ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ ТЕМАТИЧНОГО СЛОВНИКА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ "РОБОТОТЕХНІКА" МОВОЮ PYTHON**»

Виконав: студент 4 курсу, групи ПД–41
спеціальності
121 _Інженерія програмного забезпечення_
(шифр і назва спеціальності)
_____Лукашенко І. П._____
(прізвище та ініціали)
Керівник __Золотухіна О. А._____
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення_____
Ступінь вищої освіти - «Бакалавр»_____
Напрям підготовки - 121 «Інженерія програмного забезпечення»_____

ЗАТВЕРДЖУЮ
Завідувач кафедри
Телекомунікаційних технологій
_____Негоденко О. В.
«____» _____2023 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Лукашенка Ігоря Петровича

(прізвище, ім‘я, по батькові)

1.Тема роботи: «Розробка програмного забезпечення для автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка" мовою Python»_____
Керівник роботи___Золотухіна О.А., к.т.н., доц., доцент кафедри ІПЗ_____,
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом вищого навчального закладу від «24» __лютого__2023 року No26.
2. Строк подання студентом роботи___«1»___червня_2023 року

3. Вхідні дані до роботи:
- 3.1Теоретичні відомості про методи обробки природньої мови.
- 3.2 Опис методів вилучення інформації з тексту та формування тематичних словників.
- 3.3Технічна документація з описом бібліотек для обробки природньої мови.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
- 4.1Огляд технологій обробки природньої мови.
- 4.2 Аналіз існуючих методів обробки природньої мови.
- 4.3 Огляд підходів до розробки додатків.

4.4 Програмна реалізація додатку для автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка" мовою Python з використанням бібліотеки NLTK.
4.5 Тестування розробленої моделі.

5. Перелік графічного матеріалу
3.1 Мета, об’єкт та предмет дослідження.
3.2 Задачі дипломної роботи.
3.3 Аналіз аналогів.
3.4 Вимоги до програмного забезпечення.
3.5 Програмні засоби реалізації.
3.6 Діаграма варіантів використання додатку.
3.7 Діаграма діяльності додатку.
3.8 Діаграма класів додатку.
3.9 Екранні форми.
3.10 Апробація результатів дослідження.
3.11 Висновки.

6. Дата видачі завдання _«25» лютого 2023 року_

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.23	виконано
2	Аналіз та дослідження існуючих аналогів	10.04.2023-17.04.2023	виконано
3	Огляд існуючих алгоритмів обробки тексту	19.04.2023-28.04.2023	виконано
4	Проектування застосунку	01.05.2023-05.05.2023	виконано
5	Програмна реалізація	06.05.2023-28.05.2023	виконано
6	Оформлення пояснювальної записки	28.05.2023- 29.05.2023	виконано
7	Вступ, висновки, реферат	20.05.2023	виконано
8	Презентація	21.05.2023	виконано
9	Попередній захист роботи	22.05.2023	виконано
10	Подання роботи в деканат	01.06.2023	виконано

Студент _____ _____Лукашенко І.П._____
(підпис) (прізвище та ініціали)
Керівник роботи _____ __Золотухіна О.А._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 60 с., 2 табл., 24 рис., 2 додатки,19 джерел.

ТЕМАТИЧНИЙ СЛОВНИК, NLP, NLTK, ТОКЕНИЗАЦІЯ, ЛЕМАТИЗАЦІЯ, КЛЮЧОВІ СЛОВА.

Об`єкт дослідження – автоматизована побудова тематичного словника навчальної дисципліни.

Предмет дослідження – програмне забезпечення мовою Python для автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка".

Мета роботи – спрощення процесу формування тематичного словника з дисципліни за рахунок ПЗ створеного мовою Python.

Використано метод автоматичного формування ключових слів. Проведено дослідження за допомогою бібліотек NLTK для обробки текстів, Tkinter для створення користувацького інтерфейсу, Matplotlib та wordcloud для візуалізації даних. У результаті роботи здійснена програмна реалізація системи у вигляді додатку для пошуку визначень з файлів за тематикою дисципліни «Робототехніка».

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

BoW – Bag of Words

КІ – користувацький інтерфейс

LDA - Latent Dirichlet Allocation

NLP- Natural Language Processing

TF-IDF – Term Frequency - Inverse Document Frequency

ВСТУП

На сьогодні тематичні словники використовується у різноманітних галузях, зокрема в обробці природної мови, машинному навчанні та аналізі даних. Вони дозволяють автоматично визначати тематику текстів, класифікувати їх за темами та категоріями, витягати ключові слова і терміни, які найбільш точно описують зміст тексту. Тематичні словники застосовуються в багатьох галузях, таких як маркетинг, фінанси, медицина, наука та технології, що дає змогу зробити висновки на основі даних і покращити роботу відповідних галузей. Крім того, тематичні словники є важливим інструментом для автоматичної обробки великої кількості даних, що стає все більш важливим у світі швидкого розвитку інформаційних технологій [7, 8, 9].

Об'єкт дослідження - автоматизована побудова тематичного словника навчальної дисципліни.

Предмет дослідження - програмне забезпечення мовою Python для автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка"

Мета дослідження - спрощення процесу формування тематичного словника з дисципліни за рахунок програмного забезпечення створеного мовою Python.

В процесі дослідження виконувались наступні завдання:

– аналіз алгоритмів та методів для обробки природньої мови;

– аналіз аналогічних засобів для формування тематичних словників;

– вибір засобів реалізації додатку для автоматизованого формування тематичного словника;

– проектування вимог додатку для автоматизованого формування тематичного словника;

– розробка додатку відповідно до створених вимог;

– тестування програмного забезпечення.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Аналіз алгоритмів та методів для обробки природньої мови

Тематичні словники - це упорядковані списки, що дозволяють збирати слова та терміни, пов'язані з конкретною темою або галуззю знань. Вони є надзвичайно корисними для тих, хто вивчає іноземну мову або працює в певній науковій, технічній чи професійній галузі. Застосування тематичних словників дозволяє ефективно збагачувати свій словниковий запас та глибше вивчати термінологію, що використовується у конкретній галузі.

Одним з основних застосувань тематичних словників є вивчення мов. Для вивчення іноземної мови необхідно знати багато слів та виразів, які пов'язані з конкретною темою, щоб бути в змозі розуміти та використовувати спеціалізовану термінологію. Тематичні словники допомагають у цьому, збираючи слова та вирази, які використовуються в певній галузі, та надаючи їх переклади. Це надає змогу студентам та вчителям ефективніше вчити та викладати спеціалізовану лексику.

Крім того, тематичні словники є надзвичайно корисними для професіоналів, які працюють у специфічній галузі, такі як медицина, право, техніка, наука та інші. У таких галузях використовується спеціалізована термінологія, яка може бути складною для розуміння для тих, хто не має необхідних знань та досвіду. Тематичні словники допомагають професіоналам в цих галузях легше зрозуміти та використовувати спеціалізовану термінологію. Тематичні словники містять описи та визначення слів та термінів, що допомагає професіоналам зрозуміти їх сенс та застосування. Крім того, тематичні словники є корисними інструментами для перекладачів, які працюють з текстами, що містять спеціалізовану лексику. Застосування тематичних словників допомагає перекладачам знайти правильний переклад слів та виразів, що використовуються в конкретній галузі.

Тематичні словники можуть бути корисними для наукових досліджень та інших академічних дисциплін. У науці використовуються спеціалізовані терміни та

поняття, які можуть бути складними для розуміння тих, хто не має достатнього досвіду в даній галузі. Тематичні словники допомагають науковцям або тим, хто лише починає свій шлях у науці зрозуміти та використовувати термінологію, що використовується у дослідженнях, та дозволяють бути більш точними у висловлюваннях.

Існує багато видів тематичних словників, які відрізняються за своїм призначенням та змістом. Наприклад, загальновживані словники, такі як тлумачні словники, містять визначення слів та їх значень, які використовуються в повсякденному житті. Спеціалізовані словники містять терміни та поняття, що використовуються в конкретній галузі, такі як медичні словники або словники з технічних наук.

Існують словники, які спеціалізуються на термінах та поняттях, які використовуються в певній культурі або регіоні, такі як етнічні словники або словники мовленнєвого етикету. Ці словники допомагають розуміти та використовувати правильну лексику та формули вітань, привітань, формальних та неформальних виразів, що використовуються в різних культурах.

Загалом, тематичні словники є важливими інструментами для всіх, хто працює з мовою, незалежно від того, чи це студент, професіонал, перекладач або науковець. Вони допомагають розширити словниковий запас та глибше зрозуміти термінологію, що використовується в конкретній галузі. Тому рекомендується використовувати тематичні словники при вивченні мови, роботі з текстами та наукових дослідженнях, що вимагають спеціалізованої лексики.

В галузі NLP тематизація тексту допомагає проаналізувати великі об’єми текстів з метою виявлення ключових слів та тематик текстів. Моделювання теми тексту відноситься до галузі машинного навчання, в якій не обов’язкові наперед сформовані масиви даних для навчання моделі. Ця техніка називається навчання без вчителя.

Передуючим етапом для будь-яких операцій з текстом є його попередня обробка. Попередня обробка тексту – це процес перетворення певного об’єму

інформації для подальшої обробки з метою отримання найточнішого результату. Це може включати наступні етапи:

– переведення всього тексту до нижнього регістра;
– вилучення спеціальних знаків та стоп слів;
– заміна скорочень слів їх повними формами;
– токенізація слів;
– лематизація слів.

Вилучення стоп-слів є важливим етапом при обробці текстів з кількох причин[10]:

– зменшення розміру даних: Стоп-слова зазвичай входять до списку дуже поширених слів, таких як "і", "та", "який", "в" тощо, які з'являються в тексті дуже часто, але не несуть корисної інформації про зміст тексту. Вилучення цих слів допомагає зменшити обсяг даних, які потрібно обробляти, і зменшує час обробки;

– покращення якості обробки: Якщо стоп-слова не видаляти, вони можуть переважати і створювати спотворення у підрахунку важливих слів. Наприклад, якщо слово "the" не видаляти, то воно буде найбільш часто зустрічатися у тексті, але не містити корисної інформації про зміст. Це може призвести до неправильного виділення ключових слів у тексті;

– покращення точності аналізу: Вилучення стоп-слів допомагає сконцентруватися на термінах, які мають велике значення для контексту. Наприклад, якщо аналізується текст з метою визначення тематики, вилучення стоп-слів допоможе визначити ключові терміни, які будуть містити більше інформації про тему.

Токенізацією є процес розділення тексту на окремі одиниці, що називають токенами. Токеном може бути слово, символ, число або будь-яка інша частина тексту, яка має семантичний зміст. Цей процес є одним з перших кроків в обробці тексту і може бути використаний для підготовки даних для подальшого аналізу, такого як побудова моделей машинного навчання або обчислення статистичних характеристик тексту. Лематизація – це процес приведення слів до її словникових форм, яка називається лемою. Лема – це слово, яке є базовою формою слова, від

якого можуть бути утворені інші слова шляхом додавання префіксів, суфіксів, закінчень і т.д. Процес лематизації включає в себе видалення закінчень та інших афіксів зі слова і зведення його до форми леми, що дозволяє зменшити кількість унікальних слів в тексті і полегшити подальший аналіз тексту.

Далі переглянемо існуючі методи та алгоритми для побудови словників.

Метод «Мішок слів»(Bag of Words). Цей метод полягає в тому, що всі слова з колекції текстів збираються в один великий словник, який використовується для подальшої обробки текстів. Для кожного тексту в колекції створюється вектор, який містить кількість входжень із словника в даний текст. Цей метод може бути використаний для створення базового словника.

Метод TF-IDF (Term Frequency-Inverse Document Frequency). Цей метод оцінює важливість слова у тексті з урахуванням його частоти в цьому тексті та частоти входжень слова в інші колекції. Частота входжень слова у тексті(TF) розраховується як кількість входжень слова в текст поділена на загальну кількість слів у тексті. Інвертована частота документу(IDF) розраховується як логарифм відношення загальної кількості текстів до кількості текстів, що містять дане слово. Загальна вага слова(TF-IDF) розраховується як добуток TF і IDF[11].

Алгоритм LDA. Цей алгоритм знаходить тематичні кластери у колекції текстів[6]. Кожен текст розглядається як сукупність тем, кожна з яких представлена деяким розподілом слів. Алгоритм шукає такі розподіли слів, які максимально відповідають розподілу тем у текстах. Результатом роботи алгоритм є набір тем та набір слів, які характеризують кожну тему.

Метод Word2Vec. Це алгоритм машинного навчання, який використовується для векторного представлення слів у колекції текстів. Кожне слово представляється у векторному просторі. Векторне представлення слів - це вектор з числовими значеннями, що відображає взаємозв'язки між словами у векторному просторі. Слова, які зустрічаються частіше разом, відображаються векторами, що знаходяться біля один одного у векторному просторі. Цей метод може бути використаний для створення словника, який враховує семантичну близькість слів[12].

Алгоритм K-Means. Цей алгоритм знаходить кластери у колекції текстів на основі відстані між векторами слів, що представляють кожен текст. Кількість кластерів визначається користувачем. Кожен кластер може бути представлений словами, які найбільше характеризують тексти, які належать до цього кластера. Цей метод може бути використаний для створення тематичного словника на основі кластеризації тексті[13].

1.2. Аналіз аналогів для формування тематичних словників

1.2.1 Sketch Engine

Один з найбільш популярних додатків для створення тематичних словників - це Sketch Engine[1]. Це веб-сервіс, який пропонує користувачам можливість створювати словники на основі текстових даних. Sketch Engine має велику кількість різних функцій, таких як автоматичне визначення стилів тексту, побудова графіків і частотних таблиць, редагування інформації в словнику та інші.

Sketch Engine є інструментом для людей що вивчають мови. З рисунка 1.1 вино наявний набір інструментів, який дозволяє, наприклад вибрати слово і додаток надає список використання цього слова у контексті відповідно, до вибраної вами мови, пошук термінів по вашому корпусу тексту, пошук частот всіх слів тексту, вилучення n-грам та ключових слів.

Додаток має превстановлені набори текстів різних мов за замовчуванням, які можна аналізувати, або можна завантажити свої. Також існує інструмент з набору Sketch Engine, який називається OneClick Terms[2] (рис. 1.2). Він дозволяє завантажити файл і отримати ключові визначення з тексту.

Після опрацювання файлу додаток виводить на екран список слів, при натисканні на які буде відображене в якому контексті вживалося слово. Даний результат можна назвати словником з перебільшенням.

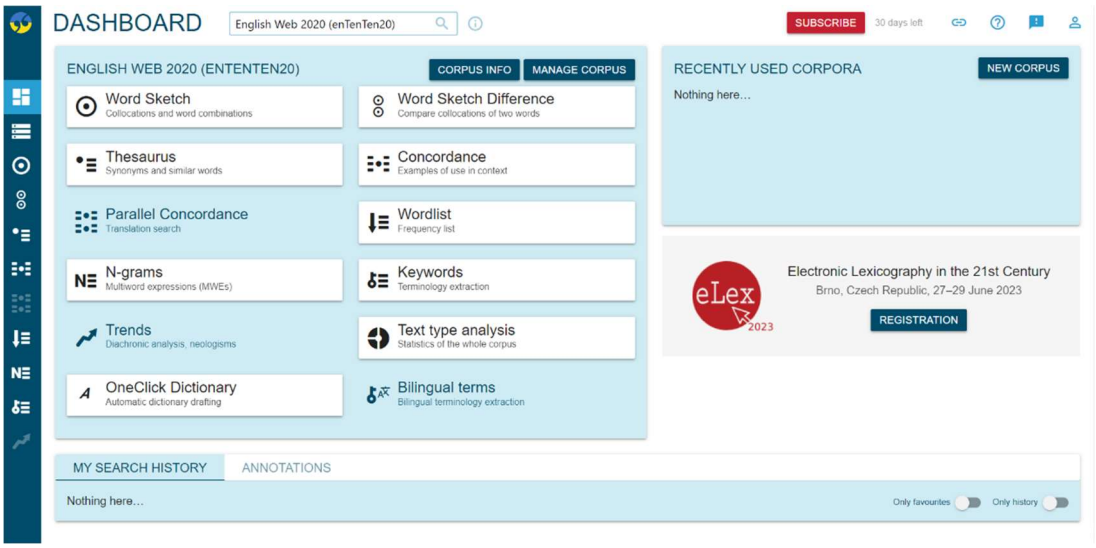


Рисунок 1.1 – Приклад екранних форм додатку Sketch Engine

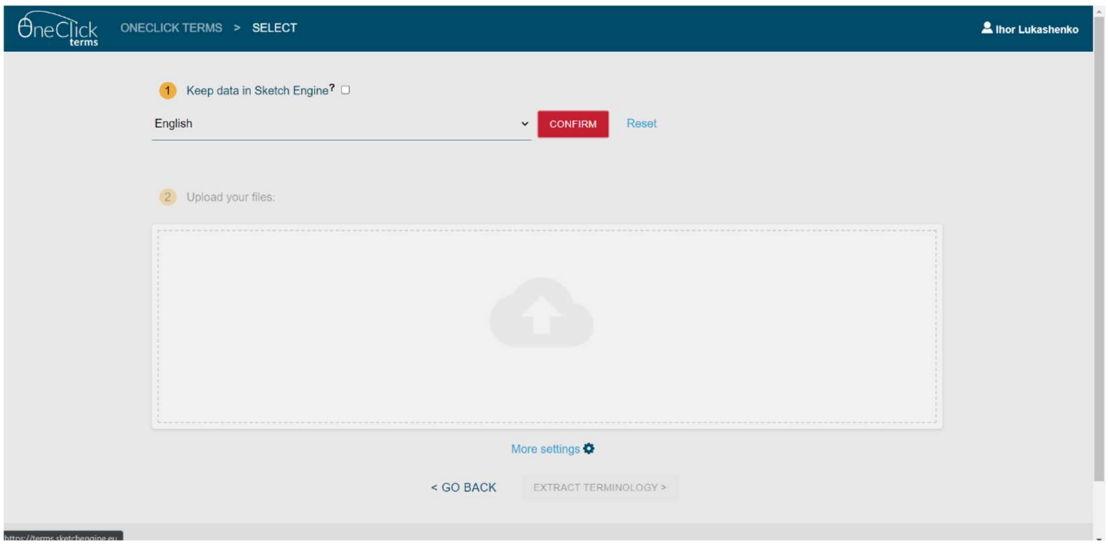


Рисунок 1.2 – Приклад екранних форм додатку Sketch Engine – OneClick Terms

Переваги:

– широкий вибір мов для вивчення використання слів;

– високоякісний аналіз тексту;

– гнучкі налаштування.

Недоліки:

– висока вартість: використання Sketch Engine може бути досить дорогим для користувачів, зокрема, для студентів та дослідників з обмеженим бюджетом;

– обмежена кількість корпусів: деякі корпуси текстів не є доступними для користувачів, що може обмежувати їхні можливості дослідження;

– вимоги до комп'ютера: Sketch Engine вимагає відносно потужного комп'ютера для ефективної роботи, що може створювати труднощі для користувачів зі слабким апаратним забезпеченням;

– неможливо автоматизувати створення тематичного словника.

1.2.2 AntConc

Ще один популярний додаток – AntConc[3]. Це безкоштовна програма з відкритим вихідним кодом, яка дозволяє створювати тематичні словники та проводити різноманітний текстовий аналіз. Як видно з рисунку 1.3, AntConc має простий інтерфейс, що дозволяє користувачам ефективно працювати з текстом, і має багато функцій, включаючи підтримку багатьох мов, функції для побудови графіків і статистики, а також можливість визначення ключових слів у тексті.

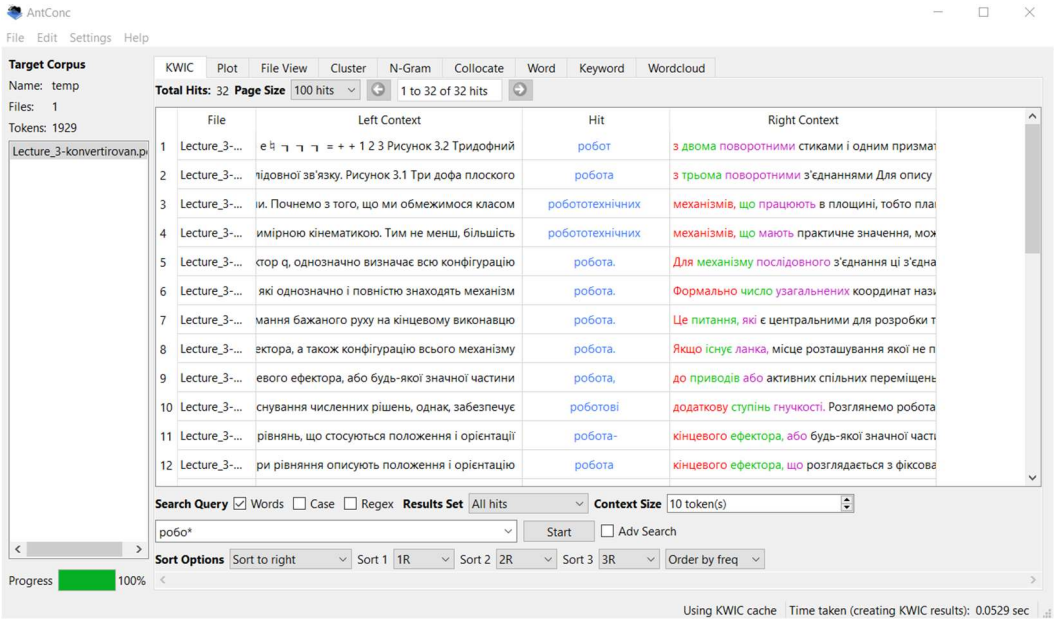


Рисунок 1.3 – Приклад екранних форм додатку AntConc

Функціоналом даного програмного забезпечення є :

– пошук слів у наборі документів з відображенням контексту та набір функцій для сортування;

– можливість графічно відобразити частоту появлення терміну в документі;

– пошук слів по документу;

– кластерний пошук слова;

– відображення і пошук n-грам;

– пошук колокацій в тексті;

– частотний аналіз слів;

– пошук ключових слів

– побудова хмари слів.

Також слід зазначити, що дане програмне забезпечення більш призначене для аналізу тематичних словників, а ніж для їх створення. Він створений для роботи з текстовими корпусами, тобто великими наборами текстів. Додаток дозволяє швидко та легко оброблювати тексти та створювати тематичні словники на основі частотного аналізу слів.

Переваги додатку:

– зберігання результатів аналізу у форматі CSV, що дозволяє легко оброблювати дані в інших програмах, таких як Microsoft Excel;

– можливість працювати з декількома файлами;

– програма має відкритий код;

– можливість виконувати аналіз мов, які використовують нелатинську абетку;

– можливість аналізувати тематичні словники.

Недоліки:

– неможливо автоматизувати створення тематичного словника;

– некоректне оброблення PDF файлів.

1.2.3 TshwaneLex

TshwaneLex[4] є програмою для створення та керування термінологічними словниками. Вона дозволяє користувачам створювати професійні термінологічні словники для будь-якої галузі знань. Програма має досить широкий функціонал,

який може задовольнити потреби більшості користувачів. Для початку роботи з програмою користувач може створити новий проект, додати терміни та їх характеристики, а також визначити різні відношення між термінами (наприклад, синонімію, антонімію, гіпонімію тощо).

Функціонал TshwaneLex включає такі можливості (рис.1.4):

– створення та редагування термінів з різними характеристиками (наприклад, визначення, контекстуальні відношення тощо);

– визначення зв'язків між термінами, таких як синонімія, антонімія, гіпонімія тощо;

– робота з багатомовними словниками, включаючи можливість визначення різних мовних варіантів для одного терміну;

– автоматичне заповнення термінів на основі джерел, таких як існуючі словники, тезауруси та інші;

– експорт термінів в різних форматах, таких як Excel, XML, HTML, RTF та інші.

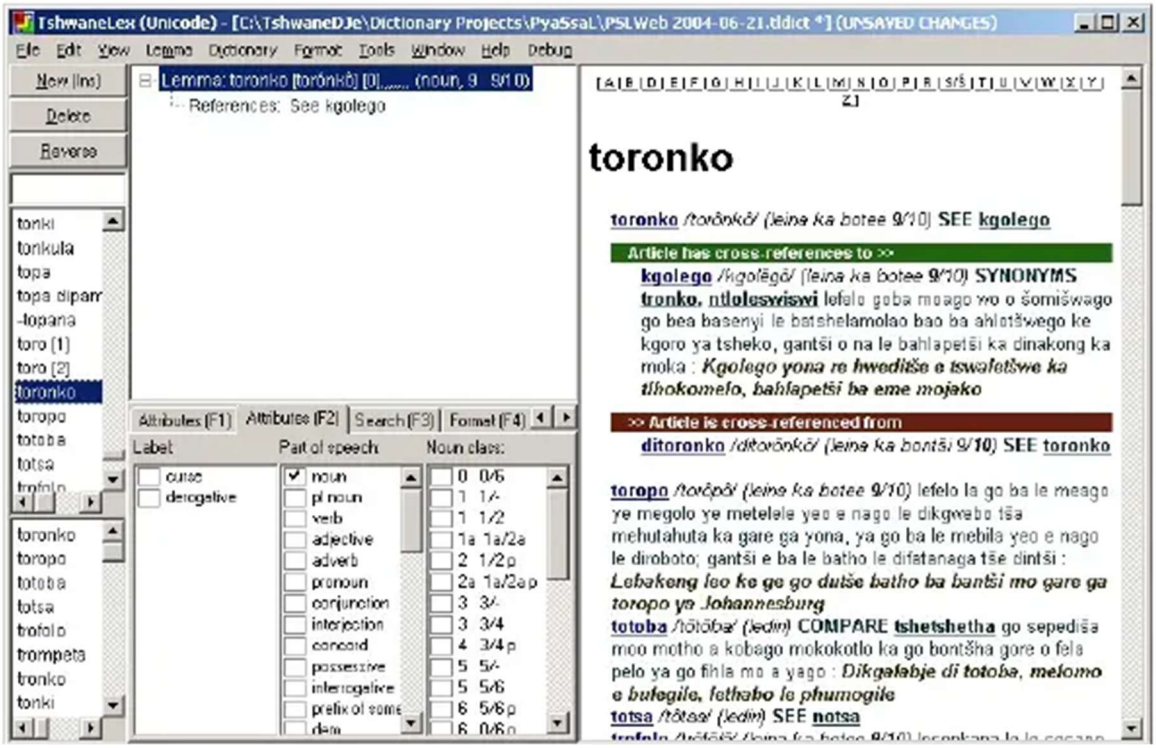


Рисунок 1.4 - Приклад екранних форм додатку TshwaneLex

Основні переваги TshwaneLex:

– розширені можливості для редагування термінів - TshwaneLex дозволяє редагувати терміни в різних форматах, включаючи фонетичні транскрипції та міжмовні відповідники;

– можливість імпортувати терміни з інших джерел - програма підтримує імпорт термінів з Excel-файлів, тезаурусів та інших термінологічних баз даних;

– функція автоматичного заповнення термінів - TshwaneLex має вбудований функціонал, що дозволяє автоматично заповнювати терміни за допомогою бази даних, тезаурусу чи іншого джерела;

– різні формати експорту - TshwaneLex підтримує експорт термінів у різних форматах, включаючи Excel, XML, HTML та інші.

Щодо недоліків TshwaneLex, можна виділити деякі проблеми з перекладом термінів, а також обмеження в роботі з багатомовними словниками.

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1 – Зведені результати аналізу характеристик додатків для автоматизованого формування словників

Показник	Sketch Engine	AntConc	TshwaneLex
Платформа	Windows, MacOS, Linux ,Web	Windows, MacOS, Linux	Windows
Підтрика корпусів тексту	+	+	+
Пошук по шаблону	+	+	+
Підтримка Unicode	+	+	+
Візуалізація даних	+	-	+
Автоматична обробка тексту	+	+	-
Підтримка машинного навчання	+	-	-

Продовження таблиці 1.1 – Зведені результати аналізу характеристик додатків для автоматизованого формування словників

Показник	Sketch Engine	AntConc	TshwaneLex
Доступ до API	+	-	-
Пошук n-грам	+	+	-
Відображення контексту слів	+	+	-

1.3. Огляд засобів реалізації додатку для автоматизованого формування тематичного словника

В даній роботі розробляється додаток для автоматизованого формування тематичного словника з дисципліни для ПК, Windows 10. Мовою програмування обрано Python. Python є однією з найбільш популярних мов програмування в світі і використовується в різних сферах, включаючи науку про дані, машинне навчання, веб-розробку та автоматизацію процесів.

Передусім, Python має велику кількість бібліотек, які забезпечують ефективну обробку текстів. Одна з найпопулярніших бібліотек, яка використовується для обробки текстів - це Natural Language Toolkit (NLTK)[5]. NLTK містить багато функцій для обробки текстів, такі як токенізація, стемінг, лематизація та тематичне моделювання, що дозволяє швидко та ефективно обробляти текстові дані.

Python також має велику кількість інших бібліотек, які можна використовувати для роботи з текстом. Наприклад, Gensim - це бібліотека, яка забезпечує функції для створення тематичних моделей, включаючи моделі LSI (Latent Semantic Indexing), LDA (Latent Dirichlet Allocation) та інші. Крім того, є бібліотеки, які дозволяють виконувати інші операції з текстом, такі як зведення тексту до нижнього регістру, вилучення стоп-слів та інші.

Python має простий та зрозумілий синтаксис, що дозволяє швидко розробляти та тестувати код. Це дозволяє швидко та ефективно розробляти скрипти, які

автоматизують процеси побудови тематичних словників. Крім того, Python має велику кількість інструментів для розробки, таких як Jupyter Notebook, PyCharm та інші, які дозволяють легко розробити код та проводити експерименти з тематичним моделюванням.

Ще однією перевагою Python є його кросплатформність. Це означає, що код, написаний на Python, може працювати на різних операційних системах, таких як Windows, MacOS та Linux. Це дає можливість розробляти та запускати скрипти на різних пристроях та у різних середовищах.

Крім того, Python має активну спільноту розробників, яка надає безкоштовну підтримку та розвиток бібліотек для обробки текстів. Це означає, що користувачі можуть швидко отримати відповіді на свої запитання та проблеми, а також взяти участь у розвитку бібліотек та інструментів для обробки текстів.

Для створення зручного інтерфейсу користувача використовується бібліотека Tkinter, яка є стандартною бібліотекою Python для розробки графічних інтерфейсів. Tkinter надає можливість створювати вікна, кнопки, тексти та інші елементи інтерфейсу для взаємодії з користувачем. Це дозволяє зробити програму більш доступною та зручною у використанні.

2 МАТЕМАТИЧНА МОДЕЛЬ ДЛЯ ОБРОБКИ ТЕКСТУ

2.1. Токенизація

Токенизація є першим кроком у математичній моделі обробки тексту. Вона виконується з метою розбиття вхідного тексту на окремі частини, які називаються токенами. Токени можуть бути різними одиницями тексту, такими як слова, речення, символи або навіть більш специфічні одиниці, в залежності від потреби аналізу. Вона дозволяє розбити текст на більш маневрені компоненти, що полегшує подальший аналіз та обробку. Наприклад, розбиття тексту на окремі слова допомагає встановити частоту вживання певних слів, виявити ключові терміни або виконати подальші мовні операції.

Токенизація впливає на текст шляхом поділу його на менші частини, що називаються токенами. Кожен токен може мати своє значення і роль в контексті тексту. Наприклад, розбиття речення на окремі слова дозволяє аналізувати семантику та структуру тексту.

Процес розбиття тексту на окремі частини може включати в себе видалення розділових знаків, символів пунктуації та спеціальних символів, а також розділення слів за допомогою пробілів або інших правил. Наприклад, речення "Сьогодні гарний день!" може бути розбите на окремі токени, такі як ["Сьогодні", "гарний", "день", "!"]. Токенізація може варіюватися залежно від потреб і специфіки текстових даних. Вона може враховувати особливості мови, контексту або використовувати машинне навчання для кращої адаптації до конкретних завдань обробки тексту.

Після токенізації тексту отримуємо послідовність окремих токенів, яку можна використовувати для подальшого аналізу, лематизації, усунення стоп-слів та інших операцій обробки тексту. Важливо враховувати контекст та мету аналізу тексту при виборі підходу до токенізації, оскільки різні типи токенів можуть бути корисними в різних аспектах обробки тексту.

2.2. Лематизація

Лематизація є процесом нормалізації тексту, за допомогою якого зводяться слова до їхньої базової форми, яка називається лемою. Лема є словом-представником для групи спільної лексичної основи. Наприклад, лематизація приводить слова "бігати", "біжить", "біжу" до їхньої базової форми "бігти".

Лематизація спрощує аналіз тексту, оскільки знижує розмірність словнику і дозволяє зосередитись на суттєвих аспектах тексту. Вона допомагає виявляти семантичні зв'язки між словами, розуміти контекст та покращувати результати подальших завдань, таких як класифікація, пошук або аналіз настрою..

У процесі використовуються морфологічні правила, словники або машинні моделі, які співставляють слова з їхніми базовими формами. Це може включати в себе розпізнавання частин мови, зміну закінчень, афіксів та інші морфологічні перетворення.

Один з популярних підходів до лематизації використовує морфологічний аналіз. Він враховує граматичні правила та морфемну структуру слів для знаходження їх базової форми. Наприклад, для англійської мови такий підхід може використовувати правила про суфікси, закінчення та інші морфеми, щоб зводити слова до їх основи.

Лематизація є важливим кроком для подальшого аналізу тексту, такого як класифікація, пошук інформації або витягування ключових слів, оскільки дозволяє зводити різні форми слова до спільного представлення.

2.3. Усунення стоп-слів

У математичній моделі обробки тексту стоп-слова є списком слів, які зазвичай не несуть значущої інформації та можуть бути виключені з аналізу. Ці слова включають загальнопоширені частки мови, прийменники, сполучники, займенники та інші слова, які часто зустрічаються, але не мають вагомого семантичного значення.

Вплив стоп-слів на текстовий аналіз полягає в їх виключенні з обробки, що дозволяє зосередитися на більш значущих словах та збільшити точність та ефективність алгоритмів обробки тексту. Наприклад, виключення стоп-слів може допомогти уникнути перевантаження системи при обробці великого обсягу тексту або зосередитися на ключових аспектах тексту [14].

Існує декілька математичних моделей, пов'язаних з роботою зі стоп-словами. Одна з них - це побудова і використання словників стоп-слів, де перелічені слова, які слід виключити з обробки. Це може бути простий список слів або більш складніша структура даних, наприклад, хеш-таблиця або дерево. Інша математична модель - це використання статистичного аналізу для визначення частотності слів та виключення тих, що зустрічаються надто часто. Наприклад, можна використовувати методи, такі як частотний аналіз чи TF-IDF (Term Frequency-Inverse Document Frequency), щоб виявити слова, які є загальними та неінформативними для конкретного текстового корпусу[15]. Статистика TF-IDF для і-го терміну в j-му документі зазвичай обчислюється таким чином:

$$t_{i,j} = TF - IDF = TF * IDF, \tag{2.1}$$

де

$$TF_{(i,j)} = \frac{\text{скільки разів термін } i \text{ зустрічається в документі } j}{\text{загальна к-сть термінів у документі } j}, \tag{2.2}$$

$$IDF_{(i)} = \frac{\text{загальна к-сть документів}}{\text{к-сть документів,що містить термін } i}. \tag{2.3}$$

2.4. Обчислення частоти та ваги слів

У математичній моделі обробки тексту обчислення частоти та ваги слів грає важливу роль. Цей пункт відноситься до статистичного аналізу тексту і дозволяє визначити, наскільки важливими є окремі слова в контексті документа або колекції документів.

Частота слів визначається шляхом підрахунку кількості входжень кожного слова у текст. Це може бути корисно для виявлення найбільш релевантних слів у тексті, які можуть вказувати на ключову тематику або інформацію.

Вага слів може бути обчислена за допомогою різних методів, таких як TF-IDF (2.1). TF-IDF враховує як частоту слова в документі (TF), так і зважує її значимість в контексті всієї колекції документів (IDF). Це дозволяє виділити слова, які є важливими для певного документа, але не дуже поширеними в загальному контексті. Наприклад, якщо слово "комп'ютер" зустрічається декілька разів у тексті і в інших документах, воно може мати низьку вагу, оскільки воно є загальним терміном. Але якщо слово "криптовалюта" з'являється лише у конкретному документі, воно може мати високу вагу, оскільки воно вказує на унікальну тему або контекст.

Крім TF-IDF, існує багато інших моделей та методів для обчислення ваги слів, наприклад LSA (Latent Semantic Analysis), LDA (Latent Dirichlet Allocation) та Word2Vec, які згадувалися попередньо. Ці моделі базуються на математичних алгоритмах, які враховують структуру тексту та відносини між словами, щоб надати їм вагу та визначити їх семантичну значущість.

Обчислення частоти та ваги слів допомагає виявити ключові терміни, теми та семантику тексту. Це може бути корисно для класифікації тексту, кластеризації документів, рекомендаційних систем та багатьох інших задач обробки тексту.

2.5. Вибір ключових слів

Вибір ключових слів є важливим етапом в обробці тексту, оскільки це допомагає визначити найбільш значущі терміни та поняття, які відображають сутність тексту. Цей пункт включає критерії вибору ключових слів та можливі математичні моделі, які можуть бути використані для цього.

Критерії вибору ключових слів:

Частота: Слова, які зустрічаються найчастіше у тексті, можуть вважатися ключовими. Чим частіше слово зустрічається, тим більша ймовірність, що воно має

суттєвий вплив на зміст тексту.

Значущість: Вага слова визначається на основі його значущості у контексті. Можуть використовуватися різні методи для обчислення ваги, такі як TF-IDF (Term Frequency-Inverse Document Frequency) або BM25 (Best Match 25), які враховують як частоту слова, так і його розповсюдження по текстах.

BM25 - це метод обчислення ваги слова в документі для інформаційного пошуку. Він базується на статистичній моделі, яка враховує частоту слова в документі та в колекції документів, а також довжину документу та середню довжину документів в колекції. BM25 належить до класу функцій ранжування, які використовуються для оцінки релевантності документа до запиту користувача 1 [7].

Семантична зв'язаність: слова, які мають семантичний зв'язок з темою або контекстом тексту, можуть бути вважатися ключовими. Наприклад, у тексті про медицину ключовими словами можуть бути "лікар", "хвороба", "ліки" та інші слова, пов'язані з медичною термінологією.

TextRank - це алгоритм, який використовує графову модель для визначення важливості термінів у тексті. Він враховує зв'язки між словами, аналізуючи структуру тексту. TextRank використовує подібність між словами для визначення їх важливості[16]. Формула TextRank може мати наступний вигляд:

$$Score(w) = (1 - d) + d * sum\left(\frac{Score(v)}{OutDegree}\right), \quad (2.4)$$

де w – слово;

Score(w) - його вага;

d - демпфуючий фактор;

v - слово, яке має посилання на слово w;

Score(v) - його вага;

OutDegree(v) - кількість посилань, що виходять зі слова v.

Ці моделі допомагають вибрати ключові слова, враховуючи їх частоту, вагу та семантичний контекст. Результатом вибору ключових слів є підкреслення основних

тематик та ідей, що містяться в тексті, що сприяє подальшому аналізу та розумінню змісту.

2.6. Візуалізація результатів

Основні методи візуалізації результатів обробки тексту включають нижченаведені.

Графіки: Графіки можуть використовуватися для візуалізації частоти або ваги слів. Наприклад, можна побудувати стовпчикову діаграму, де по осі X будуть відображені ключові слова, а по осі Y - їх відповідна частота або вага. Це дозволяє порівняти значення термінів та виділити найбільш важливі.

Хмара слів: Хмара слів (word cloud) (рис. 2.1) представляє сукупність ключових слів, де розмір та положення слова в хмарі відображають його вагу чи частоту. Найбільш значущі слова відображаються більшим розміром або більшою вагою, що дозволяє візуально виділити найважливіші терміни.

Рисунок 2.1 – Хмара слів у додатку AntConc

Діаграми: Діаграми можуть бути використані для візуалізації тематик або зв'язків між ключовими словами. Наприклад, можна побудувати дерево тем, де

головні гілки відображають основні теми, а підгілки - підтеми або конкретні ключові слова. Це допомагає структурувати і візуально відобразити тематичну ієрархію тексту.

Мережі термінів: Мережі термінів (term networks) є графовою візуалізацією, де кожен термін представляється як вузол, а зв'язки між ними - як ребра графа. Це дозволяє візуально виявити взаємозв'язки між термінами та їх важливість у тексті.

Щодо математичних моделей, пов'язаних з візуалізацією результатів обробки тексту, можуть бути використані моделі візуалізації графів, такі як:

– Force-Directed Layout: Це модель, яка використовує пружинну систему для розміщення вузлів графа на площині. Вона базується на принципах фізики, де вузли взаємодіють за допомогою пружних сил і відштовхуються один від одного за допомогою відштовхувальних сил. Ця модель дозволяє створити графічне представлення мережі термінів або зв'язків між словами[17].

– Fruchterman-Reingold Algorithm: Це алгоритм для розташування вузлів графа на площині. Він використовує принцип відштовхування та притягування між вузлами, щоб розподілити їх рівномірно. Цей алгоритм може бути використаний для візуалізації мережі термінів або ієрархічної структури тем[18].

– Hierarchical Edge Bundling: Ця модель використовується для візуалізації зв'язків між ключовими словами. Вона групує зв'язки у пучки, які представляють зв'язки між темами або термінами. Це допомагає зрозуміти взаємозв'язки та структуру тексту[19].

Ці математичні моделі та методи допомагають візуально представити результати обробки тексту та сприяють легкому сприйняттю та аналізу текстової інформації

3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ ТЕМАТИЧНИХ СЛОВНИКІВ

3.1. Етапи розробки системи

Перший крок полягав у зборі вимог до програми. Це включало аналіз потреб користувачів та визначення основної функціональності програми. Завдання цього етапу полягало у встановленні чіткого розуміння того, що очікується від програми та які можливості вона повинна надавати.

Наступним кроком було розроблення архітектури програми. На цьому етапі було визначено, які компоненти будуть входити в програму та як вони будуть взаємодіяти між собою. Було створено багатошарову архітектуру, яка дозволяє логічно розділити функціональність програми на окремі компоненти, що спрощує розробку та підтримку.

На етапі реалізації компонентів було реалізовано кожен компонент програми. Кожен компонент виконує певні завдання та має внутрішню логіку обробки даних. Для розробки було використано мову програмування Python та відповідні бібліотеки та модулі.

Після реалізації кожного компонента було виконано їх інтеграцію. Це означає поєднання окремих компонентів в єдину програму та забезпечення їх взаємодії. На цьому етапі було перевірено, чи працюють всі компоненти правильно та чи передають вони дані між собою.

Після інтеграції компонентів було виконано тестування програми. Це включало виявлення та виправлення помилок, а також піддавання програми налагодженню для досягнення оптимальної продуктивності та точності. Тестування проводилося на різних вхідних даних та варіантах використання програми.

3.2. Моделювання вимог до програмного забезпечення

Для моделювання вимог додатку для автоматизованого формування тематичних словників було використано засоби мови UML. На рис. 3.1 наведено діаграму прецедентів, яка описує основні можливості, доступні користувачеві системи.

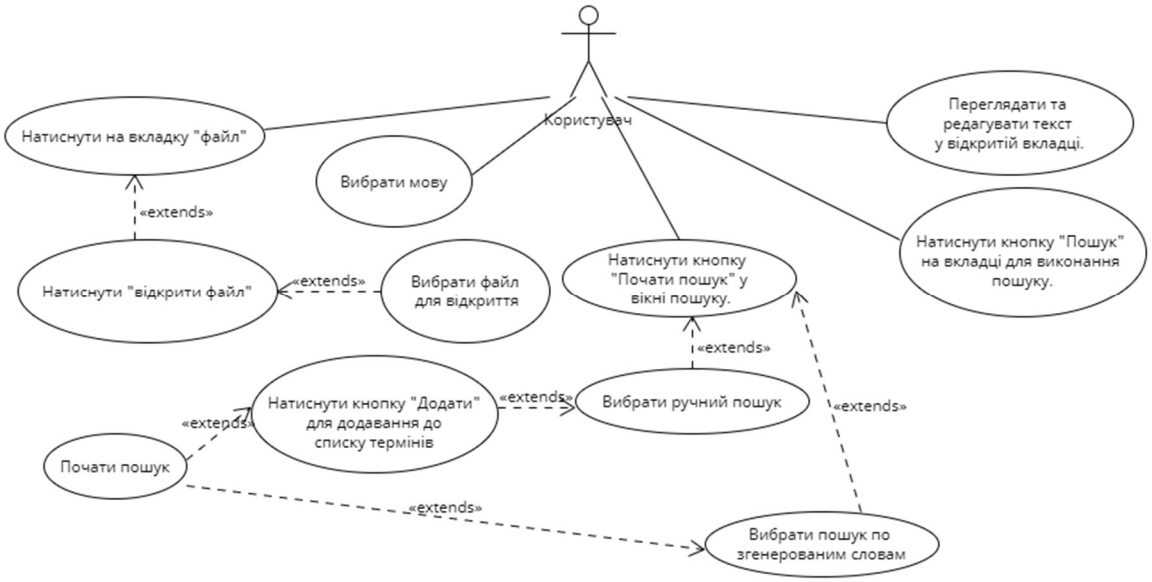


Рисунок 3.1 – Діаграма варіантів використання додатку

Основні дії, доступні користувачеві системи, включають нижченаведені:

– завантаження текстових файлів: користувач може завантажити текстові файли, які він бажає обробити та проаналізувати;

– редагування завантажених даних;

– ручний пошук: користувач може сам задати терміни які він шукає в тексті і програма виведе результат на екран;

– автоматичний пошук: програма автоматично аналізує оброблений текст та формує результати на основі виявлених ключових слів. це допомагає здійснити швидкий та об'єктивний аналіз тексту;

– візуалізація результатів: програма надає зручні засоби візуалізації результатів аналізу у вигляді хмар слів. це допомагає користувачеві зрозуміти та інтерпретувати отримані результати.

3.3. Архітектура додатку

Програма для автоматизованого формування тематичних словників має багатошарову архітектуру, яка складається з компонентів, описаних в цьому пункті.

3.3.1 Користувацький інтерфейс

Цей компонент забезпечує взаємодію користувача з програмою. Він містить графічний інтерфейс, що дозволяє користувачеві завантажувати текстові файли, налаштовувати параметри обробки тексту та переглядати результати.

Користувацький інтерфейс (КІ) є важливою частиною програмного забезпечення для автоматизованого формування тематичних словників. Він забезпечує зручний спосіб взаємодії користувача з програмою, надаючи можливість вводу даних, налаштування параметрів, перегляду результатів та керування програмою. Детальніше розглянемо основні аспекти користувацького інтерфейсу:

– Візуальний дизайн. Візуальний дизайн інтерфейсу включає в себе компоненти, кольори, шрифти, макети тощо, які сприяють зручності використання програми. Важливо створити зрозумілий та естетичний дизайн, який відповідає потребам користувача та покращує його досвід взаємодії з програмою.

– Навігація. Навігація в КІ визначає способи переміщення користувача між різними екранами, меню, кнопками та іншими елементами інтерфейсу. Потрібно забезпечити логічну та інтуїтивно зрозумілу навігацію, щоб користувач легко знаходив потрібні функції та може легко переходити між різними етапами роботи з програмою.

– Введення даних. Користувацький інтерфейс повинен надати можливість введення даних користувачем. Це може бути вибір файлів для обробки, встановлення параметрів обробки тексту, вибір методів аналізу тощо. Важливо забезпечити зручність та простоту процесу введення даних, наприклад, через текстові поля, випадаючі списки або чекбокси.

– Перегляд результатів. Користувацький інтерфейс повинен дозволяти користувачеві зручно переглядати результати аналізу. Це може бути відображення

тематичних словників, графіків, діаграм, хмар слів тощо. Важливо забезпечити зрозумілість та зручність у сприйнятті цих результатів, можливість фільтрування, сортування та деталізації інформації. На рисунку 3.2 продемонстровано вікно екранних форм на якому виведений результат формування хмари слів.

Рисунок 3.2 – Вікно виводу хмари слів

– Керування програмою. Користувацький інтерфейс повинен надати можливість керування програмою, наприклад, кнопками запуску аналізу, збереження результатів, скасування операцій та іншими функціями керування. Важливо забезпечити зрозумілість та доступність цих функцій, щоб користувач міг легко керувати роботою програми. На рисунку 3.3 наведено екранні форми, на яких зображено кнопки запуску аналізу, вводу термінів для пошуку, вибору режиму пошуку та результати пошуку.

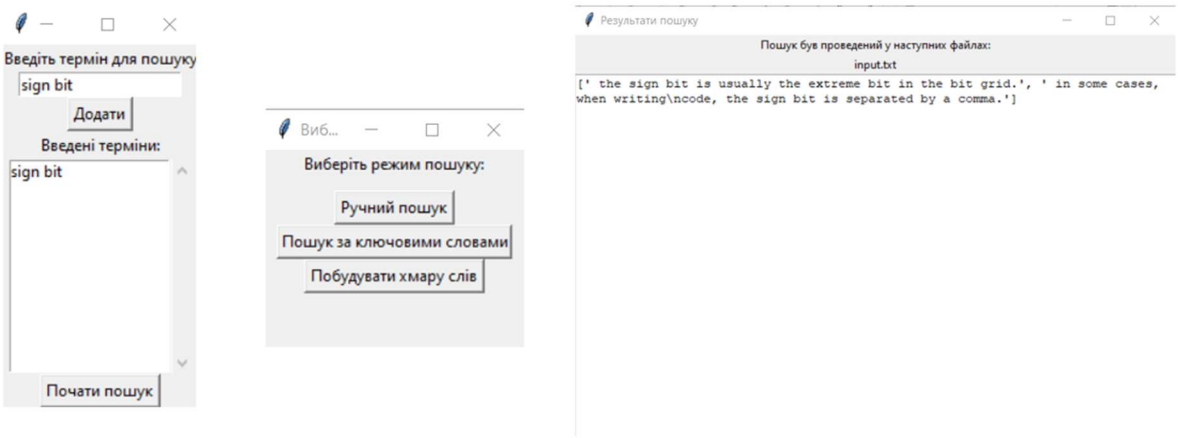


Рисунок 3.3 – Екранні форми додатку

3.3.2 Модуль обробки тексту

Цей компонент відповідає за обробку вхідного тексту. Він включає такі етапи, як токенізація, лематизація, усунення стоп-слів, обчислення частоти та ваги слів та вибір ключових слів. Кожен етап виконується послідовно з використанням відповідних алгоритмів та методів.

1. Визначення мови: При запуску програми обирається мова (англійська або українська). За замовчуванням обрана англійська мова. Вибрана мова зберігається у змінній self.language, яка містить об'єкт tkinter.StringVar(). Якщо мова змінюється, викликається функція change_language(), яка виводить обрану мову у консоль.

2. Відкриття файлу: При виборі пункту меню "Open" викликається функція open_file(). В цій функції відкривається вікно вибору файлу за допомогою filedialog.askopenfilename(). Після вибору файлу, залежно від його розширення (.txt або .pdf), виконуються наступні дії:
Якщо обраний файл є текстовим (.txt), відбувається зчитування вмісту файлу за допомогою open(file_path, "r", encoding="utf-8") і збереження вмісту в змінну self.content. Якщо обраний файл є PDF (.pdf), відбувається зчитування вмісту кожної сторінки PDF-файлу за допомогою бібліотеки PyPDF2 і збереження вмісту в змінну self.content. Потім створюється вкладка в ttk.Notebook з назвою обраного файлу, додається текстове поле tk.Text для відображення вмісту файлу. Також додаються кнопки для закриття вкладки і запуску пошуку.

3. Ручний пошук. Ручний пошук у програмі надає можливість користувачеві вручну вводити критерії пошуку для знаходження певного тексту у відкритому файлі. При натисканні кнопки "Пошук" у відкритій вкладці, викликається функція open_search_window(). Ця функція створює окреме вікно, де користувач може ввести критерії пошуку, наприклад, ключове слово або фразу, яку він бажає знайти. Після введення критеріїв пошуку, користувач натискає кнопку "Пошук", що викликає функцію manual_search(). У цій функції відбувається пошук тексту у відкритому файлі згідно з введеними критеріями. Для цього використовується метод пошуку тексту в tk.Text полі, який знаходить перше входження введеної фрази або слова. Якщо збіг знайдений, поле прокручується до відповідного рядка та виділяється вказане слово або фраза.
Цей пункт функціональності дозволяє користувачеві швидко знаходити певний текст у великому обсязі документа, що спрощує навігацію та полегшує пошук необхідної інформації. Завдяки можливості вводити власні критерії пошуку, програма надає гнучкість і зручність при ручному пошуку у відкритих текстових документах.

4. Пошук по згенерованим словам: Крім ручного пошуку програма також надає можливість виконувати пошук по згенерованим словам. Після натискання кнопки пошуку, користувач може натиснути кнопку "Пошук за ключовими словами", що викликає функцію generate_words(). Ця функція аналізує текст та виділяє окремі слова з нього. Виділені слова зберігаються у змінній self.generated_words. Після викликається функція search_generated_words(). У функції search_generated_words() відбувається порівняння згенерованих слів зі словами в текстових вкладках. Якщо знайдено збіг, відповідна вкладка відкривається та відповідне слово підсвічується у текстовому полі. Таким чином, користувач може швидко знайти всі входження згенерованого слова у відкритому тексті.

3.3.3 Аналіз тексту

Цей компонент відповідає за аналіз оброблених даних та вибір ключових слів. Він використовує математичні моделі та алгоритми, що базуються на частоті та вазі слів, для визначення найбільш важливих слів в тексті.

У функції start_manual_search_english (для англійської мови) або start_manual_search_ukrainian (для української мови) виконується пошук заданих користувачем термінів у тексті. Вона отримує список термінів зі списку terms, який містить значення введені користувачем у вікні пошуку.

Функція search_files_english (для англійської мови) або search_files_ukrainian (для української мови) виконує пошук заданих термінів у вмісті тексту self.content. Вона повертає словник з результатами пошуку, де res["results"] містить список речень, які містять знайдені терміни, а res["files"] містить список файлів, у яких були знайдені результати.

Функція display_search_results відображає результати пошуку, виводячи їх на екран. Вона отримує словник з результатами пошуку і виконує відповідне форматування та виведення результатів.

Таким чином, аналіз результатів полягає у пошуку термінів у тексті, виділенні речень, що містять ці терміни, та виведенні результатів на екран.

3.3.4 Візуалізація результатів

Цей компонент відповідає за візуалізацію оброблених даних та результатів аналізу. Він надає можливість користувачеві переглядати текст файлу до обробки, маючи можливість змінити його, та відображає результати пошуку, що допомагає зрозуміти структуру та важливість тексту.

Результати пошуку візуалізуються у вікні результатів (result_window) або у вікні попереднього перегляду файлу (file_preview_window). У вікні результатів користувач може переглянути знайдені речення та назви файлів, а також виконати додаткові дії, такі як збереження результатів у файлі або відкриття вибраного файлу. У вікні попереднього перегляду файлу користувач може відобразити знайдені збіги у контексті оригінального тексту для більш детального аналізу.

3.4 Розробка класів додатку

Для реалізації описаної вище архітектури системи було спроектовано та розроблено класи, структура та зв’язки яких наведені на рис. 3.4.

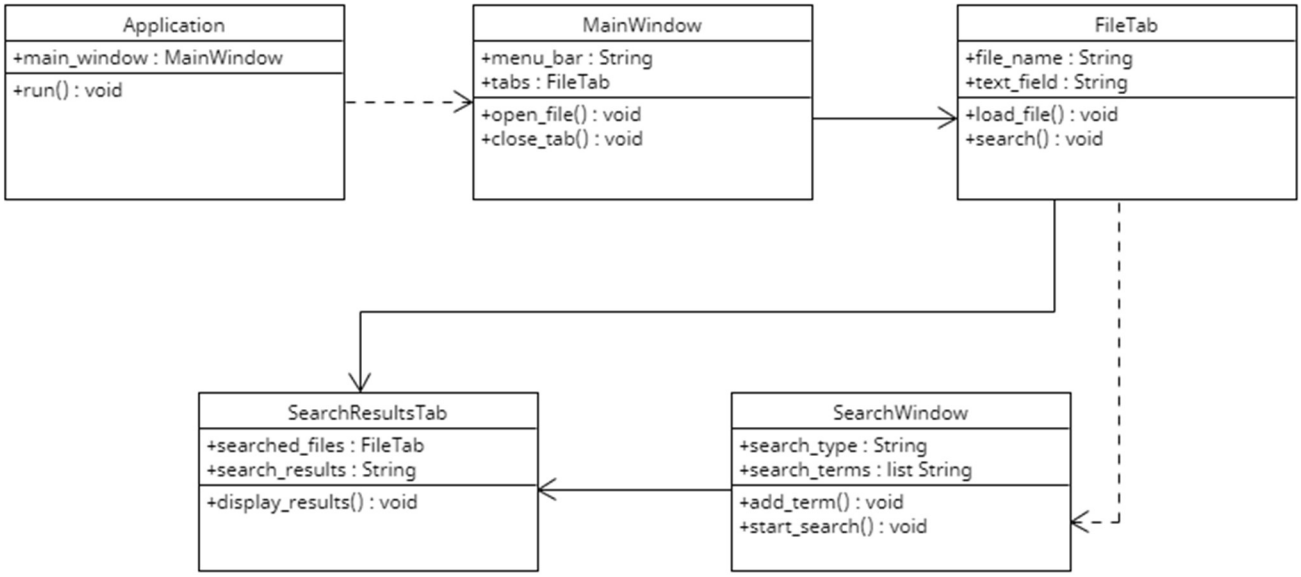


Рисунок 3.4 – Діаграма класів

Application (Додаток): Головний клас додатку, відповідає за запуск та керування графічним інтерфейсом. Атрибути: main_window - посилання на об'єкт MainWindow головного вікна. Методи: run() - запускає додаток і ініціалізує головне вікно.

MainWindow (Головне вікно): Клас, який представляє головне вікно додатку. Містить стрічку меню та вкладки. Атрибути: menu_bar - стрічка меню, tabs - список вкладок (FileTab). Методи: open_file() – відкриває діалогове вікно для вибору файлу та додає нову вкладку, close_tab(tab) - закриває вкладку tab.

FileTab (Вкладка файлу): Клас, який представляє окрему вкладку з завантаженим файлом. Містить поле для відображення тексту файлу та кнопку "Пошук". Атрибути: file_name - назва відкритого файлу, text_field - поле для відображення тексту файлу. Методи: load_file(file_path) - завантажує текст з файлу і відображає його у вкладці, search() - виконує пошук у відкритому файлі.

SearchWindow (Вікно пошуку): Клас, який представляє вікно для вибору типу пошуку та введення пошукових термінів. Містить поле для введення терміну, кнопку "Додати" та кнопку "Почати пошук". Атрибути: search_type - тип пошуку ("Ручний пошук" або "Пошук по згенерованим словам"), search_terms - список пошукових термінів. Методи: add_term(term) - додає термін до списку пошукових термінів, start_search() - починає виконання пошуку залежно від обраного типу пошуку.

SearchResultsTab (Вкладка з результатами пошуку): Клас, який представляє окрему вкладку з результатами пошуку. Містить поле для відображення файлів, в яких проводився пошук, та текстове поле з результатами пошуку. Атрибути: searched_files - список файлів, в яких проводився пошук, search_results - результати пошуку. Методи: display_results() - відображає результати пошуку у вкладці.

3.5 Опис функціонування системи

Порядок формування тематичного словника визначається діаграмою діяльності, представленою на рис.3.5.

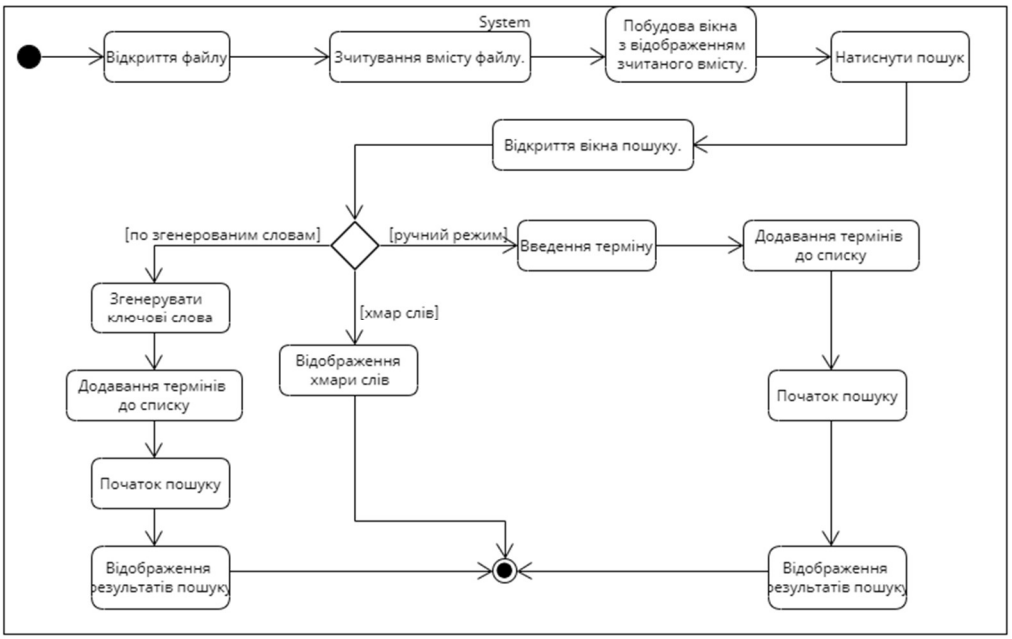


Рисунок 3.5 – Діаграма діяльності додатку

Спочатку користувач відкриває файл, який він хоче проаналізувати. Користувач обирає опцію "Відкрити файл" і вказує шлях до потрібного файлу. Програма відкриває вказаний файл для подальшої обробки. Наступним кроком програма зчитує вміст відкритого файлу і зберігає його для подальшої обробки, цей крок представлено у прецеденті «Зчитування вмісту файлу». Програма показує зчитаний вміст файлу у вікні редактора. Користувач може переглядати, редагувати та зберігати зміни – прецедент «Побудова вікна з відображенням зчитаного змісту». Далі користувач може відкрити вікно пошуку для здійснення пошуку певного тексту або виразу у відкритому файлі натиснувши кнопку пошуку. Після активації кнопки «Пошук» програма відкрити вікно пошуку для вибору режиму пошуку, в якому є три опції: «Ручний пошук», «Пошук по згенерованим словам» та «Побудова хмари слів».

Якщо користувач вибрав опцію «Ручний пошук»: з’являється вікно, в яке користувач вводить терміни або вирази, які треба знайти у відкритому файлі. Потім, ці терміни додаються програмою до списку для пошуку. Програма виконує пошук заданих термінів або виразів у вмісті відкритого файлу. Вона знаходить всі входження і показує їх користувачу.

Якщо користувач вибрав опцію «Пошук по згенерованим словам»: програма аналізує вміст відкритого файлу і виділяє найважливіші терміни та поняття. Додає ці терміни до списку пошуку і виконує пошук у документі. Знайшовши всі збіги, вона відображає результати пошуку.

Якщо користувач вибрав опцію «Побудова хмари слів»: програма аналізує вміст тексту і проводить частотний аналіз слів і записує його в список. На основі цього списку програма відображає хмару слів у вікні програми, де кожне слово представлено графічним елементом (текстовим блоком).

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування грає важливу роль у забезпеченні якості програми та виявленні помилок та неполадок перед його випуском.

4.1. Функціональне тестування

Функціональні тести є одним з видів тестів, які виконуються для перевірки функцій програми на коректність їх роботи згідно з очікуваним результатом. Ці тести дозволяють перевірити, чи відповідають реальні результати роботі програми очікуваним.

Для проведення функціональних тестів створюються тестові кейси для кожної окремої функції програми та перевіряємо, чи повертає вона правильний результат при заданих вхідних даних. Основні етапи проведення функціональних тестів включають дії наведені нижче.

Спочатку проводиться ретельний аналіз функції програми, які потрібно протестувати. Для кожної функції визначаються очікувані результати та специфікації.

Для кожної функції розробляються тестові кейси, які включають вхідні дані та очікуваний результат. Тестові кейси повинні враховувати різні можливі варіанти вхідних даних та роботи функції. Наприклад, для функції токенізації тексту можуть бути такий тестовий кейс, як зображено у таблиці 4.1.

Після створення тестових кейсів ми виконуємо тести, передаючи вхідні дані до функцій та перевіряємо отримані результати з очікуваними. Ми також забезпечуємо, що функції повертають правильні типи даних та обробляють помилки вірно.

Таблиця 4.1 – Тест-кейс для функції токенізації tokenize_text()

Test Case ID	Test Priority	Test Steps	Test Data	Expected Result
ТОК-01	Високий	1. Передати у функцію tokenize_text() вхідний текст : "Це є тестовий текст." 2. Виконати функцію tokenize_text()	"Це є тестовий текст."	['Це', 'є', 'тестовий', 'текст']

Після виконання тестів аналізуємо отримані результати. Якщо функція пройшла тест успішно та повернула очікуваний результат, вона вважається працездатною. Якщо виявлені проблеми або розходження між отриманими та очікуваними результатами, ми ретельно аналізуємо причину та вносимо необхідні зміни до програми. У разі виявлення помилок або неправильної роботи функцій, ми виправляємо проблеми, проводимо регресійне тестування (перевірка, що поправки не вплинули на раніше працездатні функції), та повторно запускаємо функціональні тести.

Функціональні тести дозволяють перевірити правильність роботи окремих функцій програми та їх взаємодію з іншими компонентами. Вони допомагають виявляти та усувати помилки, підтверджують, що програмне забезпечення відповідає специфікаціям та вимогам, та забезпечують його коректну роботу.

4.2. Модульне тестування

Юніт-тести є важливою складовою частиною процесу тестування програмного забезпечення. Вони призначені для перевірки окремих компонентів програми, таких як функції, методи чи класи, з метою забезпечення їх коректної роботи. У цьому пункті ми детальніше розглянемо юніт-тести і їх важливість у розробці програмного забезпечення.

Основна мета юніт-тестування полягає у виявленні помилок та неполадок у найменших функціональних одиницях програми. Воно дозволяє перевірити, чи відповідають окремі компоненти програми очікуваному поведінці та чи функціонують правильно. Проходження юніт-тестів дає розробникам впевненість у працездатності індивідуальних компонентів, а також забезпечує швидке виявлення та виправлення помилок у ранній стадії розробки.

Перед початком юніт-тестування необхідно підготувати середовище та залежності для виконання тестів. Зазвичай для цього використовують спеціальні фреймворки або бібліотеки для тестування, такі як unittest для мови Python. Для кожного компонента програми створюються відповідні тести, які будуть перевіряти його функціональність.

Кожен юніт-тест має певну структуру, що включає наступні елементи:

– підготовка (Setup): у цьому етапі виконується підготовка до тесту, включаючи створення необхідних об'єктів, ініціалізацію змінних та налаштування оточення для виконання тесту;

– виконання (Execution): у цьому етапі запускається перевірка окремого компонента програми, який тестується - зазвичай це виклик функції або методу, який очікується бути протестованим;

– перевірка (Assertion): після виконання компонента програми виконується перевірка його результату або стану - використовуються спеціальні оператори перевірки (assertions), які порівнюють отриманий результат з очікуваним;

– прибирання (Cleanup): на цьому етапі відбувається прибирання після виконання тесту, включаючи звільнення ресурсів, видалення тимчасових об'єктів або скасування змін відносно тестового середовища.

У юніт-тестуванні можна використовувати різні види тестів залежно від потреб проекту. Основні типи юніт-тестів включають:

– позитивні тести: тести, які перевіряють правильну роботу компонента при передачі йому коректних вхідних даних;

– негативні тести: тести, які перевіряють поведінку компонента при передачі йому некоректних або неприпустимих вхідних даних;

– межові тести: тести, які перевіряють роботу компонента на межі його можливостей, наприклад, при максимальних або мінімальних значеннях вхідних параметрів.

Наприклад, як видно з рисунка 4.1 у методі test_open_file спочатку симулюємо натискання кнопки "Open" за допомогою self.text_editor.open_file(). Потім перевіряємо, чи відкрилося вікно вибору файлу, використовуючи self.assertIsNotNone(self.text_editor.file_path). Аналогічним чином, в методах-тестах test_change_language і test_manual_search симулюємо певні події та перевіряємо очікувані результати.

```
def test_open_file(self):
    # Симулюємо натискання кнопки "Open"
    self.text_editor.open_file()
    # Перевіряємо, чи відкрилось вікно вибору файлу
    self.assertIsNotNone(self.text_editor.file_path)

def test_change_language(self):
    # Симулюємо зміну мови
    self.text_editor.language.set("Українська")
    self.text_editor.change_language()
    # Перевіряємо, чи змінилася обрана мова
    self.assertEqual(self.text_editor.language.get(), "Українська")

def test_manual_search(self):
    # Симулюємо натискання кнопки "Пошук"
    self.text_editor.open_search_window()
    self.text_editor.manual_search()
    # Перевіряємо, чи відкрите вікно ручного пошуку
    self.assertIsNotNone(self.text_editor.search_window)
```

Рисунок 4. 1 – Приклад юніт-тесту в коді програми

4.3. Інтеграційні тести

Інтеграційні тести є важливою складовою частиною тестування програмного забезпечення. Вони спрямовані на перевірку взаємодії різних компонентів програми та їх правильне функціонування разом. Ці тести перевіряють, чи відбувається передача даних між компонентами без помилок, чи правильно

відбувається обмін повідомленнями, чи реагують компоненти на зміни в середовищі тощо.

В процесі інтеграційного тестування було виконано перевірку, чи правильно взаємодіють компоненти через їх інтерфейси. Це означає, що передавались дані від одного компонента до іншого та перевіряли, чи коректно вони обробляються і повертають очікувані результати. Створювалися тестові сценарії, що охоплюють різні варіанти використання програмного забезпечення. Наприклад, перевірялось, чи правильно компоненти взаємодіють у складних сценаріях, які включають послідовність дій та зміни стану системи.У деяких випадках, коли компоненти програми були ще нереалізованими або недоступними для тестування, використовувалось мокування. Мокування дозволяє створювати симуляцію компонента, який поводиться так само, як реальний компонент, але не має реальної реалізації. Це дозволило протестувати взаємодію між компонентами на етапах, коли деякі з них ще не були готові.

ВИСНОВКИ

1. Проаналізовано алгоритми та методи для обробки природньої мови.
2. Проаналізовано інструментальні засоби для формування тематичних словників: Sketch Engine, AntConc, TshwaneLex. Порівняно їх характеристики. Виявлено переваги та недоліки.
3. Вибрано засоби реалізації додатку для автоматизованого формування тематичного словника. Обрано мову Python, бібліотеку для обробки природної мови NLTK, бібліотеку для створення інтерфейсу користувача Tkinter, бібліотеки matplotlib та wordcloud для формування та відображення хмар слів.
4. Розроблено алгоритм роботи додатку та програмно реалізовані ключові функціональні можливості, включаючи завантаження текстових документів, їх зміну, вибору мови, ручного пошуку термінів, пошуку термінів по згенерованим ключовим словам на основі текстів за тематикою дисципліни «Робототехніка».
5. Проведено функціональне та модульне тестування додатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. SketchEngine [Електронний ресурс] – Режим доступу до ресурсу: https://www.sketchengine.eu/.

2. OneClick Terms [Електронний ресурс] – Режим доступу до ресурсу: https://terms.sketchengine.eu/.

3. AntConc [Електронний ресурс] – Режим доступу до ресурсу: https://www.laurenceanthony.net/software/antconc/.

4. de Schryver G. -. TshwaneLex – Professional off-the-shelf lexicography software [Електронний ресурс] / Gilles -Maurice de Schryver – Режим доступу до ресурсу:https://www.academia.edu/8199514/TshwaneLex_Professional_off_theshelf_lexicography_software.

5. Natural Language Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/.

6. Kulshrestha R. A Beginner's Guide to Latent Dirichlet Allocation(LDA) [Електронний ресурс] / Ria Kulshrestha. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2.

7. Lasswell, H. D. (1951). The uses of content analysis data in studying social change 57-70.

8. Friedman, S. (1991). Auto-Med: a computer-assisted content analysis system. In Proceedings of the Annual Conference of the American Association for Public Opinion Research (Vol. 15, No. 1, pp. 383-392)

9. OpenNLP: A collection of tools for natural language processing. (1999) [Електронний ресурс] - Режим доступу до ресурсу: http://opennlp.sourceforge.net/

10. Yaohou F. Stop Words for Processing Software Engineering Documents: Do they Matter? [Електронний ресурс] / F. Yaohou, A. Chetan, T. Christoph. – 2023. – Режим доступу до ресурсу: https://arxiv.org/abs/2303.10439.

11. Jurafsky D. Regular Expressions, Text Normalization, Edit Distance [Електронний ресурс] / D. Jurafsky, H. James. – 2023. – Режим доступу до ресурсу: https://web.stanford.edu/~jurafsky/slp3/2.pdf.

12. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] / T.Mikolov, K. Chen, G. Corrado, J. Dean. – 2013. – Режим доступу до ресурсу: https://arxiv.org/abs/1301.3781.

13. Opinion mining on large scale data using sentiment analysis and k-means clustering [Електронний ресурс] / Sumbal Riaz, Mehvish Fatima, M. Kamran, M. Wasif Nisar. – 2017. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s10586-017-1077-z.

14. Bird S. Natural Language Processing with Python / S. Bird, E. Klein, E. Loper – Gravenstain: O'Reilly Media, 2009. – C. 61.

15. Qaiser S. Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents [Електронний ресурс] / S. Qaiser, R. Ali. – 2019. – Режим доступу до ресурсу: https://www.researchgate.net/profile/Shahzad-Qaiser/publication/326425709_Text_Mining_Use_of_TF-IDF_to_Examine_the_Relevance_of_Words_to_Documents/links/5b4cd57fa6fdcc8dae245aa3/Text-Mining-Use-of-TF-IDF-to-Examine-the-Relevance-of-Words-to-Documents.pdf.

16. Mihalcea R. TextRank: Bringing Order into Texts [Електронний ресурс] / R. Mihalcea, P. Tarau – Режим доступу до ресурсу: https://aclanthology.org/W04-3252.pdf.

17. Schönfeld M. Graph drawing by force-directed placement. Software: Practice and Experience, 21(11), / M. Schönfeld, J. Pfeffer., 1991. – 1129 c.

18. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software [Електронний ресурс] / M.Jacomy, T. Venturini, S. Heymann, M. Bastian. – 2014. – Режим доступу до ресурсу: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0098679.

19. Holten D. Force-Directed Edge Bundling for Graph Visualization [Електронний ресурс] / D. Holten, J. J. Van Wijk. – 2009. – Режим доступу до ресурсу: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01450.x.

ДОДАТОК А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ ТЕМАТИЧНОГО СЛОВНИКА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ "РОБОТОТЕХНІКА" МОВОЮ PYTHON

Виконав студент 4 курсу
групи ПД-41
Лукашенко Ігор Петрович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**
  спрощення процесу формування тематичного словника з навчальної дисципліни «Робототехніка»за рахунок програмного забезпечення створеного мовою Python
- **Об'єкт дослідження**
  автоматизована побудова тематичного словника навчальної дисципліни.
- **Предмет дослідження**
  програмне забезпечення мовою Python для автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка"

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз алгоритмів та методів для обробки природньої мови.
2. Аналіз аналогічних засобів для формування тематичних словників.
3. Вибір засобів реалізації додатку для автоматизованого формування тематичного словника.
4. Проектування вимог додатку для автоматизованого формування тематичного словника.
5. Розробка додатку відповідно до створених вимог.
6. Тестування програмного забезпечення.

3

АНАЛІЗ АНАЛОГІВ

SKETCH ENGINE AntConc

Показник	Sketch Engine	AntConc	TshwaneLex	Розроблений додаток
Платформа	Windows 10, MacOS, Linux ,Web	Windows 10, MacOS, Linux	Windows 10	Windows 10
Пошук за шаблоном	+	+	+	+
Підтримка Unicode	+	+	+	+
Автоматична обробка тексту	+	+	-	+
Хмара слів	-	+	-	+
Генерація списку ключових слів, понять та визначень за тематикою дисципліни	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Можливість завантаження текстових даних за тематикою дисципліни.
2. Наявність засобів попередньої обробки текстів.
3. Можливість пошуку понять та визначень за шаблоном.
4. Формування списку ключових слів, понять та визначень за тематикою дисципліни.
5. Ранжування ключових слів, понять та визначень .
6. Графічний інтерфейс для кращої навігації.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

python

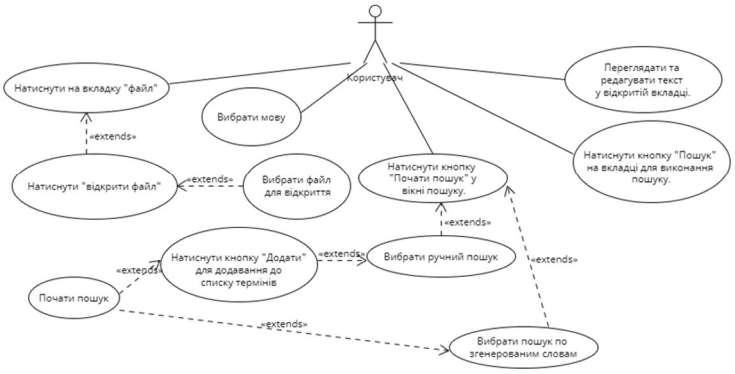
NLTK

Tkinter

matplotlib

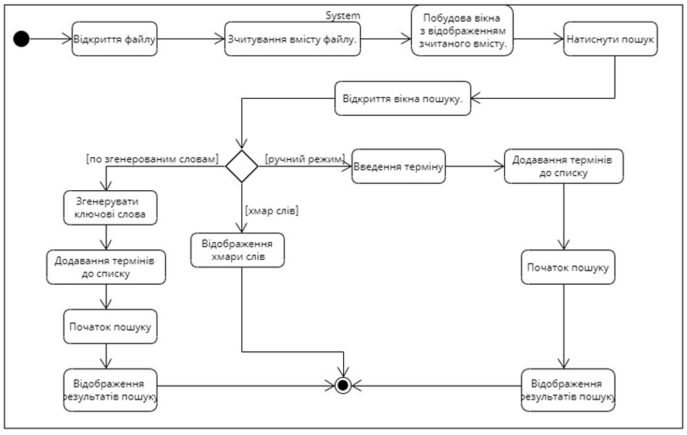
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ ДОДАТКУ



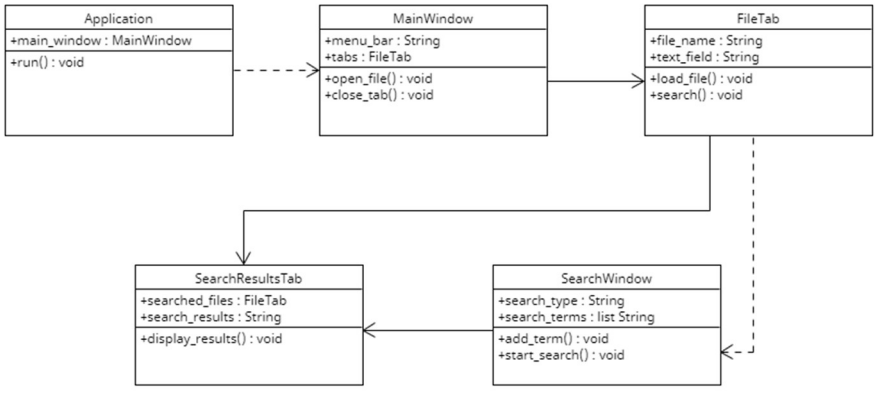
7

ДІАГРАМА ДІЯЛЬНОСТІ ДОДАТКУ



8

ДІАГРАМА КЛАСІВ ДОДАТКУ



9

ЕКРАННІ ФОРМИ

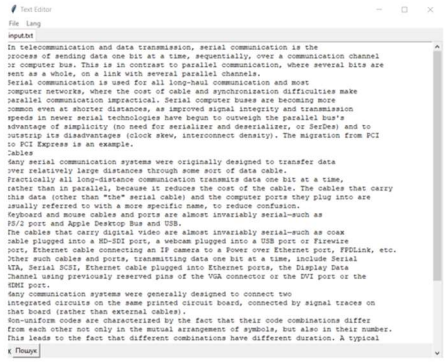


Рис. 9.1 - Головний екран додатку

Рис. 9.2 – Екран відображення хмари слів

10

ЕКРАННІ ФОРМИ

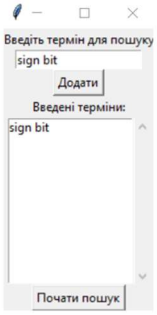


Рис. 10.1 - Вікно для вводу термінів для ручного пошуку

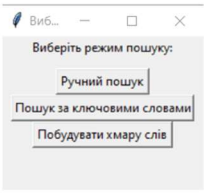


Рис. 10.2 - Вікно для вибору режиму

Рис. 10.3 - Вікно для виведення результатів

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Лукашенко І.П. Використання засобів мови python для вирішення задач обробки тексту/ О. А. Золотухіна, Лукашенко І.П.//Застосування програмного забезпечення в інфокомунікаційних технологіях. Збірник тез. 20.04.2023, ДУТ, м. Київ. - К.: ДУТ, 2023. — С. 90.
2. Лукашенко І.П. Автоматизована система побудови тематичного словника/ О. А. Золотухіна, Лукашенко І.П.//Сучасний стан та перспективи розвитку IoT. Збірник тез. ДУТ, м. Київ. - К.: ДУТ, 2023. . — С. 104.

12

ВИСНОВКИ

1. Проаналізовано алгоритми та методи для обробки природньої мови.
2. Проаналізовано інструментальні засоби для формування тематичних словників: Sketch Engine, AntConc, TshwaneLex. Порівняно їх характеристики. Виявлено переваги та недоліки.
3. Вибрано засоби реалізації додатку для автоматизованого формування тематичного словника. Обрано мову Python, бібліотеку для обробки природної мови NLTK, бібліотеку для створення інтерфейсу користувача Tkinter, бібліотеки matplotlib та wordcloud для формування та відображення хмар слів.
4. Розроблено алгоритм роботи додатку та програмно реалізовані ключові функціональні можливості, включаючи завантаження текстових документів, їх зміну, вибору мови, ручного пошуку термінів, пошуку термінів по згенерованим ключовим словам на основі текстів за тематикою дисципліни «Робототехніка».
5. Проведено функціональне та модульне тестування додатку.

13

ДОДАТОК Б
ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
#5-та версія
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
import re
import PyPDF2
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize



class TextEditor:
    def __init__(self, master):
        self.master = master
        self.master.title("Text Editor")
        self.master.geometry("800x600")

        self.notebook = ttk.Notebook(self.master)
        self.notebook.pack(fill=tk.BOTH,
expand=True)
        self.content = []
        self.filename = ''
        self.language = tk.StringVar()
        self.language.set("English")

        menubar = tk.Menu(self.master)
        filemenu = tk.Menu(menubar, tearoff=0)
        filemenu.add_command(label="Open",
command=self.open_file)
        filemenu.add_separator()
        filemenu.add_command(label="Exit",
command=self.master.quit)
        menubar.add_cascade(label="File",
menu=filemenu)
        self.master.config(menu=menubar)

        lang_menu = tk.Menu(menubar,
tearoff=0)

lang_menu.add_radiobutton(label="English",
variable=self.language, value="English",
command=self.change_language)

lang_menu.add_radiobutton(label="Українська
", variable=self.language, value="Українська",
command=self.change_language)
        menubar.add_cascade(label="Lang",
menu=lang_menu)

        self.master.config(menu=menubar)


    def change_language(self):
        selected_language = self.language.get()
        print("Selected Language:",
selected_language)

    def open_file(self):
```

```
        file_path =
filedialog.askopenfilename(title="Open file",
                                   filetypes=[("Text
files", "*.txt"), ("PDF files", "*.pdf")])
        if file_path:
            if file_path.endswith(".txt"):
                with open(file_path, "r",
encoding="utf-8") as file:
                    self.content = file.read()

            elif file_path.endswith(".pdf"):
                with open(file_path, "rb") as file:

                    pdf_reader =
PyPDF2.PdfReader(file)
                    self.content = ""

                    for page in pdf_reader.pages:
                        self.content +=
page.extract_text()


                    # file.close()

            self.filename = file_path.split("/")[-1]
            frame = tk.Frame(self.notebook)
            self.notebook.add(frame,
text=self.filename)

            scrollbar = tk.Scrollbar(frame)
            scrollbar.pack(side=tk.RIGHT,
fill=tk.Y)

            text_area = tk.Text(frame,
wrap=tk.NONE,
yscrollcommand=scrollbar.set)
            text_area.insert(tk.END, self.content)
            text_area.pack(fill=tk.BOTH,
expand=True)


scrollbar.config(command=text_area.yview)

            close_button = tk.Label(frame,
text="x", cursor="hand2", font=("Arial", 12))
            close_button.pack(side=tk.LEFT)
            close_button.bind("<Button-1>",
lambda event: self.close_tab(event, frame))

            search_button = tk.Button(frame,
text="Пошук",
command=self.open_search_window)
            search_button.pack(side=tk.LEFT)

            self.notebook.select(frame)

    def close_tab(self, event, frame):
        self.notebook.forget(frame)

    def open_search_window(self):
        search_window = tk.Toplevel(self.master)
        search_window.title("Вибір режиму
пошуку")
        search_window.geometry("200x150")

        search_label = tk.Label(search_window,
text="Виберіть режим пошуку:")
        search_label.pack()
```

```
        button_frame = tk.Frame(search_window, pady=10)
        button_frame.pack()

        manual_search_button = tk.Button(button_frame, text="Ручний пошук", command=self.manual_search)
        manual_search_button.pack(side=tk.TOP, padx=10)

        auto_search_button = tk.Button(button_frame, text="Пошук за ключовими словами")

        selected_language = self.language.get()
        print(self.language.get())
        if selected_language == "English":

auto_search_button.config(command=self.start_auto_search_english)
        elif selected_language == "Українська":

auto_search_button.config(command=self.start_auto_search_ukrainian)

        auto_search_button.pack(side=tk.TOP, padx=10)

        wordcloud_button = tk.Button(button_frame, text="Побудувати хмару слів", command=self.generate_wordcloud)
        wordcloud_button.pack()

    def generate_wordcloud(self):
        # Отримайте текст з документа або іншого джерела
        text = self.content

        # Побудуйте хмару слів
        wordcloud = WordCloud().generate(text)

        # Відобразіть хмару слів у новому вікні
        self.show_wordcloud(wordcloud)

    def show_wordcloud(self, wordcloud):
        # Створіть нове вікно


        # Відобразіть хмару слів у вікні
        figure = plt.figure(figsize=(10, 8), dpi=80)
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.show()

    def manual_search(self):
        search_window = tk.Toplevel(self.master)
        search_window.title("Ручний пошук")

        search_label = tk.Label(search_window, text="Введіть термін для пошуку")
        search_label.pack()

        search_entry = tk.Entry(search_window)
        search_entry.pack()
```

```
        add_button = tk.Button(search_window,
text="Додати", command=lambda:
self.add_search_term(search_entry.get()))
        add_button.pack()

        terms_label = tk.Label(search_window,
text="Введені терміни:")
        terms_label.pack()

        terms_frame = tk.Frame(search_window)
        terms_frame.pack()

        self.terms_listbox =
tk.Listbox(terms_frame,
selectmode=tk.MULTIPLE)
        self.terms_listbox.pack(side=tk.LEFT)

        scrollbar = tk.Scrollbar(terms_frame,
command=self.terms_listbox.yview)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)


self.terms_listbox.config(yscrollcommand=scro
llbar.set)



        search_button =
tk.Button(search_window, text="Почати
пошук")

        selected_language = self.language.get()
        if selected_language == "English":

search_button.config(command=self.start_man
ual_search_english)

        elif selected_language == "Українська":

search_button.config(command=self.start_man
ual_search_ukrainian)

        search_button.pack()



    def add_search_term(self, term):
        if term.strip() != "":
            self.terms_listbox.insert(tk.END, term)

    def start_manual_search_english(self):
        terms = self.terms_listbox.get(0, tk.END)
        search_results =
self.search_files_english(terms)
        self.display_search_results(search_results)

    def start_auto_search_english(self):
        keywords =
self.generate_keywords_english()
        search_results =
self.search_files_english(keywords)

self.display_auto_search_results(keywords,
search_results)

    def start_manual_search_ukrainian(self):
        terms = self.terms_listbox.get(0, tk.END)
        search_results =
self.search_files_ukrainian(terms)
        self.display_search_results(search_results)

    def start_auto_search_ukrainian(self):
```

```
        keywords =
self.generate_keywords_ukrainian()
        search_results =
self.search_files_ukrainian(keywords)

self.display_auto_search_results(keywords,
search_results)

    def search_files_english(self, terms):
        res = {}
        res["results"] = []
        sentences = []
        content = self.content.lower()
        for term in terms:
            if re.findall(r'[^.!?]*' + term.lower() + r'
- [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' - [^.!?]*[.!?]', content))

            if re.findall(r'[^.!?]*' + term.lower() + r'
is [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' is [^.!?]*[.!?]', content))
            else:
                if re.findall(r'[^.!?]*' + term.lower() +
r' [^.!?]*[.!?]', content):

sentences.extend(re.findall(r'[^.!?]*' +
term.lower() + r' is [^.!?]*[.!?]', content))

        res["files"] = [self.filename]
        res["results"] = sentences
        return res

    def search_files_ukrainian(self, terms):
        res = {}
        res["results"] = []
        sentences = []
        content = self.content.lower()
        for term in terms:
            if re.findall(r'[^.!?]*' + term.lower() + r'
- [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' - [^.!?]*[.!?]', content))

            if re.findall(r'[^.!?]*' + term.lower() + r'
- це [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' is [^.!?]*[.!?]', content))

            if re.findall(r'[^.!?]*' + term.lower() + r'
це [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' is [^.!?]*[.!?]', content))

            if re.findall(r'[^.!?]*' + term.lower() + r'
є [^.!?]*[.!?]', content):
                sentences.extend(re.findall(r'[^.!?]*'
+ term.lower() + r' is [^.!?]*[.!?]', content))
            else:
                if re.findall(r'[^.!?]*' + term.lower() +
r' [^.!?]*[.!?]', content):

sentences.extend(re.findall(r'[^.!?]*' +
term.lower() + r' is [^.!?]*[.!?]', content))

        res["files"] = [self.filename]
        res["results"] = sentences
        return res
```

```
    def generate_keywords_english(self):
        # Код генерації ключових слів з тексту
        # Наприклад, keywords = ["ключове", "слово", "генерація"]
        keywords = []

        stop_words = set(stopwords.words('english'))  # Застосуйте відповідну мову
        word_frequencies = {}

        # Аналіз файлу і знаходження частоти вживання слів
        if self.filename.endswith(".txt"):
            with open(self.filename, "r", encoding="utf-8") as file:
                content = file.read()
                tokens = word_tokenize(content)

                for word in tokens:
                    word = word.lower()
                    if word.isalpha() and word not in stop_words:
                        if word not in word_frequencies:
                            word_frequencies[word] = 1
                        else:
                            word_frequencies[word] += 1

        elif self.filename.endswith(".pdf"):
            keywords = []
            with open(self.filename, 'rb') as file:
                pdf_reader = PyPDF2.PdfReader(file)
                for page_num in range(len(pdf_reader.pages)):
                    page = pdf_reader.pages[page_num]
                    text = page.extract_text()
                    tokens = word_tokenize(text)
                    keywords.extend([token for token in tokens if token.isalpha()])  # Фільтрація тільки слів


        # Сортування слів за частотою вживання
        sorted_words = sorted(word_frequencies, key=word_frequencies.get, reverse=True)

        # Вибір перших N слів як ключових слів
        num_keywords = 10  # Задайте бажану кількість ключових слів
        keywords = sorted_words[:num_keywords]

        return keywords

    def generate_keywords_ukrainian(self):
        # Код генерації ключових слів з тексту
        # Наприклад, keywords = ["ключове", "слово", "генерація"]
        keywords = []

        stop_words = set(stopwords.words('ukrainian'))  # Застосуйте відповідну мову
        word_frequencies = {}
```

```
        # Аналіз файлу і знаходження частоти вживання слів
        if self.filename.endswith(".txt"):
            with open(self.filename, "r", encoding="utf-8") as file:
                content = file.read()
                tokens = word_tokenize(content)

                for word in tokens:
                    word = word.lower()
                    if word.isalpha() and word not in stop_words:
                        if word not in word_frequencies:
                            word_frequencies[word] = 1
                        else:
                            word_frequencies[word] += 1

        elif self.filename.endswith(".pdf"):
            keywords = []
            with open(self.filename, 'rb') as file:
                pdf_reader = PyPDF2.PdfReader(file)
                for page_num in range(len(pdf_reader.pages)):
                    page = pdf_reader.pages[page_num]
                    text = page.extract_text()
                    tokens = word_tokenize(text)
                    keywords.extend([token for token in tokens if token.isalpha()])  # Фільтрація тільки слів


        # Сортування слів за частотою вживання
        sorted_words = sorted(word_frequencies, key=word_frequencies.get, reverse=True)

        # Вибір перших N слів як ключових слів
        num_keywords = 10  # Задайте бажану кількість ключових слів
        keywords = sorted_words[:num_keywords]

        return keywords

    def display_search_results(self, search_results):
        result_window = tk.Toplevel(self.master)
        result_window.title("Результати пошуку")

        result_label = tk.Label(result_window, text="Пошук був проведений у наступних файлах:")
        result_label.pack()

        files_label = tk.Label(result_window, text="".join(search_results["files"]))
        files_label.pack()

        result_text = tk.Text(result_window, wrap=tk.WORD)
        result_text.pack(fill=tk.BOTH, expand=True)

        for result in search_results["results"]:
            result_text.insert(tk.END, str(result) + '\n')
```

```
        print(search_results)

    def display_auto_search_results(self,
keywords, search_results):
        auto_result_window =
tk.Toplevel(self.master)
        auto_result_window.title("Результати
автоматичного пошуку")

        keywords_label =
tk.Label(auto_result_window,
text="Згенеровані ключові слова:")
        keywords_label.pack()

        keywords_text =
tk.Text(auto_result_window, wrap=tk.WORD)
        keywords_text.pack(fill=tk.BOTH,
expand=True)

        for keyword in keywords:
            keywords_text.insert(tk.END,
str(keyword) + '\n')

        result_label =
tk.Label(auto_result_window, text="Пошук
був проведений у наступних файлах:")
        result_label.pack()

        files_label =
tk.Label(auto_result_window,
text="".join(search_results["files"]))
        files_label.pack()

        result_text = tk.Text(auto_result_window,
wrap=tk.WORD)
        result_text.pack(fill=tk.BOTH,
expand=True)

        keywords_text.configure(height=5)

        for result in search_results["results"]:
            result_text.insert(tk.END, str(result) +
'\n')


if __name__ == "__main__":
    root = tk.Tk()
    app = TextEditor(root)
    root.mainloop()
```