МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
“БАКАЛАВР”

Тема: «Веб-додаток перевірки текстів на плагіат»

Виконавець: Козлов О.В.

Керівник: Глазок О.М.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Напрям (спеціальність) 123 «Комп'ютерна інженерія»
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

«____»_____2021 р.

ЗАВДАННЯ

на виконання дипломного проєкту

Козлова Олексія Вячеславовича
(прізвище, ім'я, по батькові)

1. Тема дипломного проєкту: Веб-додаток перевірки текстів на плагіат

затверджена наказом ректора від «04» Лютого 2021 р. № 135/ст.

2. Термін виконання проєкту: з 17.05.2021 по 20.06.2021

3. Вихідні дані до проєкту: мова програмування *Python*, фреймворк для Веб-розробки *Django*, алгоритм шинглів

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
Аналіз явища плагіату; аналіз алгоритмів перевірки текстів на плагіат; застосування алгоритму шинглів; користувацький інтерфейс Веб-додатку; застосування фреймворку *Django* для Веб-розробки; інструкція користувача веб додатка

5. Перелік обов'язкового графічного матеріалу:

1. Шингл (схема алгоритму);
2. Діаграма класів сутностей бази даних;
3. Екранні форми програми (Скріншоти інтерфейсу користувача);
4. Скріншоти звіту з модульного тестування додатку;

6. Календарний план

№ п/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою дипломного проєкту. Провести аналіз існуючих реалізацій програм з перевірки текстів на плагіат. Підготувати текст першого розділу пояснювальної записки	17.05.21-24.05.21	
2	Виконати проєктування додатку. Підготувати текст другого розділу пояснювальної записки	25.05.21-29.05.21	
3	Розробити Веб-додаток. Провести тестування додатку. Підготувати текст третього розділу пояснювальної записки	30.05.21-6.06.21	
4	Завершити оформлення пояснювальної записки. Оформити графічний та ілюстративний матеріал	7.06.21-10.06.21	
5	Пройти нормоконтроль. Підготувати доповідь та презентацію до захисту.	11.06.21-20.06.21	

7. Дата видачі завдання «_17_» травня 2021 р.

Керівник дипломного проєкту_____ Глазок О.М.
(підпис)

Завдання прийняв до виконання_____ Козлов О.В.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Веб-додаток перевірки текстів на плагіат»: 40 сторінок, 19 рисунків, 1 таблиця, 17 використаних джерел, 1 додаток.

Ключові слова: ВЕБ-ДОДАТОК, ПЛАГІАТ, УНІКАЛЬНІСТЬ, АЛГОРИТМ ШИНГЛІВ , *DJANGO*, *PYTHON*.

Об'єкт дослідження – процес перевірки текстів на плагіат та встановлення їх відсотку унікальності.

Предмет дослідження – веб-додаток для перевірки текстів на плагіат і встановлення відсотку унікальності заданого тексту.

Мета дипломного проєкту – розробити веб-додаток, що виконує функцію перевірки текстів на плагіат і встановлює їх унікальність.

У даній дипломній роботі спроєктовано і розроблено веб-додаток, який аналізує тексти на предмет плагіату і підраховує відсоток їх унікальності. За основу додатку взято фреймворк *Django*. Код додатку написано на мові *Python*.

Прогнозні припущення про розвиток предмету дослідження – на основі більш ґрунтовних досліджень можливо доопрацювання додатку шляхом внесення змін до алгоритмів аналізу даних та їх збереження в базі даних, що може привести до збільшення швидкості обробки даних та точності виявлення плагіату.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

MVC – Model View Controller

СВП – Система виявлення плагіату

ВП – Виявлення плагіату

SQL – Structured Query Language

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

ВСТУП

Актуальність теми. У сучасному світі зі зростаючою кількістю різних наукових або художніх робіт, які потребують дотримання унікальності. Зростає потреба в засобах, які можуть визначати унікальність текстів. Враховуючи це актуальною стає проблема швидкої обробки великої кількості текстів. Більшість існуючих систем антиплагіату є комерційними проєктами, які обмежують кількість текстів на перевірку. Отже установи, яким необхідні засоби виявлення плагіату у промислових масштабах, мають розробляти власні системи.

Наразі стрімкого розвитку набувають різні веб-додатки, це обумовлено великими конкурентними перевагами над класичними десктопними додатками, такими як: незалежність від операційної системи та пристрою користувача, єдиний значущий недолік – потреба у інтернет з'єднані, але ця проблема з часом стає все менш значимою, оскільки покриття є вже практично в кожному куточку планети. В цілому можна резюмувати, що веб-додатки мають більшу доступність для користувачів.

Об'єкт дослідження – процес перевірки текстів на плагіат та встановлення їх відсотку унікальності.

Предмет дослідження – Веб-додаток для перевірки текстів на плагіат і встановлення відсотку унікальності заданого тексту.

Мета виконання дипломного проєкту – розробити Веб-додаток, що виконує функцію перевірки текстів на плагіат і встановлює їх унікальність.

Для досягнення цієї мети необхідно виконати наступні завдання:

– виконати аналіз проблематики плагіату та способи його подолання;

– розглянути архітектуру та принципи роботи веб-додатків;

– спроєктувати та розробити веб-додаток з функцією перевірки текстів на плагіат та встановлення їх унікальності;

– протестувати розроблений додаток на предмет правильності його роботи.

Практичне значення отриманих результатів.

В ході виконання дипломного проєкту було розроблено веб-додаток для перевірки текстів на плагіат та встановлення відсотку їх унікальності. Виконано тестування додатку. Розроблений додаток може бути використаний на практиці для перевірки текстів в промислових масштабах або можна використовувати в комерційних цілях. В обох випадках розробку також можна використати як базис для подальших модифікацій, як алгоритмів для перевірки текстів, так і для користувацького інтерфейсу. База текстів, що була створена в процесі розробки та експлуатації, може бути використана для тестування модифікацій.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості про явище плагіату

Плагіат – привласнення авторства на чужий твір, а також використання у своїх працях чужого твору без посилання на автора.

Згідно чинного законодавства України передбачено кримінальну і адміністративну відповідальність за порушення авторських і винахідницьких прав [1].

Плагіат з появою Інтернету перетворився в серйозну проблему. Потрапивши в Інтернет, знання стає надбанням всіх, дотримуватися авторського права стає все важче і навіть в деяких випадках неможливо. Поступово стає складніше визначити первісного автора.

Наразі плагіат – це серйозна проблема у академічному суспільстві та студентстві. Статистичні дослідження стверджують [2], що в Україні рівень плагіату у студентських роботах сягає 34,4%. При цьому за даними міжнародних опитувань [3, 4] тільки від 2 до 9% студентів не займалися плагіатом під час виконання свої робіт. Розповсюдженість плагіату впливає на якість підготовки кадрів, а отже в перспективі є ризики економічних втрат.

Можна визначити такі основні види плагіату [5]:

– *Copy & paste* плагіат – дослівне копіювання роботи або її фрагменту без належного оформлення цитування;

– Парафраза – запозичений текст перефразується;

– Компіляція – текст створюється на основі чужих праць без самостійного дослідження та опрацювання джерел.

Загалом існує дуже багато різних класифікацій плагіату. На сайті Наукової бібліотеки Чернівецького національного університету імені Юрія Федьковича у матеріалі під назвою «Впровадження системи антиплагіату в ЧНУ» представлено 12 видів плагіату [6]: «републікація, реплікація, рерайт, фальсифікація, дослівний плагіат, мозаїчний плагіат, відсутність посилань на прямі цитати, неадекватне перефразування, поєднання власного та запозиченого тексту без цитування джерел, копіювання чужої наукової роботи та привласнення результатів праці, списування письмових робіт інших студентів, згадування джерел без посилання».

Зарубіжні науковці в першу чергу поділяють плагіат на навмисний та ненавмисний.

Американський педагог та науковець Баррі Гілмор так класифікує плагіат [7]:

Навмисний:

– списування одне в одного;

– привласнення цілої роботи;

– компіляція тексту з різних джерел;

– заміна слів у реченні.

Ненавмисний:

– копіювання речення чи двох;

– погані вміння перефразування;

– включення джерел у список бібліографії без цитування їх у тексті роботи.

1.2. Методи виявлення плагіату

Наразі для перевірки текстів на плагіат використовуються спеціальні комп'ютеризовані системи виявлення плагіату (СВП). Такі системи автоматизують більшу частину перевірки і в більшості випадків виключає потребу в аналізі тексту людиною. Більшість СВП працюють за процесом, який складається з трьох етапів [8] зображених на рис. 1.1.

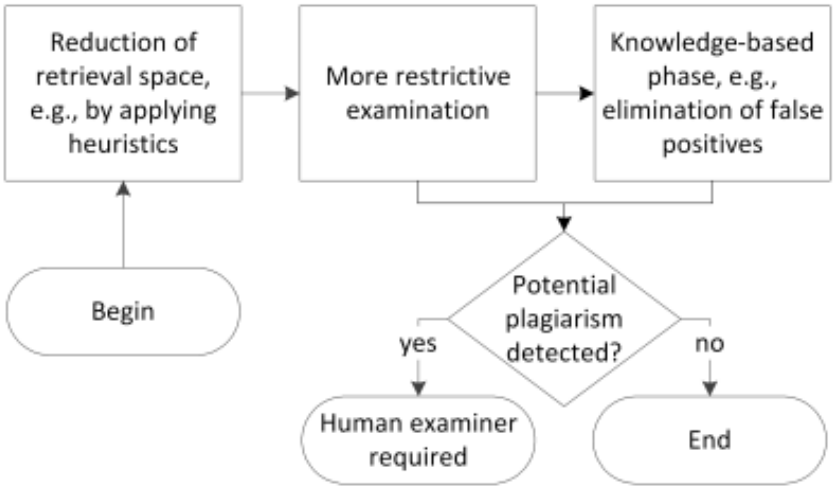

Рис. 1.1. Загальний процес виявлення плагіату

На першому етапі СВП використовує швидкі евристичні алгоритми для виявлення підозрілих ділянок тексту та можливих джерел такого тексту.

На другому етапі СВП проводить детальне порівняння текстів за допомогою повільніших алгоритмів.

На третьому етапі підозрілі фрагменти проходять аналіз на основі знань. Метою цього етапу є усунення помилково-позитивного результату, які могли виникнути на попередніх етапах. Типовим випадком таких хибних результатів можуть бути правильно оформлені цитати.

Слід зазначити, що третій етап потребує аналізу, що проводиться людиною або нейромережею, що значно збільшує вартість такої СВП, тому більшість безкоштовних СВП надають висновки ґрунтуючись на результатах другого етапу.

В процесі виявлення плагіату (ВП) використовується велика кількість методів [8], які направленні на виявлення певних видів плагіату. Методи ВП можна поділити на локальні і глобальні, як показано на рис. 1.2.

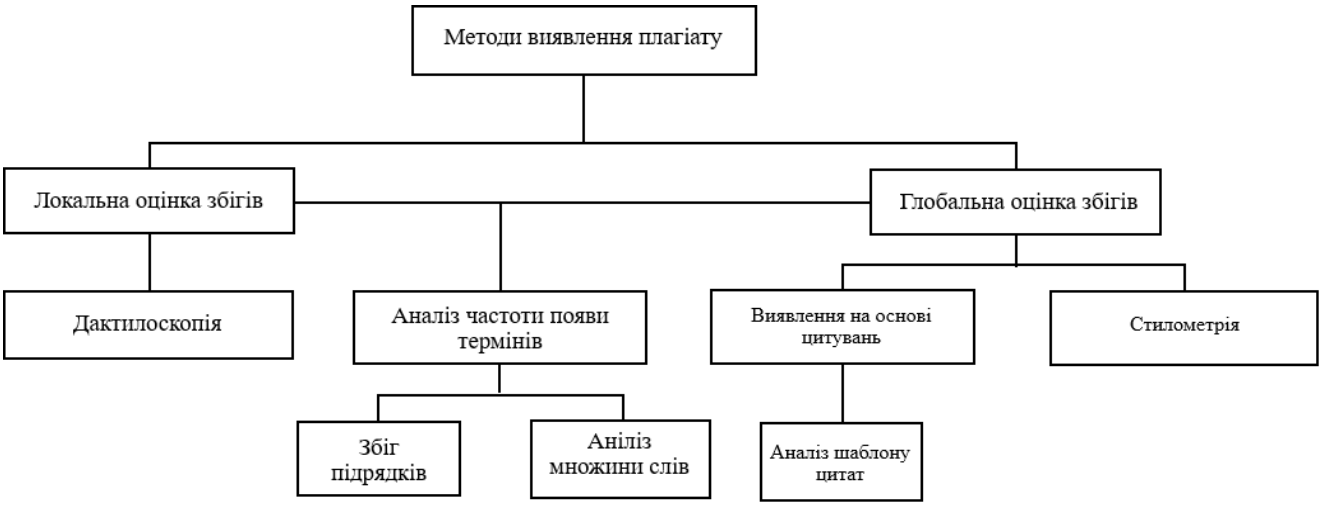

Рис. 1.2. Класифікація методів виявлення плагіату

Локальні методи оцінки збігів аналізують обмежені ділянки тексту.

Глобальні методи оцінки збігів аналізують великі частини тексту або весь текст.

Найбільш поширеним локальним методом ВП є дактилоскопія. Метод полягає у поділі тексту на підрядки. Набір підрядків розглядається як «відбиток пальця», відбиток в свою чергу поділяється на фрагменти, яким призначується хеш-код. СВП встановлює підозрілі фрагменти тексту шляхом порівняння кодів фрагментів відбитків текстів. Фрагменти відбитків розподіляються по наборах, якщо фрагмент відбитку має унікальних хеш-код то він формує набір, якщо в хеш-код фрагменту відбитку співпадає з хеш-кодом іншого фрагменту, то обидва формують один набір. Після завершення порівнянь фрагментів аналізуються створені набори. Фрагменти, що знаходяться в одному наборі потенційно містять плагіат.

Аналіз частоти появи термів – це невелика група методів ВП, які поєднує в собі, як локальну, так і глобальну оцінку збігів.

Збіг підрядків – це метод, що відносять до аналізу частоти появи термів. При аналізі СВП дослівно перевіряє текст або його фрагменти.

Аналіз множини слів – інший метод аналізу частоти появи термів. Суть методу полягає у представлення тексту у виглядів одного або декількох векторів, що представляють собою множину слів які використовуються у відповідному

тексті або його фрагменті. Вектори використовуються для попарного порівняння і виявлення потенційного плагіату.

Аналіз шаблону цитат – це найбільш поширений глобальний метод, що належить до групи методів, які виявляють плагіат на основі цитат. Метод базується на аналізі посилань двох текстів і виявлення закономірностей в послідовності цитування. При цьому метод не покладається на текстову схожість.

Стилометрія – це глобальний метод, який полягає в аналізі стилю написання тексту і встановлення авторства фрагменту на його основі.

1.3. Алгоритми виявлення плагіату

Для реалізації в СВП існує велика кількість алгоритмів, які мають свої переваги і недоліки.

Алгоритм накопичення сум (кумулятивних сум) [9] полягає в графічному порівняні двох текстів, шляхом побудови двох графіків функцій, так званих функцій речень. Функції речень будуються в декілька етапів: обрання пари характеристик, обрахунок значень функції для кожної пари, обрахування середніх значень кожної пари-функції, обрахування відхилення від середнього для кожної пари-функції; за текстом сума відхилень накопичується. Якщо графіки зростання кумулятивних сум для двох текстів майже співпадають, робиться висновок про те, що має місце запозичення. Перевагами алгоритму є можливість виявлення нечітких дублікатів. Недоліками є невелика точність і швидкість.

Алгоритм «шинглів» [9][10] розповсюджений через повноту інформації, яку можна отримати за його допомогою. Цей алгоритм полягає у порівнянні спрощених (канонічних) форм текстів. Етапи методу: канонізація – перетворення тексту до єдиної нормальної форми шляхом прибирання прийменників, союзів, розділових знаків, розбивання тексту на шингли – групи послідовних слів заданої довжини n; обчислення хеш-значень для кожного шинглу, порівняння кожного хеш-значення шинглів одного тексту з кожним хеш-значенням шинглів

іншого тексту, обрахування відношення успішних порівнянь із загальною кількістю шинглів першого тексту. Перевагами алгоритму є можливість як глобального так і локального аналізу ,гнучкість, простота, швидкість і точність у виявленні *Copy & paste* плагіату. Недоліками є неможливість виявлення нечітких дублікатів.

Алгоритм *Moodle Crot* [10]. Суть алгоритму полягає у видаленні із тексту слів до трьох символів та усіх небуквених знаків. Далі ми отримаємо ланцюжок букв, який ділимо з певним кроком n, в результаті отримаємо ланцюжки з N символів у кожному. Наступним кроком буде обрахування хеш-кодів ланцюжків та порівняння їх ланцюжками іншого тексту. Ефективність та швидкість залежить від обраних параметрів n та N, чим менше n, тим більше точність і тим менше швидкість. Перевагами алгоритму є гнучкість. Недоліками є вразливість перед прихованим плагіатом.

1.4. Загальні відомості про веб-додатки

Веб-додатки – це вид програмного забезпечення, особливістю якого є спосіб взаємодії з користувачем через інтернет браузер. На відміну від класичних комп'ютерних програм веб-додатки не потребують свого встановлення на комп'ютер користувача, що дозволяє позбавитися залежностей від операційних систем, а отже значно спростити розробку. Також більшість обрахувань виконуються на сервері, а отже такий сервіс доступний для більшої кількості користувачів, оскільки вимагає значно менше ресурсів на обрахування, ніж програма з аналогічним функціоналом, що встановлена на комп'ютері користувача. Іншим суттєвим плюсом веб-додатків є їх захищеність, оскільки інформація зберігається на сервері і користувачі мають обмежений доступ до неї. Це робить можливим існування сервісів з високими вимогами безпеки даних, такими як банківські, медицинські чи різноманітні державні послуги.

Архітектура веб-додатків [11], що схематично зображена на рис. 1.3, складається з двох основних частин: фронт-енд та бек-енд.

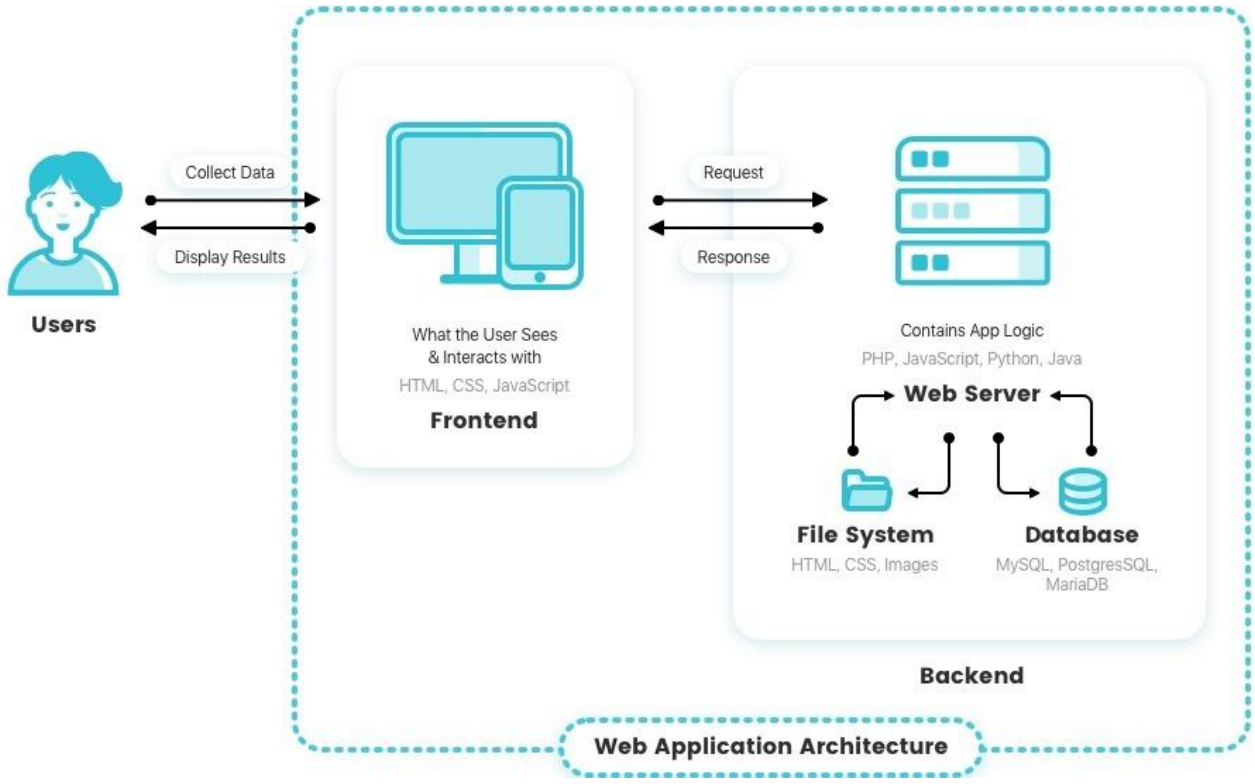

Рис. 1.3. Схема архітектури веб-додатку

Фронт-енд відповідає за інтерфейс та деяку частину обрахувань, наприклад валідація форм. Для розробки фронт-енду використовують мови розмітки *HyperText Markup Language (HTML)* та *Cascading Style Sheets (CSS)* та мову програмування *JavaScript*. Файли фронт-енду безпосередньо завантажує браузер і вони зберігаються на боці клієнта, тому інколи фронт-енд називають клієнтською стороною.

Бек-енд відповідає за більшу частину обробки даних, їх зберігання та обслуговування запитів клієнта. Для розробки використовуються наступні мови програмування: *PHP*, *JavaScript*, *Python*, *Java*, *C#*. Також бек-енд окрім файлів коду зберігає файли фронт-енду, які завантажує користувач при вході на сайт та база даних, яка в свою чергу має власний бек-енд.

Бек-енд бази даних – це програма для роботи з даними. Їх поділяють на два класи: *Structured Query Language (SQL)* або *noSQL* бази даних, до перших відносять такі програми: *MySQL*, *PostgresSQL*, *sqlLite*, до других: *MongoDB*,

MariaDB. Різниця полягає у використанні мови *SQL* для обробки даних або ж інших засобів у випадку *noSQL* бек-ендів.

Взаємодія між бек-ендом і фронт-ендом виконується за *HTTP* протоколом або у більш сучасних додатках за *HTTPS* протоколом. Різниця між версіями протоколу полягає у додаткових засобах захисту даних у *HTTPS* і більшість браузерів наразі вважають *HTTP* протокол небезпечним і помічають сайти, що його використовують відповідною міткою. Для передачі даних використовуються дві сутності *request* і *response* основна відмінність полягає у напрямі передачі даних, від клієнта до сервера та від сервера до клієнта відповідно.

1.5. Висновки до розділу

В даному розділі було визначено проблематику явища плагіату, а саме його розповсюдженість та можливі наслідки для системи освіти і економіки. Також визначено класифікацію плагіату: *Copy & paste*, парафраза, компіляція.

Описано загальний процес виявлення плагіату, що складається трьох етапів: евристичний аналіз, точний аналіз, аналіз на основі знань.

Проаналізовано методи виявлення плагіату такі як: дактилоскопія, аналіз частоти появи термів, аналіз на основі цитат та стилометрія.

Проаналізовано алгоритми виявлення плагіату, наприклад: алгоритм «шинглів», алгоритм кумулятивної суми та алгоритм *Moodle Crot*.

Також проаналізовано обраний спосіб вирішення проблеми, а саме веб-додаток та його клієнт-серверну архітектуру.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Інструментальні засоби розробки проєкту

Python – це інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією [12]. Мова *Python* була розроблена в 1990 році нідерландським програмістом Гвідо ван Россумом.

Python – мова високого рівня, що підтримує об'єктно-орієнтовану, процедурну, функціональну та аспектно-орієнтовану парадигми. За синтаксисом *Python* є *C-like* мовою.

Станом на 2021 рік *Python* є дев'ятою за розповсюдженістю технологією [13] на ринку інтернет технологій та найбільш популярною мовою програмування за *PopularitY of Programming Language index* [14], що свідчить про подальше зростання долі ринку, яку займає мова *Python*.

Однією із основних спеціалізацій мови *Python* є розробка веб-додатків. Існує велика кількість готових рішень і фреймворків для веб розробки на *Python*, таких як: *Django* і *Flask*.

Для виконання даного дипломного проєкту мова *Python* була обрана через зручність розробки, простий і інтуїтивно зрозумілий синтаксис, велику кодову базу та загальний тренд ринку.

Django – це *Python* фреймворк для веб розробки з відкритим вихідним кодом, що реалізує шаблон розробки *Model View Controller (MVC)*

Django містить велику кількість готових шаблонних рішень для таких процесів як: авторизація користувача, панель адміністратора, взаємодія з базою даних та т.п. Також існує велика кількість більш нішевих рішень, які можна напряму підключити до свого додатку, наприклад: авторизація за допомогою сервісів *Google* та асинхронна обробка запитів.

Як було зазначено вище *Django* реалізує шаблон розробки програмного забезпечення *MVC* [15] схема роботи якого зображена на рис. 2.1. Використання цього шаблону дозволяє значно спростити процес розробки, а також зробити додаток написаний на *Django* стандартизованим, що дозволяє інтегрувати такі додатки без зайвих проблем.

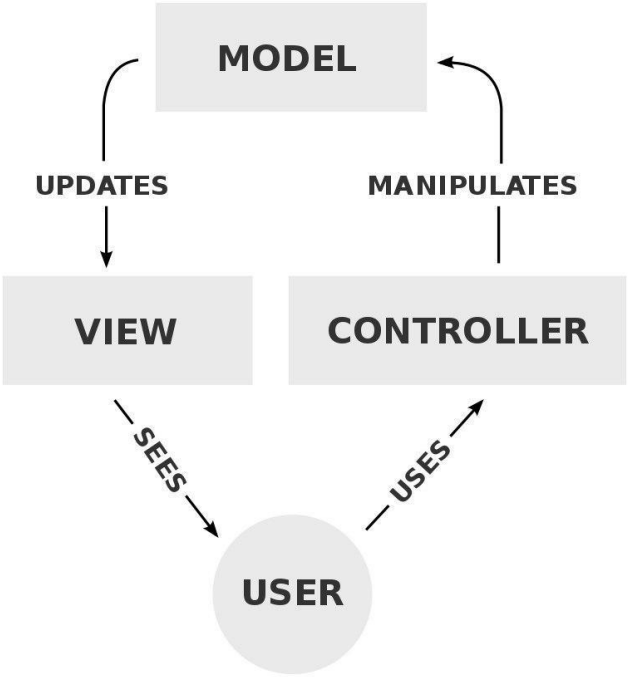

Рис. 2.1. Схема взаємодії компонентів у шаблоні *MVC*

Додаток на *Django* складається таких компонентів:

Налаштування *settings.py* – модуль, який містить глобальні змінні налаштувань таких як: секретний ключ, шляхи до інших компонентів, підключені додатки, прапор режиму відладки, бек-енд бази даних та ін. Також у *settings.py* можна додавати користувацькі налаштування для свого додатку.

Модуль-контролер *views.py* – містить функції які виконуються при переході на певну адресу. У *Django* існує невідповідність шаблону, оскільки

модуль контролю називається *views* хоча така невідповідність ніяк не впливає на роботу фреймворку і додатків написаних з його використанням.

Модуль роутингу *urls.py* – містить список адрес і відповідні їм функції-контролери.

Модуль сутності бази даних та бізнес-логіки *models.py* – класи які відповідають сутностям баз даних та їх методи. У шаблоні *MVC* відповідає компоненту *Model*. Серед розробників на *Django* розповсюджена думка, що методи класу мають виноситися в окремий файл як функції, щоб краще розділити модулі за значенням.

Папка шаблонів *templates* – папка, яка містить *HTML*-файли сторінок і у шаблоні *MVC* відповідає компоненту *view*.

Папка статичних файлів *static* – папка, яка містить *CSS*-файли, *JavaScript*-файли, зображення, які використовуються на сайті.

Під час розробки, робота ведеться в основному із перерахованими вище компонентами, проте створення своїх модулів не забороняється і деякі готові додатки також можуть мати свої унікальні модулі з якими, в разі їх використання, можлива взаємодія.

2.2. Складові веб-додатку для перевірки текстів на плагіат

Оскільки додаток розробляється з використанням фреймворку *Django*, то він має містити компоненти зазначені вище. Щодо бізнес-логіки, то додаток повинен мати модулі для аналізу текстів на плагіат та модуль для пошуку фрагментів тексту в інтернеті.

Іншим важливим компонентом додатку є база даних. Для бек-енду бази даних буде використовуватися стандартний бек-енд *Django – sqlLite*.

Додаток також потребує асинхронної обробки текстів, для цієї задачі буде використовуватися додаток *Django-celery* а для брокеру задач *Redis*.

2.3. Сутності баз даних

Додаток буде містити дві сутності бази даних: «Текст» та «Користувач». Відношення між сутностями зображено на діаграмі класів на рис. 2.2.

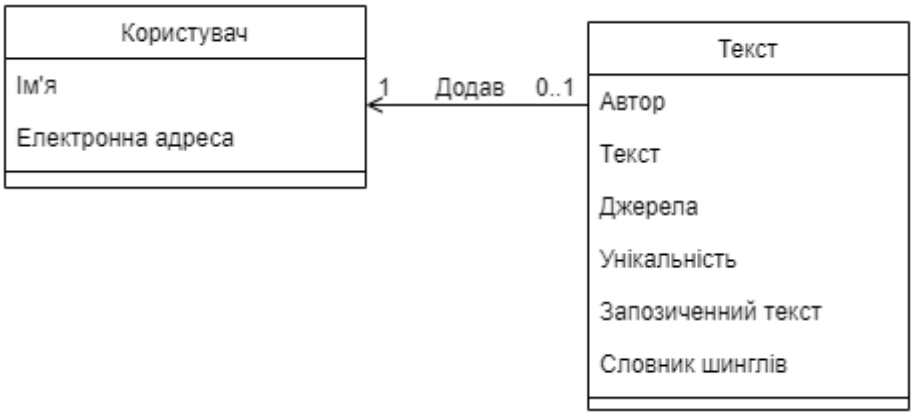

Рис. 2.2. Діаграма класів для сутності бази даних

Сутність «Користувач» містить дані про ім'я користувача та його електрону адресу.

Сутність «Текст» містить дані про текст, наприклад: авторство або джерело тексту, зміст тексту, словник шинглів, запозичений текст із джерелами, унікальність.

Між сутностями «Користувач» і «Текст» існує відношення один до багатьох по полям «Ім'я» для «Користувача» і «Автор» для «Тексту». Таке відношення означає, що текст може мати тільки одного автора, а користувач багато текстів.

2.4. Склад модуля для аналізу текстів на плагіат

Модуль для аналізу текстів на плагіат має реалізувати алгоритм для аналізу текстів на плагіат обраний за результатами порівняльного аналізу із першого розділу.

За результатами аналізу було обрано алгоритм шинглів через можливість як глобального так і локального аналізу та гнучкість налаштування. Слід також

зазначити, що для більшої точності виявлення можна використовувати декілька алгоритмів і порівнювати їх результати, проте це значно збільшує часові витрати і у випадку аналізу малих об'ємів тексту, недоцільно.

Модуль для аналізу текстів на плагіат під бути мати назву *Shingling.py* та містити реалізацію алгоритму шинглів, схема якого зображена на рис. 2.3.

Параметри алгоритму, такі як: довжина шинглу та крок шинглу, повинні імпортуватися з налаштувань додатку.

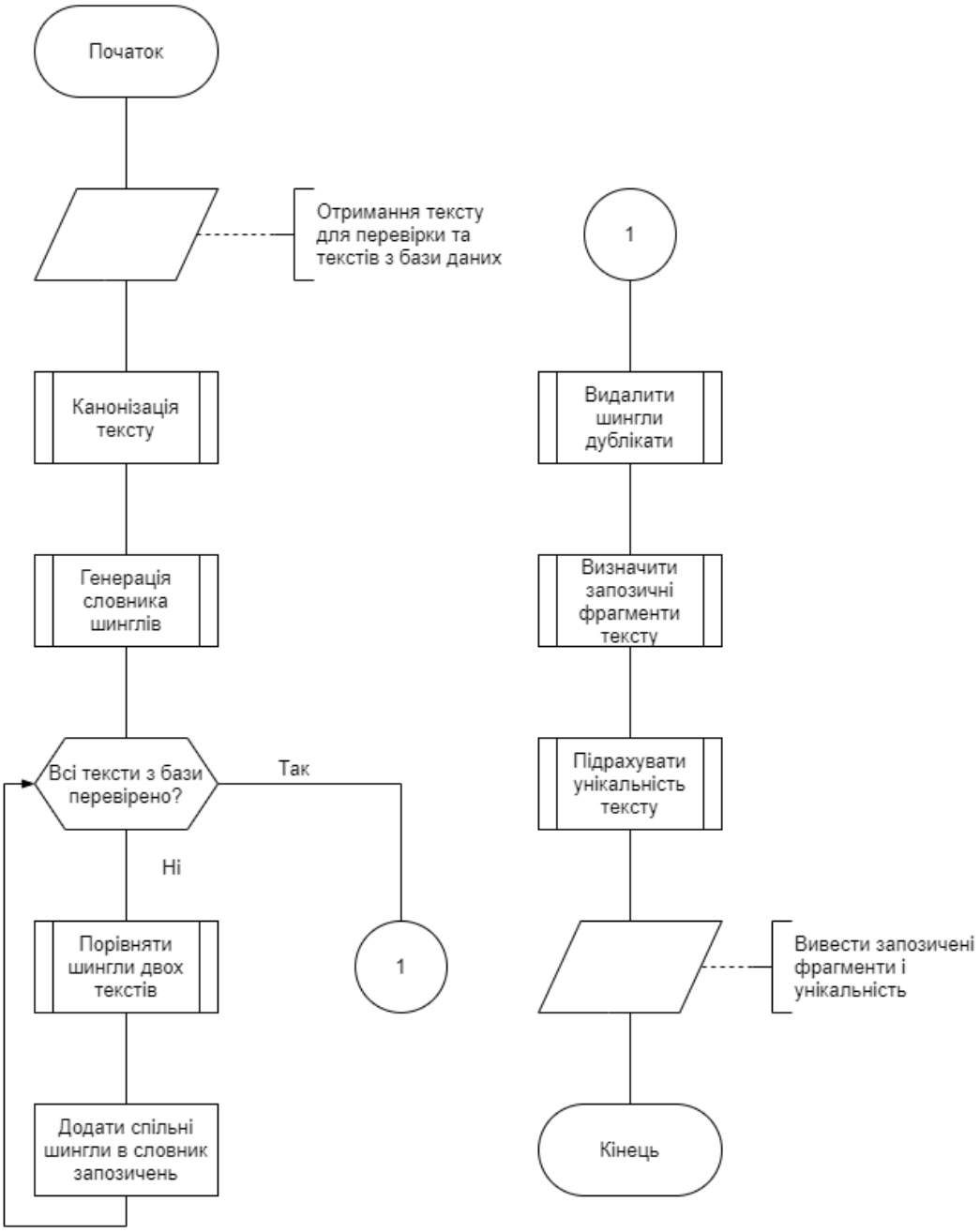

Рис. 2.3. Схема алгоритму шинглів

В результаті роботи модуля можна отримати такі дані:

– Відсоток унікальності – відсоток, який відповідає відношенню розміру унікальної частини тексту до загальної;

– Словник шинглів – містить хеш-код шинглів і відповідний до нього текст;

– Словник запозичених шинглів – містить індекс тексту з якого робилися запозичення та список відповідних до нього шинглів.

2.5. Склад модуля для парсингу веб-сторінок

Модуль для пошуку фрагментів тексту в інтернеті під назвою *Parser.py* має містити реалізацію алгоритму для пошуку посилань на текст в інтернеті зображеного на рис. 2.4. та алгоритму для отримання текстів з інтернету зображеного на рис. 2.5.

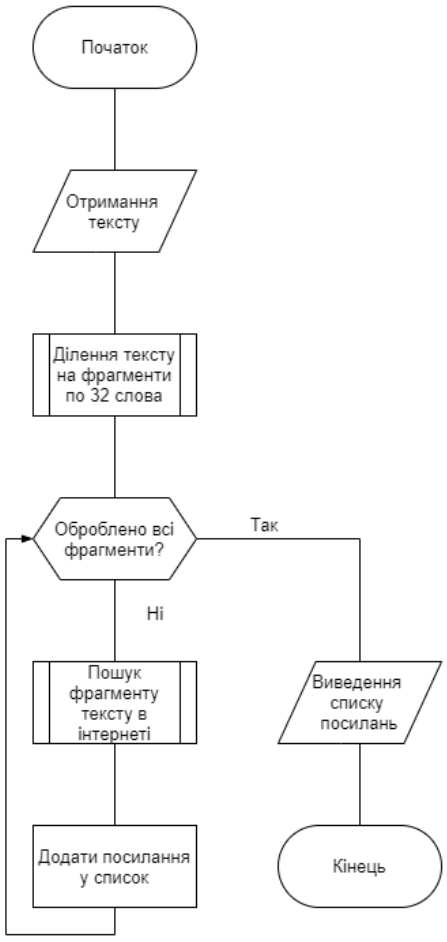

Рис. 2.4. Схема алгоритму для пошуку посилань на текст в інтернеті

Рис. 2.5. Схема алгоритму для отримання текстів з інтернету

2.6. Висновки до розділу

В даному розділі було спроєктовано компоненти веб-додатку для перевірки текстів на плагіат.

Визначено основні інструменти розробки, а саме: мова програмування *Python* та фреймворк для веб-розробки *Django* через їх зручність, розповсюдженість та велику кодову базу.

Визначено складові додатку, а саме: базові складові фреймворку *Django*, бек-енд бази даних *sqlLite*, обробник асинхронних запитів *Celery* та брокер задач *Redis*, модуль для перевірки текстів на плагіат *Shingling.py*, модуль для пошуку фрагментів тексту в інтернеті *Parser.py*.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-ДОДАТКУ ТА ТЕСТУВАННЯ ЙОГО РОБОТИ

3.1. Розробка сутностей бази даних у фреймворку *Django*

Сутності баз даних в *Django* створюються у модулі *Models.py*. Кожна окрема сутність являє собою клас на мові *Python*.

У попередньому розділі було визначено дві сутності, а саме: «Користувач» і «Текст».

Сутність «Користувач» буде створено автоматично зі всіма необхідними полями при створені проєкту *Django*, а отже написання коду не потребує.

Сутність «Текст», як було зазначено раніше, буде складатися з наступних полів: автор, джерела, зміст, запозичений текст, словник шинглів, унікальність. Також для первинного ключа буде використовуватись унікальний числовий ідентифікатор.

Поля сутності «Текст» будуть містити наступні типи даних зображені у таблиці 3.1.

Таблиця 3.1

Поля сутності «Текст» та їх типи даних

Поле	Тип даних
id	*int*
Автор	*string*
Джерела	*list*
Зміст	*text*
Запозичений текст	*dictionary*
Словник шинглів	*dictionary*
Унікальність	*float*

Словник, що відповідає полю «Запозичений текст» буде складатися ключа – *id* даного тексту та значення, яке в свою чергу також є словником, що складається з ключа – *id* тексту з якого робилося запозичення та значення – списку рядків, які були запозиченні. Приклад такої структури можна побачити на рисунку 3.1.

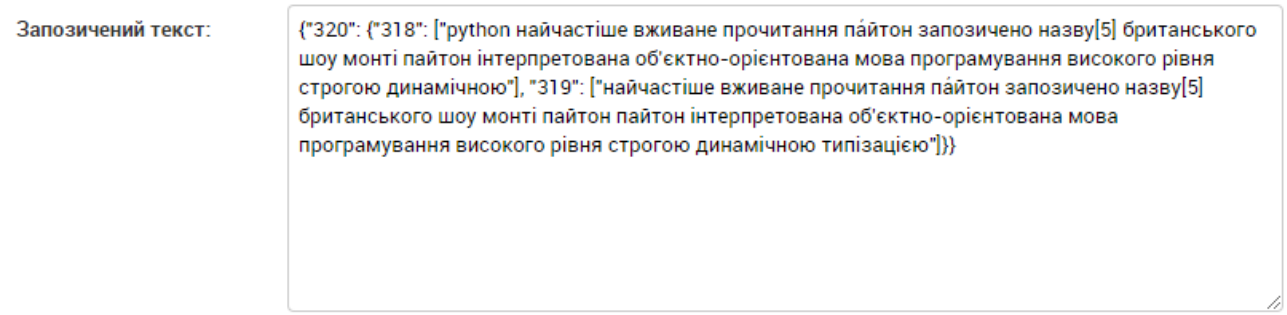

Рис. 3.1. Приклад поля «Запозичений текст»

«Словник шинглів» складається з ключа – хеш-коду шингла та значення – списку слів, що відповідають шинглу. Приклад такої структури можна побачити на рисунку 3.2.

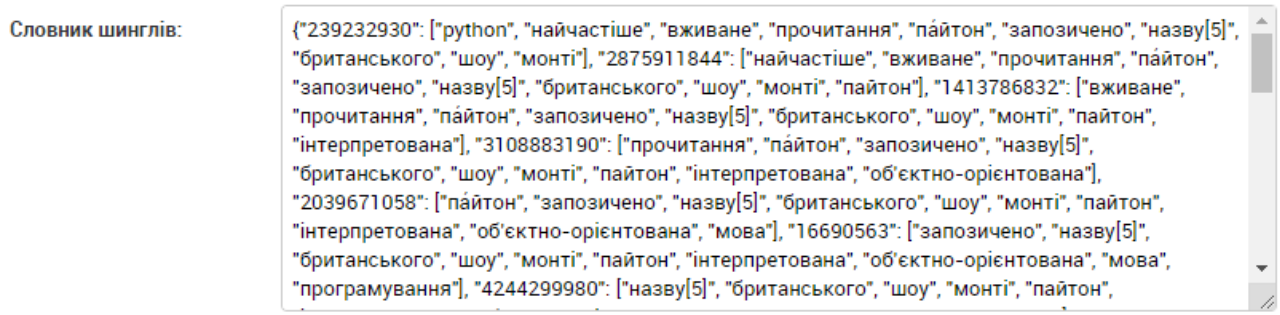

Рис. 3.2. Приклад поля «Словник шинглів»

Код класу *Text* наведено далі:

class Text(models.Model):

id = models.AutoField(primary_key=True)

*author = models.CharField(max_length=255, verbose_name="*Автор*", default="Unknown", editable=settings.DEBUG)*

*sources = models.TextField(default=[-1], verbose_name="*Джерела*", editable=settings.DEBUG)*

*content = models.TextField(verbose_name="*Текст*", editable=settings.DEBUG)*

*burrowed_content = models.TextField(default=json.dumps({"": []}), verbose_name="*Запозичений текст*", editable=settings.DEBUG)*

*shingle_dict = models.TextField(default=json.dumps({"": ""}), verbose_name="*Словник шинглів*", editable=settings.DEBUG)*

*uniqueness = models.FloatField(default=-1, verbose_name="*Унікальність*", editable=settings.DEBUG)*

3.2. Розробка модуля для аналізу текстів на плагіат на мові *Python*

Як було зазначено у попередньому розділі, модуль для аналізу текстів на плагіат має містити реалізацію алгоритму шинглів. Для реалізації необхідно розділити алгоритм на блоки кожному з яких буде відповідати своя функція.

Всі функції мають працювати з полями сутності «Текст», які були описані у попередньому пункті.

В модулі *Shingling.py* будуть наступні функції:

– Функція *Canonize* виконує канонізацію тексту. На вхід приймає неканонізований текст. Далі виконується канонізація шляхом приведення всього тексту у нижній регістр, видалення небуквених символів та частин мови, що самі по собі не несуть змістовного навантаження, такі як: сполучники, прийменники, займенники та частки. В кінці всі слова, що залишилися, розділяються по одному і формують список;

– Функція *ShingleGeneration* на основі канонізованого тексту у вигляді упорядкованого списку слів генерує шингли та відповідні їм хеш-коди. Шингл генерується шляхом поєднання *n* слів в один рядок, наступний шингл виключає перше слово попереднього та додає в кінець наступне слово після останнього. Так триває, доки весь текст не буде поділено на шингли. Загальну кількість шинглів можна обрахувати за наступною формулою:

$$S = N - (n - 1)$$

де S – загальна кількість шинглів;

N – кількість слів у канонізованому тексті;

n – довжина шинглу.

Функція *CreateShingleDictionary* на основі канонізованого тексту та хеш-кодів шинглів формує словник, де ключ: хеш-код шинглу, а значення: текст шинглу.

Функція *GetSimilarAreas* за списком хеш-кодів шинглів двох текстів встановлює спільні шингли.

Функція *RemoveDuplicates* за словниками із запозиченими шинглами поточної та попередньої версії встановлює дублікати та видаляє їх при цьому зберігаючи посилання на джерела.

Функція *GetSimilarAreasDefinition* за словником запозичених шинглів визначає запозичений текст.

Функція *SimilarityPercentageCalculation* на основі списку запозичених шинглів та всіх шинглів тексту встановлює відсоток спільних шинглів.

Код функцій наведено у Додатку А.

3.3. Розробка модуля для парсингу веб-сторінок на мові *Python*

Модуль *Parser.py* буде розроблятися із використанням бібліотеки *BeautifulSoup*, ця бібліотека містить набір функцій, що дозволяють робити запити на віддалені сервери для отримання даних.

Для пошуку текстів буде використовуватися пошукова система *Google*. Дана пошукова система може обробляти запити по 32 слова.

Модуль буде містити наступні функції:

TextSeparation – розділяє текст на частини по 32 слова.

GetSearchQueryResult – виконує запит у пошукову систему і повертає список знайдених посилань.

GetHTML – отримує весь *HTML* код сторінки за посиланням.

GetCleanLink – видаляє посилання на картинки та відео та приводить решту у стандартний вигляд.

GetWebContent – отримає змістовний текст з веб сторінки за посиланням.

Код функцій наведено у Додатку А.

3.4. Асинхронна обробка запитів за допомогою додатку *Celery*

При додані тексту на обробку він має прямувати в чергу на обробку, яка виконується в окремому процесі, щоб не заважати процесу обробки запитів користувача. Асинхронна обробка текстів буде виконуватися за допомогою *Django* додатку – *Celery*.

Для створення процесу необхідно визначити функцію яка його створює:

```
@shared_task
def check_uniqueness(user_text_id):
    FindSimilarInWeb(user_text_id)
    CompareWithDatabaseTexts(user_text_id)
    return 0
```

Функції *FindSimilarInWeb* та *CompareWithDatabaseTexts* виконують пошук тексту в інтернеті та порівняння з текстами із бази даних використовуючи попередньо описані модулі. Код функцій приведено в Додатку А.

Функція *check_uniqueness* викликається у функції *add_text* з модуля *views.py*, яка в свою чергу викликається при додаванні тексту, тобто при переході за певним посиланням.

Код функції *add_text*:

```
def add_text(request):
    author = "Unknown"
    content = request.POST.get("Content")
    if request.user.is_authenticated:
        author = request.user.username
    text_id = CreateText(content, author)
    check_uniqueness.delay(text_id)
    return HttpResponseRedirect("/text/" + str(text_id) + '/')
```

3.5. Інтерфейс користувача

Для доступу до користувацького інтерфейсу необхідно скористатися браузером та перейти за посиланням, на якому буде базуватися хост додатку. При відкритті додатку у основній частині вікна браузера буде зображено інтерфейс додатку.

Інтерфейс додатку складається з навігаційного меню та основної частини, яка в залежності від сторінки може відрізнятися.

При першому відкритті додатку у основній частині вікна браузера буде відкрита домашня сторінка додатку для неавторизованих користувачів, приклад якої зображено на рисунку 3.3.

Навігаційне меню (зверху) містить три кнопки: «Домашня сторінка», «Вхід» та «Реєстрація».

На домашній сторінці є можливість додати текст на перевірку. Для цього необхідно додати бажаний текст в поле під написом «Введіть текст» та натиснути на кнопку «Перевірити».

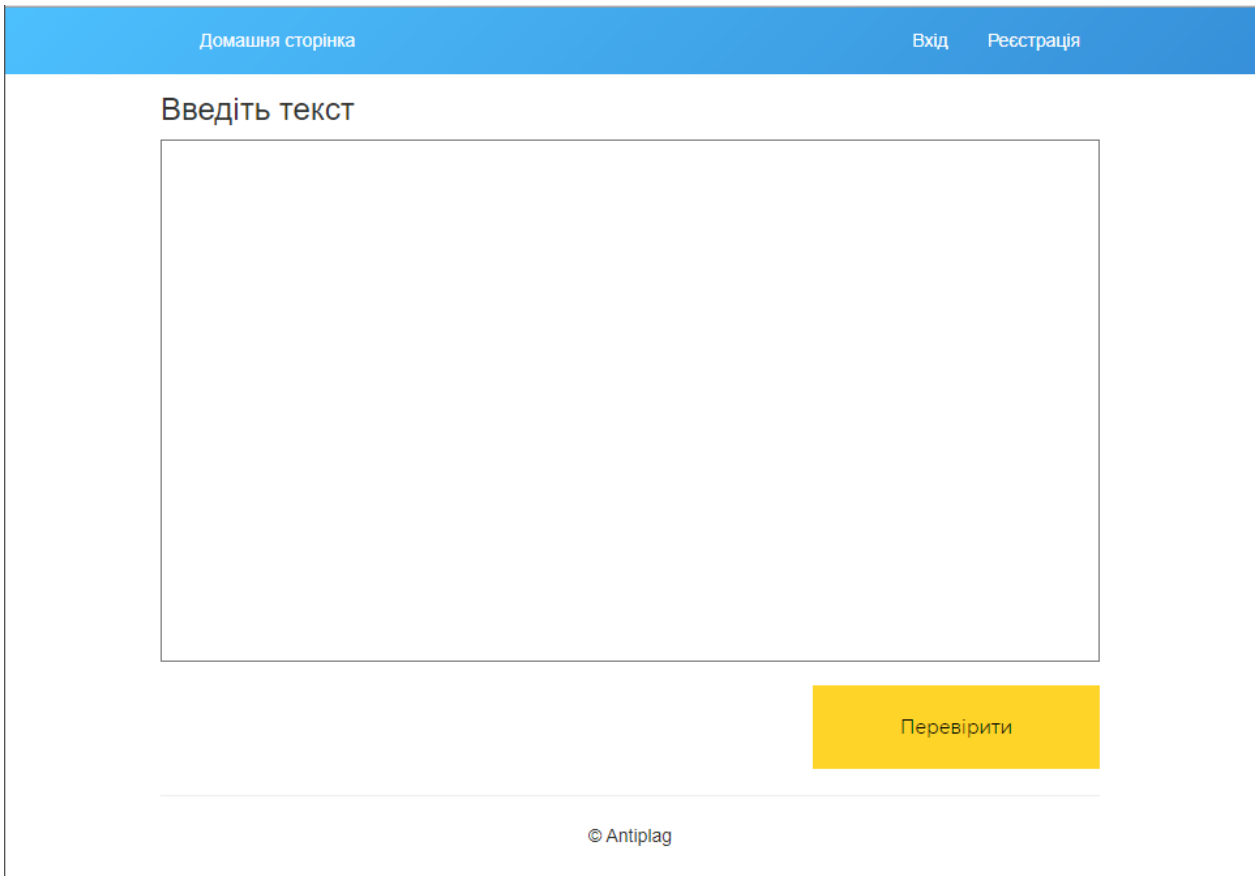

Рис. 3.3. Домашня сторінка до авторизації

При натисканні на навігаційному меню кнопки «Вхід» або «Реєстрація», користувач переноситься на сторінку авторизації, що зображена на рисунку 3.4. або реєстрації, які за зовнішнім видом не відрізняються.

Сторінка авторизації містить поля для введення ім’я користувача та паролю, а також кнопки для авторизації через сервіси *Google*, здійснення входу та відновлення паролю.

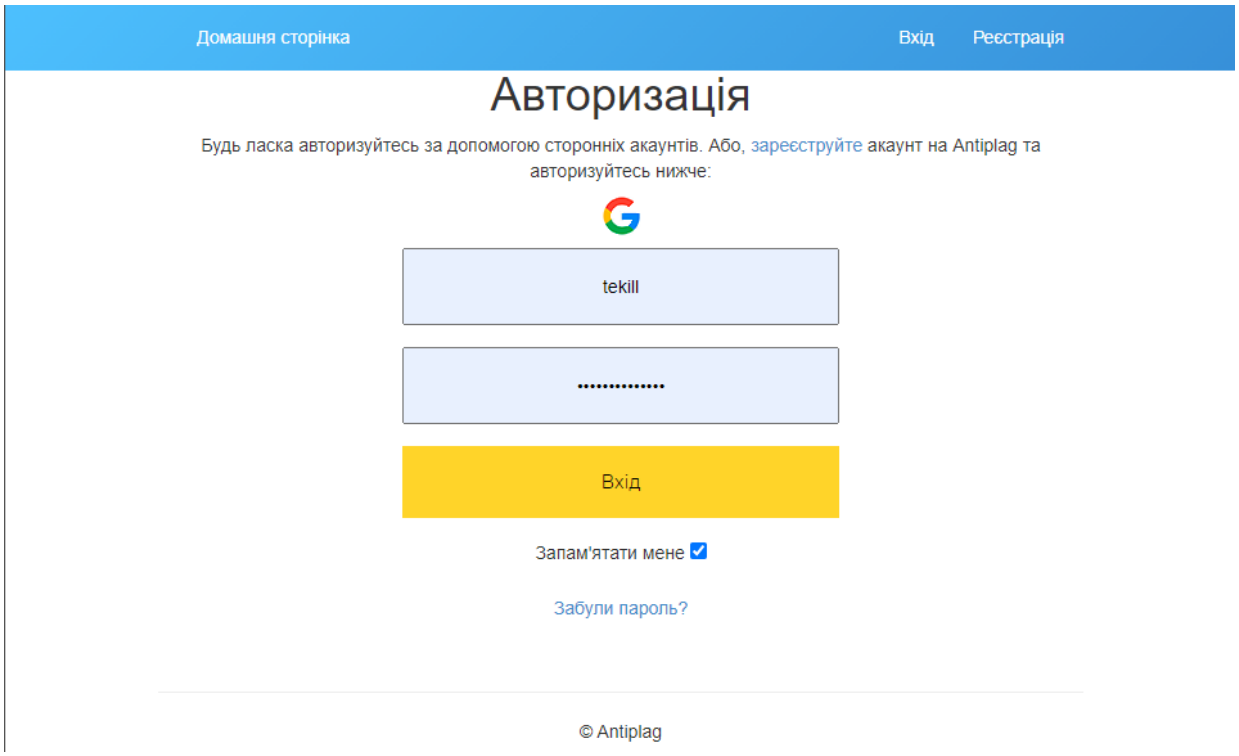

Рис. 3.4 Сторінка авторизації

Після виконання авторизації домашня сторінка змінюється. У навігаційному меню додаються вкладки «Вибірка текстів» та «Каталог текстів», також замість кнопок «Вхід» та «Реєстрація» з'являється привітання поточного користувача та кнопка «Вихід» для виходу з облікового запису. Приклад сторінки зображено на рисунку 3.5.

Основна частина сторінки не відрізняється від версії домашньої сторінки до авторизації.

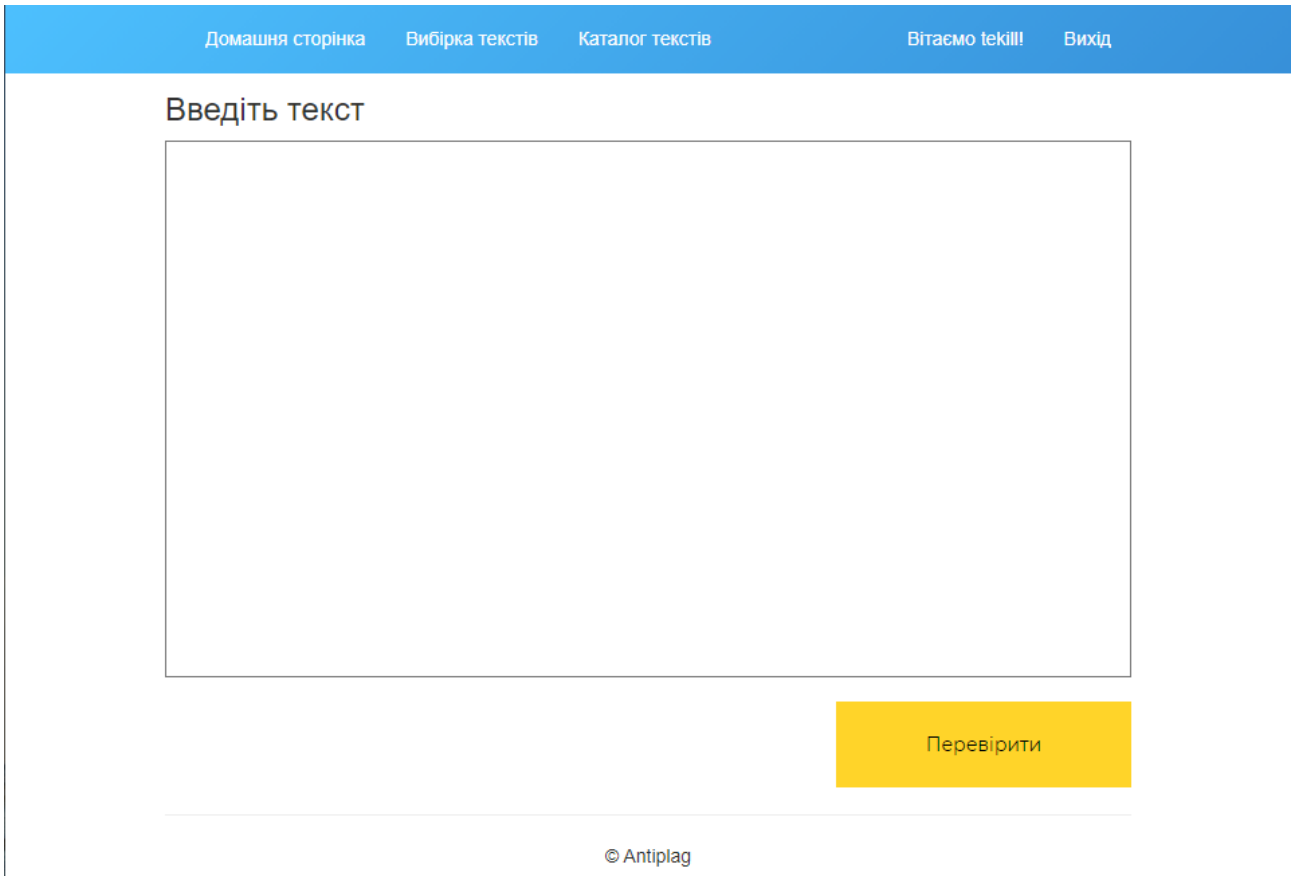

Рис. 3.5. Домашня сторінка після авторизації

На вкладці «Вибірка текстів», що зображена на рисунку 3.6. можна зробити запит в базу даних та отримати тексти із заданими в запиті параметрами фільтрів.

Фільтр по автору визначає тексти якого автора слід додати у вибірку.

Фільтр по ключовим словам джерела використовується для додання у вибірку текстів з інтернету, в посиланнях яких є введені ключові слова (ключові слова розділяються пробілом). У вибірку потраплять тексти, які мають в своєму джерелі хочаб одне ключове слово.

Фільтр по відсотку унікальності додає у вибірку тексти, унікальність яких знаходиться в заданих межах. Якщо встановити верхню границю рівну 101, то у вибірку будуть додані тексті, які було завантажено з інтернету. Якщо встановити нижню границю –1, то у вибірку будуть додані тексти, які ще не пройшли перевірку на плагіат. «Поріг успішного проходження» визначає, які тексти у звіті будуть помічені такими, що пройшли перевірку.

Фільтр по даті завантаження визначає діапазон дат, в який мали бути завантажені тексти щоб потрапити у вибірку.

Після натискання на кнопку «Зробити вибірку» користувача буде переадресовано на сторінку з результатом вибірки, яку зображено на рисунку 3.7.

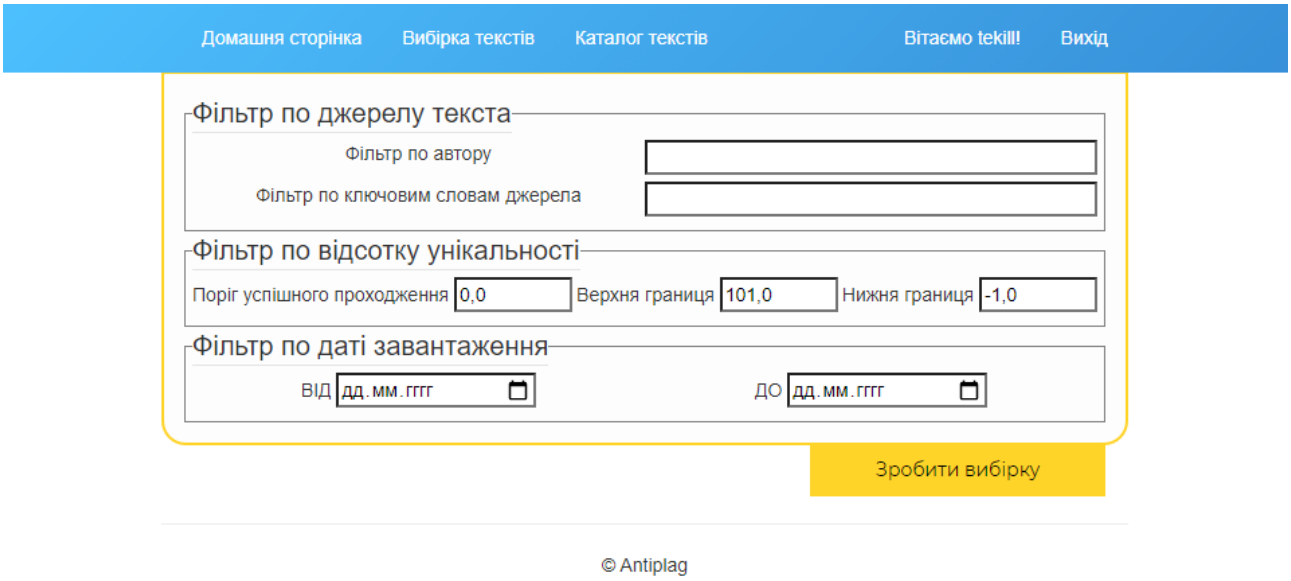

Рис. 3.6. Сторінка «Вибірка текстів»

Сторінка звіту містить таблицю, в якій зображено основну інформацію звіту (зверху) та пронумерований перелік текстів у вибірці (знизу). При натисканні на заголовок стовбця, список буде відсортовано, при одному натисканні за спаданням, при двох за зростанням. При натисканні на номер тексту у вибірці можна перейти на сторінку детального перегляду тексту. Також вибірку можна завантажити як текстовий файл, для цього необхідно натиснути на кнопку «Завантажити вибірку».

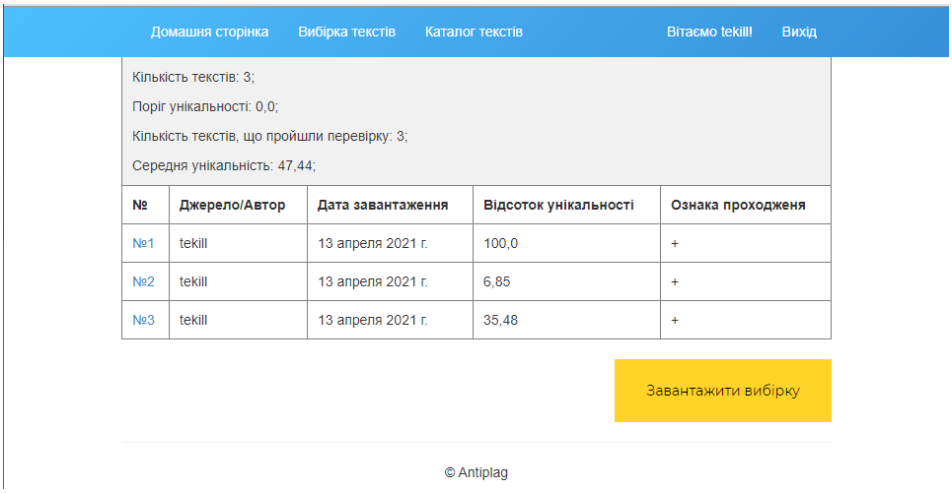

Рис. 3.7. Результат виконання запиту

При натисканні на навігаційному меню на вкладку «Каталог текстів» користувача буде переадресовано на однойменну сторінку.

На сторінці «Каталог текстів», що зображена на рисунку 3.8. можна побачити загальну інформацію про кількість текстів в базі даних, кількість текстів в обробці та кількість текстів завантажених із інтернету, а також перелік текстів в базі даних з вказанням автора або джерела тексту, дати завантаження, вікном короткого перегляду змісту тексту та кнопкою «Переглянути детально», яка при натисканні, переадресує користувача на сторінку детального перегляду конкретного тексту.

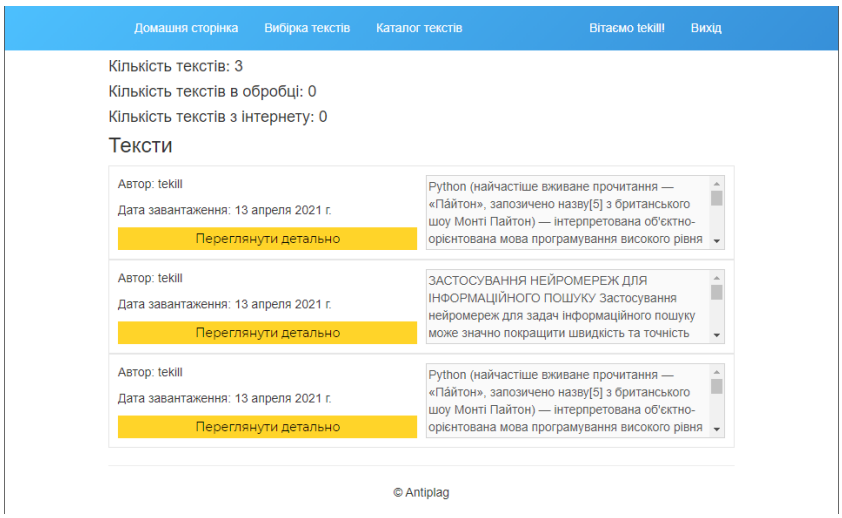

Рис. 3.8. Сторінка «Каталог текстів»

Сторінка детального перегляду текстів, зображена на рисунку 3.9. містить інформацію про унікальний ідентифікаційний номер тексту, дату завантаження тексту, унікальність, зміст, посилання на запозичення, а також в разі перегляду чужого тексту, то ім'я автора або у випадку тексту із інтернету – посилання на джерело.

У переліку запозичень відображено список посилань на тексти, з яких даний текст запозичував фрагменти та кнопку «Порівняти з текстом», яка при натисканні, переадресує користувача на сторінку детального перегляду відповідного тексту.

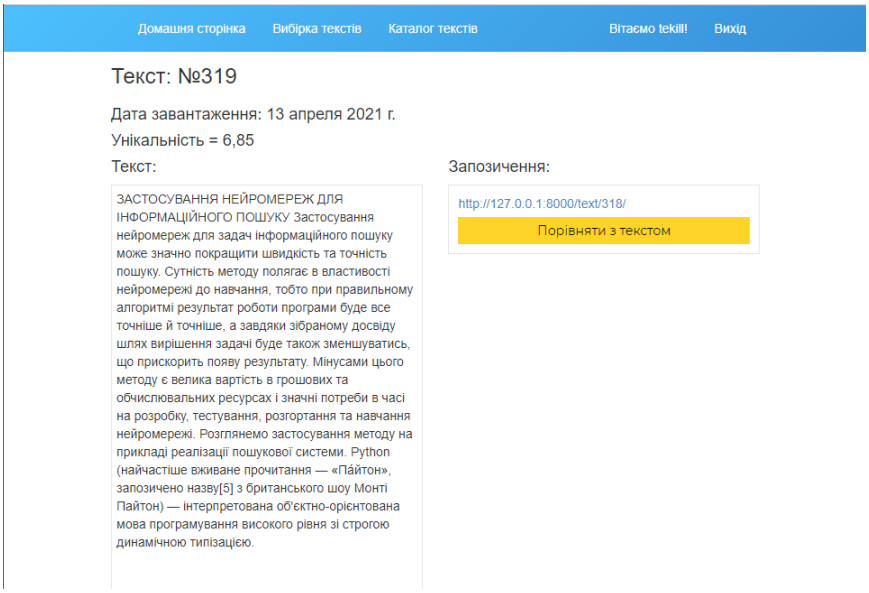

Рис. 3.9. Сторінка детального перегляду тексту

3.6. Тестування додатку

Для валідації створеного продукту буде виконуватися два види тестування: системне тестування – високорівнева перевірка функціонала всієї системи в цілому та модульне тестування для модулів аналізу текстів (*Shingling.py*) та пошуку текстів в інтернеті (*Parser.py*).

Для модульного тестування перевірки текстів на плагіат будуть створені два спеціальних тексти, які покривають різні компоновки запозичень. Для підготованих текстів унікальність не має перевищувати 10%.

Для модульного тестування пошуку текстів в інтернеті в якості тестових даних буде використовуватися запозичена із вікіпедії стаття про мову *Python*. В результаті тестування програма має повернути посилання на статтю про мову *Python*.

Результат тестування наведено на рисунках 3.10 та 3.11.

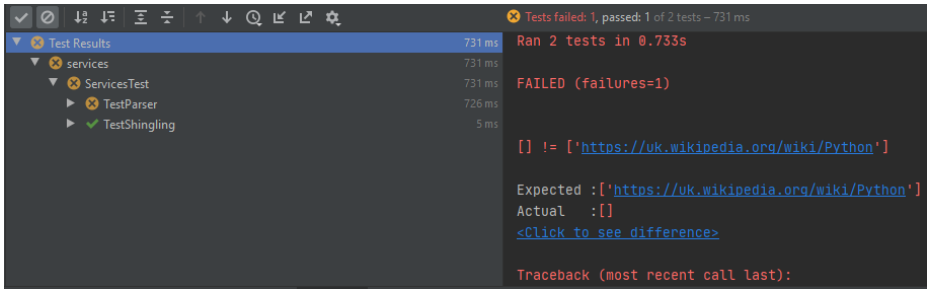

Рис. 3.10. Результат виконання модульного тестування із однією помилкою

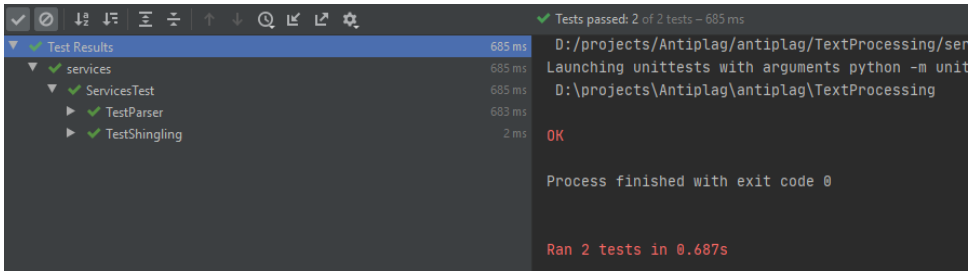

Рис. 3.11. Результат виконання модульного тестування без помилок

Як видно з результатів тестування модуль *Shingling.py* працює справно. Проте модуль *Parser.py* в певні години дня не проходить перевірку.

Системне тестування проводилося наступним чином: три користувача впродовж декількох днів використовували додаток за призначенням та відправляли звіти про помилки.

Аналіз звітів вказує на погану роботу функції пошуку текстів в інтернеті в певні години дня, що співпадає з результатами, отриманими під час модульного тестування.

3.7. Інструкція користувача

Для встановлення додатку передбачено два способи:

– Встановлення та запуск за допомогою додатку *Docker*;

В цьому випадку необхідно завантажити додаток *Docker* та з його допомогою запустити наявний в проєкті *Dockerfile*.

– Встановлення та запуск за допомогою командного рядка.

Для такого запуску необхідно мати встановлений інтерпретатор *python* версії не нижче 3.8.

Далі у командному рядку необхідно виконати команду: *pip install -r /requirements.txt*

Також необхідно завантажити додаток *redis* останньої версії та у командному рядку виконати команду: *redis-server*

У іншому вікні командного рядка необхідно виконати команду: *celery -A Antiplag worker --pool=solo -l info*

У інтерпретаторі *python* необхідно виконати команду: *python manage.py runserver 127.0.0.1:8000*

В обох випадках після виконання всіх дій потрібно в обраному браузері у пошуковому рядку за посиланням: *http://127.0.0.1:8000/* можна зайти на головну сторінку додатку.

3.8. Висновки до розділу

В даному розділі було виконано розробку і тестування веб-додатку для перевірки текстів на плагіат.

Були розроблені модулі: *Parser.py*, *Shingling.py*, *Models.py*, *Views.py* та ряд *HTML*, *CSS*, *JavaScript* файлів для реалізації інтерфейсу користувача.

В ході тестування було виявлені несправності в роботі модуля *Parser.py*. Під час спроб виправлення несправності було виявлено, що помилка виникає через блокування запиту пошуковою системою *Google*. На жаль вирішення проблеми потребує грошових ресурсів, вкладення яких не було можливим в рамках виконання дипломного проєкту.

Решта функцій додатку працює справно, що підтверджується як модульним, так і системним тестуванням.

ВИСНОВКИ

У даному дипломному проєкті було спроєктовано і розроблено веб-додаток для перевірки текстів на плагіат на базі фреймворку *Django*. До основних функцій даного додатку входять: аналіз тексту на плагіат та обрахування його відсотку унікальності, виявлення ділянок запозиченого тексту, виявлення джерел запозичень.

У роботі досліджено предметну область, проблематику явища плагіату та веб-додатки.

На основі розглянутих даних було зроблено висновок про розповсюдженість явища плагіату у сучасному академічному суспільстві та необхідність боротьби із ним за допомогою в тому числі із застосування автоматизованих систем із архітектурою веб-додатків. Такий вибір архітектури був обумовлений доступністю для користувачів та популярністю даної архітектури.

Розроблений веб-додаток було протестовано, і хоча виявлено деякі проблеми з функцією пошуку текстів в інтернеті, додаток працює справно і свою основну задачу виконує. Такий висновок було зроблено за результатами системного та модульного тестування.

Актуальність розробки полягає у відкритості коду, наразі більшість безкоштовних СВП обмежують за об’ємом тексти, які можна з їх допомогою перевіряти, дана розробка позбавлена таких обмежень. Будь-яка організація може використовувати дану роботу з метою виявлення плагіату. Параметри перевірки можна налаштовувати, що дає можливість балансувати витрати ресурсів і точність виявлення плагіату. Іншою перевагою проєкту є архітектура веб-додатку *Django*, в якій додаток можна підключити в інший проєкт або ж до самого додатку підключити інші з метою, наприклад, додання нових алгоритмів ВП.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про авторське право і суміжні права : Закон України від 23 грудня 1993 р. *No* 3792-XII // Відомості Верховної Ради України. – 1994. – *No* 13. – Ст. 64.

2. Рівень плагіату в студентських роботах у країнах Європи [Електронний ресурс]. – 2016 – Режим доступу: *http://aphd.ua/riven-plahiatu-v-studentskykh-robotakh-u-kranakh-ievropy-/* (дата звернення 17.05.2021). – Назва з екрану.

3. *Biliæ-Zulle L., Frkoviæ V., Turk T., Azman J., Petroveèki M. Prevalence of Plagiarism among Medical Students, Croat Med, J* 2005, 46 (1), 126-131.

4. *Krishna Sharma B. Plagiarism among University Students: Intentional or Accidental? // Journal of NELTA Vol. 12 No. 1&2 December* 2007, 134-141.

5. Програмне забезпечення для перевірки наукових текстів на плагіат: інформаційний огляд // автори-укладачі: А.Р. Вергун, Л.В. Савенкова, С.О. Чуканова ; редколегія: В.С. Пашкова, О.В. Воскобойнікова-Гузєва, Я.Є. Сошинська ; Українська бібліотечна асоціація. – Київ : УБА, 2016. – Електрон. вид. – 1 електрон. опт. диск (*CDROM*). – 36 с.

6. Впровадження системи антиплагіату в ЧНУ // Наукова бібліотека Чернівецького національного університету ім. Ю. Федьковича. [Електронний ресурс]. – 2016 – Режим доступу: *http://www.library.chnu.edu.ua/index.php?page=ua/archive&data[5008][news_id]=6209* (дата звернення 20.05.2021). – Назва з екрану.

7. *Gilmore B. Plagiarism: A How-Not-to Guide for Students. Portsmouth, NH: Hienemann*, 2009. 104 *p*.

8. *Meuschke, N. State-of-the-art in detecting academic plagiarism / N. Meuschke, B. Gipp // Intern. J. for Educational Integrity. – 2013. – Vol. 9, No. 1. – P. 50–71.*

9. Козлов О.В. Методи семантичного аналізу текстової інформації в програмних засобах // Сучасні тенденції розвитку системного програмування Тези доп. наук.-практ. Конф. 25-26 листопада 2020 р. – К.: НАУ, 2020. – С. 53.

10. Білощицький А.О. Ефективність методів пошуку збігів у текстах / А.О. Білощицький, О.В. Діхтяренко // Управління розвитком складних систем. – 2013. – Вип. 14. – С. 144-147.

11. *The Fundamentals of Web Application Architecture* [Електронний ресурс]. – 2020 – Режим доступу: *https://reinvently.com/blog/fundamentals-web-application-architecture/* (дата звернення 23.05.2021). – Назва з екрану

12. *Guido van Rossum, Python Reference Manual, release 2.4.4,* [Електронний ресурс]. – 2006. – Режим доступу: *http://ft-sipil.unila.ac.id/dbooks/Python%20Reference%20Manual.pdf* (дата звернення 22.05.2021). – Назва з екрану.

13. *Programming Languages Market Share* [Електронний ресурс]. – 2020. – Режим доступу*: https://www.datanyze.com/market-share/programming-languages--67* (дата звернення 22.05.2021). – Назва з екрану.

14. *PYPL PopularitY of Programming Language* [Електронний ресурс]. – 2020. – Режим доступу: *https://pypl.github.io/PYPL.html* (дата звернення 20.05.2021). – Назва з екрану.

15. *Django documentation* [Електронний ресурс]. – 2020. – Режим доступу: *https://docs.djangoproject.com/en/3.2/* (дата звернення 22.05.2021). – Назва з екрану.

16. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

17. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.»

ДОДАТОК А

КОД ПРОГРАМИ

```
"""
Module contains business-logic responsible for comparison of texts
"""

import binascii
from typing import Union, List, Dict, Tuple
from pymorphy2 import MorphAnalyzer
from django.conf import settings

morph = MorphAnalyzer(lang='uk')
if __name__ != "__main__":
    from antiplag.TextProcessing.services.StopSymbols import stop_words, stop_symbols
    shingle_len = 10
    shingle_step = 1
else:
    from .StopSymbols import stop_words, stop_symbols
    shingle_len = settings.SHINGLE_LENGTH
    shingle_step = settings.SHINGLE_STEP
    pass

def Canonize(source: str) -> List[str]:
    return [x for x in [y.strip(stop_symbols) for y in source.lower().split()] if x and (x not in stop_words)]
```

```
    # generates shingles
    def ShingleGeneration(source: List[str]) -> List[int]:
        words_crc = []
        for i in range(len(source) - (shingle_len - 1)):
            words_crc.append(binascii.crc32(' '.join([x for x in source[i:i + shingle_len]]).encode('utf-8')))
        return words_crc


    # Splits the text and returns dictionary of hashes and phrases
    def CreateShingleDictionary(text: str) -> Dict[int, List[str]]:
        canonized = Canonize(text)
        hashes = ShingleGeneration(canonized)
        words = []
        for i in range(len(canonized) - (shingle_len - 1)):
            words.append([x for x in canonized[i:i + shingle_len]])
        words_dict = dict(zip(hashes, words))
        return words_dict


    # compares texts and returns list of similar hashed shingles
    def GetSimilarAreas(source1: List[int], source2: List[int]) -> List[int]:
        similar_phrases = []
        for i in range(0, len(source1), shingle_step):
            if source1[i] in source2:
```

```
            similar_phrases.append(source1[i])
        return similar_phrases


    # returns common parts of the texts
    def GetSimilarAreasDefinition(text1_dictionary: Dict[int, List[str]],
                                  compared_texts: Dict[Union[int, str], List[Union[int, str]]],
                                  text_type=0)\
        -> Union[Dict[Union[int, str], List[Union[int, str]]], List[List[str]]]:
        list_of_areas = []
        for i in range(len(compared_texts)):
            list_of_local_areas = []
            local_areas = []
            for k in range(len(list(compared_texts.values())[i])):
                if list(compared_texts.values())[i][k] in text1_dictionary:

local_areas.append((text1_dictionary.get(list(compared_texts.values())[i][k])))
            area = ' '.join(local_areas[i])
            for k in range(len(local_areas) - 1):
                same_words = sum(word in local_areas[k] for word in local_areas[k +
1])
                if same_words == shingle_len - 1:
                    area += ' '
                    area += local_areas[k + 1][-1]
                elif same_words == shingle_len:
                    pass
                else:
```

```
                list_of_local_areas.append(area)
                area = ' '.join(local_areas[k + 1])
            list_of_local_areas.append(area)
            list_of_areas.append(list_of_local_areas)

        if text_type:
            dict_of_areas = {}
            for i in range(len(list_of_areas)):
                dict_of_areas.update({list(compared_texts.keys())[i]: list_of_areas[i]})
            return dict_of_areas
        else:
            return list_of_areas


    def RemoveDuplicates(similar_phrases: Dict[Union[int, str], List[int]],
                         new_similar_phrases: Dict[int, List[int]]) -> Dict[Union[int, str],
List[int]]:
        duplicates_keys = []
        transformed_dict = {}
        similar_phrases.update(new_similar_phrases)

        for key in similar_phrases:
            if str(key) != str(list(new_similar_phrases.keys())[0]):
                if list(new_similar_phrases.values())[0] == similar_phrases[key]:
                    sources = str(key) + "_" + str(list(new_similar_phrases.keys())[0])
                    transformed_dict.update({sources:
list(new_similar_phrases.values())[0]})
```

```
                duplicates_keys.append(key)
                duplicates_keys.append(list(new_similar_phrases.keys())[0])

        if duplicates_keys:
            for key in duplicates_keys:
                del similar_phrases[key]
            similar_phrases.update(transformed_dict)

        return similar_phrases


    # counts the percentage of two texts, will be changed later
    def SimilarityPercentageCalculation(source: List[int], burrowed: List[int]) ->
float:
        same = 0
        for i in range(len(source)):
            if source[i] in burrowed:
                same = same + 1
        try:
            return float(format(100 - same*2/float(len(source) + len(burrowed))*100,
'.2f'))
        except ZeroDivisionError:
            return 0.0
```

```
"""
Module contains business-logic responsible for finding fragments of text in web
"""
import math
import requests
from typing import Union, List
from bs4 import BeautifulSoup
from django.conf import settings


# debug
if __name__ == "__main__":
    pages_of_search = 1
    search_results = 1
else:
    search_results = settings.SEARCH_RESULTS
    pages_of_search = search_results // 10 + 1
    pass


# function returns web page html code or 0 in case of failure
def GetHTML(url: str) -> Union[int, requests.Response]:
    try:
        HTMLCode = requests.get(url)
    except requests.exceptions.RequestException:
        return 0
    else:
        return HTMLCode
```

```
    # function parsing search result
    def GetSearchQueryResult(search_query: str) -> List[str]:
        QueryResultUrls = []
        SearchUrl = 'http://www.google.com/search?q=' + search_query + '&start='
        for NumOfPage in range(pages_of_search):
            SearchResult = GetHTML(SearchUrl + str(NumOfPage * 10))
            if SearchResult:
                QueryResultUrls = GetCleanLink(SearchResult)
        return QueryResultUrls


    # function cleans correct link
    def GetCleanLink(search_result: Union[int, requests.Response]) -> List[str]:
        ClearedLinks = []
        soup = BeautifulSoup(search_result.text, "html.parser")
        AllLinks = soup.find_all('div', class_="BNeawe UPmit AP7Wnd")
        for Link in AllLinks:
            LinkIsCorrect = (Link.string.rfind('youtube') == -1) and
(Link.string.rfind('yandex') == -1) and \
                        (Link.string.rfind('mail.ru') == -1) and (Link.string.rfind('.jpg')
== -1) and (
                                Link.string.rfind('.png') == -1) and (Link.string.rfind('.gif'))
== -1
            if LinkIsCorrect:
                ClearedLinks.append(Link.string.replace(" › ", "/"))
```

```
        return ClearedLinks


    # function extracts web page textual content
    def GetWebContent(url: str) -> str:
        CleanText = ""
        WebResource = GetHTML(url)
        if WebResource:
            HTMLCode = BeautifulSoup(WebResource.text, "html.parser")
            for Script in HTMLCode(["script", "style"]):
                Script.extract()
            Text = HTMLCode.get_text().split("\n")
            for Phrase in Text:
                PhraseIsMeaningful = (Phrase != '') and (
                        (Phrase[-1] == '.') or (Phrase[-1] == '?') or (Phrase[-1] == '!') or
(Phrase[-1] == ':') or (
                         Phrase[-1] == ';'))
                if PhraseIsMeaningful:
                    CleanText += str(Phrase)
                    CleanText += "\n"
            del Text
            return CleanText
        else:
            return "0"


    # function divides text into 32-word phrases
```

```
    def TextSeparation(text: str) -> List[str]:
        SeparatedPhrases = []
        ListOfWords = text.split()
        NumOfWords = len(ListOfWords)
        NumOfParts = math.ceil(NumOfWords / 32)
        for PartIdx in range(NumOfParts):
            SeparatedPhrases.append(" ".join(ListOfWords[32 * (PartIdx - 1): 32 *
PartIdx]))
        del SeparatedPhrases[0]
        return SeparatedPhrases


    # function finds text references in web
    def FindTextUrls(text: str) -> List[str]:
        Urls = []
        AllParts = TextSeparation(text)
        for Query in AllParts:
            try:
                Urls += GetSearchQueryResult(Query)[0:search_results]
            except IndexError:
                return ["0"]
            else:
                return list(set(Urls))
```

```
    """
    Module contains business-logic responsible for operating and extraction data
from data base
    """

    import json
    from typing import List, Dict, Tuple
    from .Parser import *
    from .Shingling import *
    from ..models import Text
    from django.db.models import Q
    from typing import NamedTuple

    class SelectionRequest(NamedTuple):
        author_filter: str
        key_words_filter: List[str]
        uniqueness_upper_border: float
        uniqueness_down_border: float
        left_date: str
        right_date: str

        def GetSelectionSet(self):
            selection_set =
Text.objects.filter(uniqueness__range=(self.uniqueness_down_border,
                                                    self.uniqueness_upper_border))

            if self.author_filter != "":
                selection_set = selection_set.filter(author=self.author_filter)

            if self.key_words_filter:
                for key_word in self.key_words_filter:
```

```
                selection_set = selection_set.filter(source__icontains=key_word)

            if self.left_date == "" and self.right_date != "":
                selection_set = selection_set.filter(upload_date__lte=self.right_date)
            elif self.left_date != "" and self.right_date == "":
                selection_set = selection_set.filter(upload_date__gte=self.left_date)
            elif self.left_date != "" and self.right_date != "":
                if self.left_date < self.right_date:
                    selection_set =
selection_set.filter(upload_date__range=(self.left_date, self.right_date))
                else:
                    selection_set =
selection_set.filter(upload_date__range=(self.right_date, self.left_date))

            return selection_set


        # function creates text in database and returns its id in database
        def CreateText(content: str, author: str = "Unknown") -> int:
            text = Text()
            text.author = author
            text.content = content
            text.save()
            return text.id


        # function returns text content by id
        def GetTextContent(text_id: int) -> str:
            text = Text.objects.get(id=text_id)
            return text.content
```

```
    # function returns text burrowed content by main text id and second text id(if
needed)
    def GetTextBurrowedContent(main_text_id: int, second_text_id: int = -1) -> str:
        main_text = Text.objects.get(id=main_text_id)
        text_burrowed_content = json.loads(main_text.burrowed_content)
        if second_text_id == -1:
            return text_burrowed_content.get(str(main_text_id))
        else:
            return text_burrowed_content.get(str(second_text_id))


    # function separates burrowed content and its source in another database field
    def SeparateBurrowedContent(text_id: int, main_burrowed_content: Dict[int,
List[str]],
                                another_texts_burrowed_content: Dict[int, List[str]]) ->
None:
        text = Text.objects.get(id=text_id)
        burrowed_content = {text.id: main_burrowed_content}
        # burrowed_content.update(another_texts_burrowed_content)
        text.burrowed_content = json.dumps(burrowed_content, ensure_ascii=False)
        text.save()


    # function finds and adds texts with content similar to text content by id
    def FindSimilarInWeb(text_id: int) -> None:
        text = Text.objects.get(id=text_id)
        SetOfUrls = FindTextUrls(text.content)
        if SetOfUrls:
            for url in SetOfUrls:
                if not Text.objects.filter(source=url).exists():
```

```
                web_text = Text()
                web_text.author = "web"
                web_text.source = url
                web_text.content = GetWebContent(url)
                web_text.uniqueness = 101
                if web_text.content != "0":
                    web_text.shingle_dict =
json.dumps(CreateShingleDictionary(web_text.content),
                                                ensure_ascii=False)
                    web_text.save()
                else:
                    del web_text


        # function finds similar areas in text by id and
        # returns list with sources, similar parts and dictionary with sources id as a key
and text parts as a value
        def FindSimilarAreas(text_id: int, user_text_shingles: List[int])\
            -> Tuple[List[int], Dict[int, List[int]], Dict[int, List[str]]]:
            sources: List[int] = []
            similar_parts: Dict[int, List[int]] = {}
            database_text_similar_content: Dict[int: List[str]] = {}
            for data_base_text in
Text.objects.exclude(id=text_id).exclude(uniqueness=0.0):

                database_text_shingles = [int(item) for item in
list(json.loads(data_base_text.shingle_dict).keys())]

                similar_part = {data_base_text.id: GetSimilarAreas(user_text_shingles,
database_text_shingles)}
```

```
            if similar_part[data_base_text.id]:
                sources.append(data_base_text.id)

                str_similar_part = [str(item) for item in list(similar_part.values())[0]]

                similar_parts = RemoveDuplicates(similar_parts, similar_part)

                # similar_parts.update(similar_part)

                data_base_text_shingle_dict = json.loads(data_base_text.shingle_dict)

                data_base_similar_content =
GetSimilarAreasDefinition(data_base_text_shingle_dict,
                                                    {data_base_text.id: str_similar_part})
                database_text_similar_content[data_base_text.id] =
data_base_similar_content

            return sources, similar_parts, database_text_similar_content



        # function compares text by id with other texts in database and compute its
uniqueness
        # adds sources and similar parts in sources
        # EXPLANATION OF IMPLEMENTATION: function contains many actions
because it
        # contains a lot of calculations with a lot of supporting data useful for other
calculations in the future
        def CompareWithDatabaseTexts(text_id: int) -> None:
            text = Text.objects.get(id=text_id)

            user_text_shingle_dict = CreateShingleDictionary(text.content)
```

```
        text.shingle_dict = json.dumps(user_text_shingle_dict, ensure_ascii=False)

        user_text_shingles = list(user_text_shingle_dict.keys())

        sources, similar_parts, database_text_similar_content =
FindSimilarAreas(text_id, user_text_shingles)

        try:
            text.sources = json.dumps(sources)
        except TypeError:
            text.sources = -1
        try:
            text.uniqueness = SimilarityPercentageCalculation(user_text_shingles,
list(similar_parts.values())[0])
        except IndexError:
            text.uniqueness = 100.0
        if text.uniqueness < 0:
            text.uniqueness = 0.0
        text.save()
        if text.uniqueness != 100.0:
            user_text_similar_content =
GetSimilarAreasDefinition(user_text_shingle_dict, similar_parts, 1)
            SeparateBurrowedContent(text_id, user_text_similar_content,
database_text_similar_content)
```

```
"""
Module contains controller logic
"""

from django.contrib.auth.models import User
from django.shortcuts import render
from django.http import HttpRequest, HttpResponseRedirect, JsonResponse,
Http404
from .tasks import check_uniqueness
from .services.TextService import *


def home(request):
    """Renders the home page."""
    assert isinstance(request, HttpRequest)
    return render(request,
                  'app/home_page.html',
                  )


def selection(request):
    """Renders the text selection page."""
    assert isinstance(request, HttpRequest)
    return render(request,
                  'app/selection_page.html',
                  )


def texts(request):
    """Renders the text list page."""
    assert isinstance(request, HttpRequest)
```

```
    all_text = Text.objects.all()
    processing_texts = len(all_text.filter(uniqueness=-1.0))
    web_texts = len(all_text.filter(uniqueness=101))
    return render(request,
                  'app/text_catalog.html',
                  {'all_sources': all_text,
                   'processing_texts': processing_texts,
                   'web_texts': web_texts, }
                  )


def account_render(request):
    """Renders the account page."""
    assert isinstance(request, HttpRequest)
    user_texts = Text.objects.filter(author=request.user.username)
    processing_texts = len(user_texts.filter(uniqueness=-1.0))
    return render(request,
                  'app/account.html',
                  {'all_sources': user_texts,
                   'processing_texts': processing_texts, },
                  )


def text_details(request, pk):
    """Renders the text details page."""
    assert isinstance(request, HttpRequest)
    text = Text.objects.filter(id=pk)
    return render(request,
                  'app/text_details.html',
                  {'text': text},
                  )
```

```
        def add_text(request):
            author = "Unknown"
            content = request.POST.get("Content")
            if request.user.is_authenticated:
                author = request.user.username
            text_id = CreateText(content, author)
            check_uniqueness.delay(text_id)
            return HttpResponseRedirect("/text/" + str(text_id) + '/')


        def select_texts(request):
            uniqueness_upper_border =
float(request.POST.get("uniqueness_upper_border")),
            uniqueness_upper_border = uniqueness_upper_border[0]
            uniqueness_down_border =
float(request.POST.get("uniqueness_down_border")),
            uniqueness_down_border = uniqueness_down_border[0]
            if request.user.is_staff:
                author = request.POST.get("author_texts")
            else:
                author = request.user.username
                if not (0.0 <= uniqueness_upper_border <= 100.0):
                    uniqueness_upper_border = 100.0
                if not (0.0 <= uniqueness_down_border <= 100.0):
                    uniqueness_down_border = 0.0
            Selection = SelectionRequest(author_filter=author,
                                         key_words_filter=request.POST.get("source_filters").split('
'),
                                         uniqueness_upper_border=uniqueness_upper_border,
```

```
                        uniqueness_down_border=uniqueness_down_border,
                        left_date=request.POST.get("left_date"),
                        right_date=request.POST.get("right_date")
                        )

        success_uniqueness_test_border =
float(request.POST.get("uniqueness_success_border"))
        selection_set = Selection.GetSelectionSet()
        set_length = len(selection_set)
        return render(request,
                    'app/selection_set.html',
                    {'SelectionSet': selection_set,
                     'success_uniqueness_test_border': success_uniqueness_test_border,
                     'set_length': set_length},
                    )


    def validate_username(request):
        """Answers ajax request check free username."""
        username = request.GET.get('username', None)
        data = {
            'is_taken': User.objects.filter(username__iexact=username).exists()
        }
        return JsonResponse(data)


    def text_source(request, first_text_id, second_text_id):
        """Answers ajax request to get text content."""
        if request.is_ajax():
            text_content = GetTextContent(second_text_id)
```

```
            text_burrowed_content = GetTextBurrowedContent(first_text_id,
second_text_id)
            response = {'text': text_content.rstrip('\n'),
                    'burrowed_content': text_burrowed_content}
            return JsonResponse(response)
        else:
            raise Http404


    def highlight_text(request, pk):
        """Answers ajax request to highlight text parts."""
        if request.is_ajax():
            text_burrowed_content = GetTextBurrowedContent(pk)
            # print(text_burrowed_content)
            if text_burrowed_content:
                text_burrowed_content_sources = list(text_burrowed_content.keys())
                response = {'text': text_burrowed_content,
                        'sources': text_burrowed_content_sources}
            else:
                response = None
            return JsonResponse(response, safe=False)
        else:
            raise Http404
```