Language_Processing.pdf.

7. Broder A. Z. On the Resemblance and Containment of Documents // Proceedings of Compression and Complexity of Sequences (SEQUENCES ’97). 1997. P. 21–29. URL: https://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/broder97resemblance.pdf.

8. Elmagarmid A. K., Ipeirotis P. G., Verykios V. S. Duplicate Record Detection: A Survey // IEEE Transactions on Knowledge and Data Engineering. 2007. Vol. 19, no. 1. P. 1–16. URL: https://www.cs.purdue.edu/homes/ake/pub/TKDE-0240-0605-1.pdf

9. Stein B., Meyer zu Eissen S. Intrinsic Plagiarism Detection // Proceedings of the 28th European Conference on Information Retrieval (ECIR 2006). Lecture Notes in Computer Science. 2006. Vol. 3936. P. 565–569. URL: https://www.researchgate.net/publication/221397393_Intrinsic_Plagiarism_Detection

10. Turnitin. iThenticate – Similarity Checking for Research Integrity. URL: https://www.turnitin.com/products/ithenticate.

11. Unicheck. Plagiarism Detection System: LMS Integration and Technical Capabilities. URL: https://unicheck.com/.

12. PlagScan. Online Plagiarism Detection System: Technical Overview and Reporting Features. URL: https://www.plagscan.com/en/plagiarism-analysis.

13. Potthast M., Hagen M., Beyer A., Busse M., Tippmann M., Rosso P., Stein B. Overview of the 6th International Competition on Plagiarism Detection // Working Notes for CLEF 2014 Conference (CLEF 2014). CEUR Workshop Proceedings. Vol. 1180. Sheffield, UK, 2014. P. 845–876. URL: https://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-PotthastEt2014.pdf.

14. Foltýnek T., Meuschke N., Gipp B. Academic Plagiarism Detection: A Systematic Literature Review // ACM Computing Surveys. 2019. Vol. 52, no. 6. P. 112:1–112:42. URL: https://doi.org/10.1145/3345317.

15. Vrbanec T., Meštrović A. Relevance of Similarity Measures Usage for Paraphrase Detection // *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR 2021), IC3K*. 2021. P. 129–138. URL: https://www.scitepress.org/PublishedPapers/2021/106498/.

16. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p. URL: https://www.studyhalo.com/media/resources/resources/INF3705/Textbook/Software_Engineering_-_Ian_Sommerville.pdf.

17. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 8th ed. New York : McGraw-Hill Education, 2014. 976 p. URL: https://www.researchgate.net/publication/305000760_Software_Engineering_A_Practitioner's_Approach_8th_Ed.

18. Stallings W., Brown L. Computer Security: Principles and Practice. 4th ed. Boston : Pearson, 2018. 816 p. URL: https://ebooks.karbust.me/Technology/Stallings,%20William_%20Brown,%20Lawrie%20-%20Computer%20Security_%20Principles%20and%20Practice-Pearson%20Education%20Limited%20(2018).pdf.

19. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston : Addison-Wesley / Pearson, 2012. 624 p. URL: https://ptgmedia.pearsoncmg.com/images/9780321815736/samplepages/0321815734.pdf .

20. Jurafsky D., Martin J. H. *Speech and Language Processing*. 3rd ed. Stanford : Stanford University, 2021. URL: https://web.stanford.edu/~jurafsky/slp3/

21. Tanenbaum A. S., Bos H. Modern Operating Systems. 4th ed. Hoboken : Pearson, 2015. 1136 p. URL: https://linux.harshkapadia.me/files/books/tanenbaum-modern-operating-systems-4th-edition.pdf.

22. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux Journal. 2014. Vol. 2014, no. 239. P. 2. URL: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.

23. Chacon S., Straub B. *Pro Git*. 2nd ed. New York : Apress, 2014. 456 p. URL: https://git-scm.com/book.

24. Stallings W. Cryptography and Network Security: Principles and Practice. 7th ed. Hoboken : Pearson, 2016. 768 p. URL: http://staff.ustc.edu.cn/~mfy/moderncrypto/crypto7ed.pdf.

25. Bishop M. Computer Security: Art and Science. 2nd ed. Boston : Addison-Wesley / Pearson, 2019. 1440 p. URL: https://books.google.com/books?id=pfdBiJNfWdMC.

26. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. Irvine : University of California, 2000. 162 p. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

27. PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*. 2024. URL: https://www.postgresql.org/docs/16/.

28. OWASP Foundation. *OWASP Application Security Verification Standard*. 2023. URL: https://owasp.org/www-project-application-security-verification-standard/.

29. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Київ : Міністерство охорони здоров'я України, 1998. 24 с. URL: https://zakon.rada.gov.ua/laws/show/z0419-98

30. ДСТУ EN ISO 9241-5:2015. Ергономіка взаємодії людина–система. Частина 5. Вимоги до компонування робочого місця та постури. Київ : УкрНДНЦ, 2016. 22 с. URL: https://online.budstandart.com/ua/catalog/doc-page?id_doc=63607.

31. Міжгалузеві норми часу на відпочинок при роботі з комп'ютерною технікою. Київ : Міністерство праці та соціальної політики України, 2000. 12 с. URL: https://zakon.rada.gov.ua/laws/show/z0484-00

32. Про охорону праці : Закон України від 14.10.1992 № 2694-XII // Відомості Верховної Ради України. 1992. № 49. URL: https://zakon.rada.gov.ua/laws/show/2694-12

.

Додаток А.

Загальна архітектура + API маршрути

```
from fastapi import APIRouter, UploadFile, File, Depends
from fastapi.responses import FileResponse
from sqlalchemy.orm import Session
import os
import re

from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet
```

```
    from backend.app.db.session import SessionLocal
    from backend.app.models.document import Document

    from backend.app.services.file_service import save_file
    from backend.app.services.text_extractor import extract_text

    from backend.app.services.plagiarism_service import (
        calculate_similarity,
        get_plagiarism_level,
        find_text_matches
    )

    router = APIRouter()

    # ---------------- DB ----------------
    def get_db():
        db = SessionLocal()
        try:
            yield db
        finally:
            db.close()

    # ---------------- CLEAN TEXT ----------------
    def clean_pdf_text(text: str) -> str:
        if not text:
            return ""

        text = str(text)
        text = re.sub(r"[^\w\s\.,\-\(\):;%]", " ", text)
        text = re.sub(r"\s+", " ", text)

        return text.strip()

    # ---------------- CHECK PLAGIARISM ----------------
    @router.post("/documents/check-plagiarism")
    def check_plagiarism(file: UploadFile = File(...), db: Session =
Depends(get_db)):

        file_path = save_file(file)
        new_text = extract_text(file_path)

        existing_docs = db.query(Document).all()
        existing_texts = [d.content for d in existing_docs if
d.content]

        similarities = calculate_similarity(new_text, existing_texts)

        max_score = max(similarities) if similarities else 0.0
        level = get_plagiarism_level(max_score)

        matches = find_text_matches(new_text, existing_texts)
```

```
        if not isinstance(matches, list):
            matches = []

        doc = Document(
            filename=file.filename,
            file_path=file_path,
            content=new_text,
            max_similarity=max_score,
            level=level,
            matches=matches
        )

        db.add(doc)
        db.commit()
        db.refresh(doc)

        return {
            "id": doc.id,
            "filename": file.filename,
            "max_similarity": max_score,
            "level": level,
            "matches": matches
        }

    # ---------------- HISTORY ----------------
    @router.get("/documents/history")
    def history(db: Session = Depends(get_db)):

        docs = db.query(Document).all()

        return [
            {
                "id": d.id,
                "filename": d.filename,
                "max_similarity": d.max_similarity,
                "level": d.level
            }
            for d in docs
        ]

    # ---------------- PDF REPORT ----------------
    @router.get("/documents/report/{document_id}")
    def generate_report(document_id: int, db: Session =
Depends(get_db)):

        document = db.query(Document).filter(Document.id ==
document_id).first()

        if not document:
            return {"error": "Document not found"}

        reports_dir = os.path.join(os.getcwd(), "reports")
```

```
        os.makedirs(reports_dir, exist_ok=True)

        pdf_path = os.path.join(reports_dir,
f"report_{document_id}.pdf")

        pdf = SimpleDocTemplate(pdf_path, pagesize=A4)
        styles = getSampleStyleSheet()

        elements = []

        # ---------------- UNIVERSITY HEADER ----------------
        elements.append(Paragraph(
            "NATIONAL UNIVERSITY OF LIFE AND ENVIRONMENTAL SCIENCES OF
UKRAINE",
            styles["Title"]
        ))
        elements.append(Spacer(1, 6))

        elements.append(Paragraph(
            "Faculty of Information Technologies",
            styles["Title"]
        ))

        elements.append(Spacer(1, 20))

        # ---------------- REPORT TITLE ----------------
        elements.append(Paragraph(
            "PLAGIARISM ANALYTICAL REPORT",
            styles["Heading1"]
        ))

        elements.append(Spacer(1, 14))

        # ---------------- BASIC INFO ----------------
        elements.append(Paragraph(
            f"<b>Document:</b> {clean_pdf_text(document.filename)}",
            styles["Normal"]
        ))

        elements.append(Paragraph(
            f"<b>Similarity score:</b> {round(document.max_similarity
* 100)}%",
            styles["Normal"]
        ))

        elements.append(Paragraph(
            f"<b>Plagiarism level:</b> {document.level.upper()}",
            styles["Normal"]
        ))

        elements.append(Spacer(1, 20))
```

```
        # ---------------- ANALYTICAL REPORT ONLY ----------------
        elements.append(Paragraph(
            "ANALYTICAL REPORT",
            styles["Heading2"]
        ))

        elements.append(Spacer(1, 10))

        match_count = len(document.matches) if document.matches else 0

        if document.level == "low":
            analysis = "The document shows minimal similarity with existing sources."
        elif document.level == "medium":
            analysis = "Moderate similarity detected. Revision is recommended."
        else:
            analysis = "High similarity detected. Significant parts match existing sources."

        elements.append(Paragraph(
            f"<b>Total detected similarities:</b> {match_count}",
            styles["Normal"]
        ))

        elements.append(Spacer(1, 6))

        elements.append(Paragraph(
            f"<b>Conclusion:</b> {analysis}",
            styles["Normal"]
        ))

        # ---------------- BUILD PDF ----------------
        pdf.build(elements)

        return FileResponse(
            path=pdf_path,
            media_type="application/pdf",
            filename=f"plagiarism_report_{document_id}.pdf"
    )
```

Додаток Б.

Обробка та підготовка тексту

```
import os
import shutil
from fastapi import UploadFile

UPLOAD_DIR = "uploads"

def save_file(file: UploadFile) -> str:
    os.makedirs(UPLOAD_DIR, exist_ok=True)

    file_path = f"{UPLOAD_DIR}/{file.filename}"

    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    return file_path

from pypdf import PdfReader
import docx
import os

def extract_text_from_pdf(path: str) -> str:
    try:
        reader = PdfReader(path)
        text = ""

        for page in reader.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"

        return text.strip()

    except Exception as e:
        raise ValueError(f"PDF extraction error: {e}")

def extract_text_from_docx(path: str) -> str:
    try:
        doc = docx.Document(path)

        text = "\n".join(
            p.text for p in doc.paragraphs if p.text and
p.text.strip()
        )

        return text.strip()

    except Exception as e:
```

```
            raise ValueError(f"DOCX extraction error: {e}")

    def extract_text(file_path: str) -> str:
        if not os.path.exists(file_path):
            raise FileNotFoundError(f"File not found: {file_path}")

        ext = os.path.splitext(file_path)[1].lower()

        if ext == ".pdf":
            return extract_text_from_pdf(file_path)

        if ext == ".docx":
            return extract_text_from_docx(file_path)


        raise ValueError(f"Unsupported file format: {ext}")
```

Додаток В.

Алгоритм очищення та preprocessing

```
# ---------------- CLEAN TEXT ----------------
def clean_pdf_text(text: str) -> str:
    if not text:
        return ""

    text = str(text)
    text = re.sub(r"[^\w\s\.,\-\(\):;%]", " ", text)
    text = re.sub(r"\s+", " ", text)

    return text.strip()
```

Додаток Г.

Алгоритм перевірки плагіату

```
    from sqlalchemy import Column, Integer, String, Text, Float, JSON

    from backend.app.db.base import Base

    class Document(Base):
        __tablename__ = "documents"

        id = Column(Integer, primary_key=True, index=True)

        filename = Column(String, nullable=False)

        file_path = Column(String, nullable=True)

        content = Column(Text, nullable=True)

        # результат перевірки
        max_similarity = Column(Float, default=0)

        # green / yellow / red
        level = Column(String, default="green")

        # список співпадінь
        matches = Column(JSON, nullable=True)

    from sqlalchemy import create_engine
    from sqlalchemy.orm import sessionmaker

    DATABASE_URL                                                    =
"postgresql://plagiarism_user:1234@localhost:5432/plagiarism_db"

    engine = create_engine(DATABASE_URL)

    SessionLocal  =  sessionmaker(autocommit=False,  autoflush=False,
bind=engine)

    def get_db():
        db = SessionLocal()
        try:
            yield db
        finally:
            db.close()
```

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Білик Владислав Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення системи перевірки плагіату» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [✓] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [✓] перекладу іншомовних джерел;
        - [✓] стилістичного редагування та перевірки граматики;
        - [✓] допомоги у написанні/відлагодженні програмного коду;
        - [✓] інше: Пошук та перевірка інформації.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ Владислав БІЛИК
(підпис)