Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

за спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка системи інтелектуального аналізу тексту

Засвідчую, що в цій кваліфікаційній роботі немає
запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-20-2
_____/М.С.Чепурко/

Керівник
кваліфікаційної роботи _____
/Н.Н.Шаповалова/

Завідувач кафедри _____ /А.М.Стрюк/

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20-2 Чепурко Максиму Сергійовичу

1. Тема: Розробка системи інтелектуального аналізу тексту. Затверджено наказом по КНУ №__ від «_» _____ 2024р.

2. Термін подання студентом закінченої роботи : «_» _____ 2024р.

3. Вихідні дані по роботі: розроблювана система повина працювати з текстами (статтями, коментарями, відгуками) англійської мовою.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз вже існуючих систем інтелектуального аналізу тексту, розглянути методи та технології які можна використати під час розробки, вибрати найефективніші. Спроектувати систему інтелектуального аналізу тексту, програмно реалізувати розроблену систему, провести фінальне тестування для перевірки правильності роботи функціоналу системи.

5. Перелік ілюстративного матеріалу: блок-схеми розроблюваних алгоритмів системи, знімки екранних форм, графи нейронних мереж, схеми взаємодії програмних модулів між собою.

Календарний план**:**

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	28.01.2024 – 02.02.2024
2	Проведення аналізу існуючих систем інтелектуального аналізу тексту	04.02.2024 – 08.02.2024
3	Підготовка матеріалів першого розділу роботи	09.02.2024 – 15.02.2024
4	Проектування програмного забезпечення для реалізації системи інтелектуального аналізу тексту	18.02.2024 – 28.02.2024
5	Підготовка матеріалів другого розділу роботи	02.03.2024 – 06.03.2024
6	Розробка візуальної складової програмного забезпечення	07.03.2024 – 14.03.2024
7	Підготовка матеріалів третього розділу роботи	15.03.2024 – 25.03.2024
8	Розробка та реалізація функціоналу системи	26.03.2024 – 14.04.2024
9	Структуризація інформації	15.04.2024 – 17.04.2024
10	Оформлення появснювальної записки	19.04.2024 – 20.05.2024

Дата видачі завдання: «27» січня 2024р.

Студент _____/ М.С.Чепурко

Керівник роботи _____/Н.Н.Шаповалова

РЕФЕРАТ

СИСТЕМА ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ТЕКСТУ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, АЛГОРИТМ НАВЧАННЯ, АНАЛІЗ ТЕКСТУ, КЛАСИФІКАЦІЯ ТЕКСТУ, ВИДОБУТОК ІНФОРМАЦІЇ.

Пояснювальна записка: 85 с., 6 табл., 35 рис., 1 дод., 12 джерел.

Інтелектуальний аналіз тексту є актуальною проблемою в сучасному інформаційному суспільстві. У даній кваліфікаційній роботі досліджується розробка системи інтелектуального аналізу тексту з використанням сучасних нейромережевих методів.

Метою кваліфікаційної роботи є розробка системи інтелектуального аналізу тексту здатною до: семантичного аналізу, знаходження лексичних та орфографічних помилок, визначення теми тексту та цільової аудиторії, можливість побудови хмар слів, графіків та короткого змісту.

В ході дослідження буде проведено аналіз існуючих методів та алгоритмів машинного навчання для обробки природної мови. На основі цього буде розроблено архітектуру системи, що включатиме модулі попередньої обробки тексту, класифікації тексту та видобутку інформації.

Розроблена система буде досліджена на тестових наборах даних. Її ефективність буде оцінена та порівняна з іншими системами аналізу тексту.

Практичне значення розробленої системи полягає в можливості її використання для вирішення різних задач, пов'язаних з аналізом тексту, таких як автоматична категоризація документів, аналіз відгуків, виявлення спаму, видобуток інформації з текстів, аналіз на помилки та скорочення текстів.

ABSTRACT

TEXT MINING, MACHINE LEARNING, NATURAL LANGUAGE PROCESSING, LEARNING ALGORITHM, TEXT ANALYSIS, TEXT CLASSIFICATION, INFORMATION EXTRACTION.

Thesis in 85 p., 6 table, 35 figure, 1 appendix, 12 source.

Intellectual text analysis is an urgent problem in the modern information society. This qualification work investigates the development of a text mining system using modern neural network methods.

The purpose of the qualification work is to develop a system of intellectual text analysis capable of: semantic analysis, finding lexical and spelling errors, determining the topic of the text and the target audience, the ability to build word clouds, graphs and summaries.

The research will analyze existing machine learning methods and algorithms for natural language processing. Based on this, the system architecture will be developed, including modules for text pre-processing, text classification, and information extraction.

The developed system will be tested on test datasets. Its effectiveness will be evaluated and compared with other text analysis systems.

The practical significance of the developed system lies in the possibility of using it to solve various tasks related to text analysis, such as automatic document categorization, review analysis, spam detection, information extraction from texts, error analysis, and text abridgement.

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ ТА СКОРОЧЕННЯ

1. GPT - Generative Pre-trained Transformer, це штучний інтелект, який вміє розуміти і створювати людську мову.

2. NLTK - провідна платформа для створення програм на мові Python для роботи з даними людської мови.

3. RNN - це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі.

4. NLP (Natural Language Processing) - здатність комп'ютерної програми розуміти людську мову в усній та письмовій формі.

5. Датасет – набір даних який в контексті машинного навчання необхідний для навчання нейронної мережі.

ВСТУП

У еру динамічного розвитку інформаційної сфери, текст все одно залишається домінуючим носієм людської думки. Від об'ємних статей та книг до невеликих «твітів[1]», коментарів або відгуків про якийсь товар. Все це несе в собі інформацію. Тому, звичайно, виникає потреба в інтелектуальному аналізі текстів - прагненні вийти за межі простого розпізнавання слів і заглибитися в суть того про що текст говорить.

За своєю суттю інтелектуальний аналіз тексту (IAT) охоплює спектр методологій спрямованих на автоматизацію розуміння та інтерпретації текстового контенту. Ключовими завданнями IAT є категоризація текстів, пошук та обробка інформації в них, а також різні способи представити цю інформацію користувачу (візуалізувати, знайти з тексту головне).

Ця кваліфікаційна робота заглиблюється в сферу IAT, досліджуючи методи створення власної інтелектуальної системи. Використовуючи передові методи з обробки природної мови (NLP[2]), машинного навчання та когнітивних наук, я хочу створити власну систему яка буде здатна до семантичного аналізу, розпізнання чи використовувалися GPT технології для написання тексту, знаходження лексичних та орфографічних помилок, визначення теми тексту та цільової аудиторії, можливість побудови хмар слів, графіків, короткого змісту, тощо.

В ході роботи буде заглиблення в практичні аспекти реалізації системи інтелектуального аналізу тексту на мові Python. Це включає попередню обробку текстових даних для забезпечення їхньої придатності для аналізу, вилучення ознак для фіксації відповідних лінгвістичних патернів і структур, а також вибір і навчання моделей для забезпечення точного прогнозування та висновків.

---

[1] Твіт – невелике повідомлення у соціальній мережі «X» (в минулому Twitter).
[2] NLP - Natural Language Processing, обробка природної мови.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

Актуальність теми "Розробка системи інтелектуального аналізу тексту" можно характеризувати наступними важливими факторами:

1. Перше це зростаючий обсяг текстової інформації. Інтернет і соціальні мережі продовжують зростати кожного дня як з точки зору кількості користувачів так і в обсягу створюваного контенту. Велика кількість текстів, відгуків та коментарів які потрібно аналізувати, створює попит на автоматизовані системи для ефективного опрацювання і розуміння цієї інформації.

2. У світі де присутня велика кількість дезінформації, розуміння тексту стає важливим завданням для різних сфер, включаючи медіа, сферу дослідження цільової аудиторії, бізнес, тощо.

3. Простий пошук ключових слів вже не дає достатньо інформації для розуміння складних текстів. ІАТ використовує методи машинного навчання та NLP для виявлення семантичних зв'язків, емоцій, тональності, інтенції та інших більш детальних та важливих аспектів. Це дозволяє отримати глибше розуміння контенту та краще використовувати його для різних завдань.

4. Постійно з'являються нові методи та алгоритми, які роблять ІАТ більш ефективним і точним. Це відкриває нові можливості для розробки більш складних та досконалих систем аналізу тексту.

5. Велика кількість текстових даних в Інтернеті, може бути використана для прийняття рішень в бізнесі, аналізу трендів, прогнозування подій. Наприклад ІАТ може автоматично аналізувати текст коментарів клієнтів, виявляти їхні настрої та емоції щодо продуктів або послуг компанії. Це дозволяє бізнесу зрозуміти, як сприймаються їхні продукти або послуги на ринку і які аспекти можуть бути покращені.

1.2 Цілі та завдання кваліфікаційної роботи

Метою кваліфікаційної роботи є розробка системи інтелектуального аналізу тексту, здатної виконувати широкий спектр завдань, включаючи:

1. Семантичний аналіз - розуміння значення тексту, виявлення зв'язків між словами та фразами.
2. Розпізнавання GPT - визначення, чи використовувалися GPT-технології для написання тексту.
3. Виявлення помилок - знаходження лексичних та орфографічних помилок.
4. Визначення теми та аудиторії - визначення теми тексту та цільової аудиторії.
5. Візуалізація - побудова хмар слів, графіків та короткого змісту.

Для досягнення цієї мети я хочу провести аналіз наявних методів та алгоритмів машинного навчання для обробки природної мови. Після цього на основі цих знань буде розроблена архітектура системи, що включатиме модулі попередньої обробки тексту, класифікації текстів та видобутку необхідної інформації.

Методи дослідження наступні: аналіз літературних джерел, дослідження методів та алгоритмів NLP, проектування та реалізація програмного забезпечення, тестування та оцінка системи, аналіз практичного значення

Розроблена система буде піддана тестуванню на тестових наборах даних, а її ефективність буде оцінена та порівняна з іншими системами аналізу тексту. Практичне значення розробленої системи полягатиме у можливості використання її для автоматичної категоризації документів, аналізу відгуків, виявлення спаму, видобутку інформації з текстів, а також у аналізі помилок та скороченні текстів.

1.3 Критичний аналіз літературних джерел за темою

Прочитавши та проаналізувавши досить велику кількість літературних джерел, в тому числі наукових робіт і статей я прийшов до деяких висновків щодо поточної ситуації з ІАТ.

Наявні обмеження існуючих методів семантичного аналізу текстів, зокрема у вловленні семантичних нюансів. У той час як більшість підходів покладаються на зовнішні API для виділення семантичного фрейму, більш ефективно вводити лексико-семантичні функції, отримані з лексиконів настроїв і семантичних моделей розподілу. Це має на меті усунути неефективність обчислень і підвищити точність класифікації шляхом включення семантичної інформації безпосередньо в процес розробки функцій [1].

Семантична та лексикологічна різноманітність текстів це основна проблема в обробці природної мови (NLP) методами штучного інтелекту (ШІ). Ця різноманітність призводить до появи різних метрик машинного навчання, пристосованих для нейромережевого аналізу. Крім того, наявність "інформаційного сміття" або шуму в контенті ще більше ускладнює завдання класифікації текстів [2].

Вибір архітектур нейронних мереж, таких як згортковий (Conv1D) і рекурентний (CNN, LSTM) рівні для захоплення послідовностей і комбінацій аналізованих термінів.

Для дослідження та реалізації системи інтелектуального аналізу тексту Python є однією з найбільш доцільних і простих мов програмування через ряд переваг [3].

По-перше Python має широкий вибір бібліотек та фреймворків для розробки системи ІАТ. Деякі з найпопулярніших бібліотек включають:

1. TensorFlow і Keras: Для роботи з нейронними мережами.
2. PyTorch: Ще один потужний фреймворк для роботи з нейронними мережами.

3. Scikit-learn: Для машинного навчання та статистичного аналізу даних.

4. NLTK (Natural Language Toolkit): Для обробки природної мови. OpenCV: Для комп'ютерного зору та обробки зображень.

5. Pandas і NumPy: Для обробки та аналізу даних.

Також розробка на Python має велику та активну спільноту, яка постійно розширює функціональність мови та розробляє нові бібліотеки.

Семантичний аналіз є важливою особливістю підходу NLP. Він допомагає розуміти значення слів, фраз та речень у контексті мовлення. Це важливо для ефективного розуміння та інтерпретації інформації, яку генерують та сприймають люди [4].

Також семантичний аналіз використовується в системах рекомендацій для розуміння інтересів та вподобань користувачів на основі їхньої активності та інтеракцій з контентом. Семантичний аналіз дозволяє автоматизувати обробку великих обсягів текстових даних, виявляючи ключові концепти, відносини між ними та іншу важливу інформацію.

В цілому, в наукових роботах останніх років відзначається зростаючий інтерес до створення систем інтелектуального аналізу тексту. Більшість авторів звертаються до методів машинного навчання та обробки природної мови для розробки систем, спроможних розпізнавати та аналізувати текстову інформацію. Але також системи інтелектуального аналізу часто стикаються з проблемами точності та інтерпретації складних контекстуальних ситуацій.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка та докладний опис функціональної схеми програми

Структурна схема при проектуванні програмного забезпечення - це візуальне представлення взаємозв'язків між різними компонентами програми та їх організація в логічні блоки або модулі.

Ця схема допомагає розуміти архітектуру програми, визначити порядок виконання дій та зрозуміти взаємодію між компонентами.

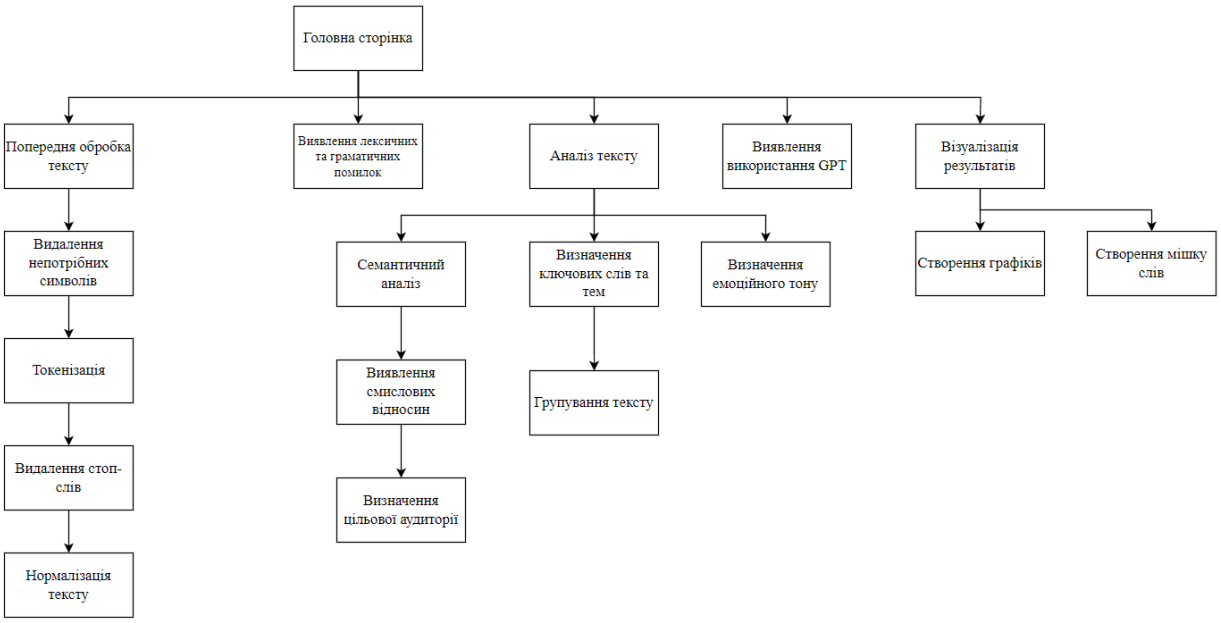


Рисунок 2.1 – Структурна схема програми

На рисунку 2.1 зображена структурна схема програми, вона описує головний функціонал ПЗ та складається з наступних пунктів:

1. Введення тексту:
    a. Користувач має можливість ввести текст для аналізу через інтерфейс або завантажити текстовий файл (JSON, txt).
2. Попередня обробка тексту:
    a. Видалення непотрібних символів, таких як пунктуація, спеціальні символи.

b. Токенізація - розділення тексту на окремі слова (токени).
c. Видалення зайвих слів (стоп-слів).
d. Нормалізація тексту - перетворення слів у їхню базову форму (лематизація або стемінг).

3. Аналіз тексту:
   a. Визначення мети тексту: розпізнання, чи текст призначений для інформування, переконання, розважання і т.д.
   b. Виявлення теми тексту - використання тематичного моделювання для ідентифікації головної теми або тем, що присутні у тексті.
   c. Визначення цільової аудиторії - застосування методів аналізу для визначення характеристик аудиторії, яка ймовірно зацікавлена у тексті.
4. Виявлення лексичних та орфографічних помилок:
   a. Використання методів виявлення помилок у написанні слів (spell checking) для корекції орфографічних помилок.
   b. Пошук та виправлення граматичних помилок.
5. Семантичний аналіз:
   a. Використання моделей з розуміння природної мови для виявлення семантичних зв'язків між словами та реченнями.
   b. Виділення ключових термінів та концепцій у тексті.
6. Візуалізація результатів:
   a. Відображення найбільш часто зустрічаються слів у вигляді графічної хмари.
   b. Створення графіків та діаграм для відображення результатів аналізу.
   c. Автоматичне створення короткого опису тексту на основі виявлених ключових термінів та інформації.

Контекстна діаграма аналізу тексту зображена на рисунку 2.2 дає загальне уявлення про систему та її взаємодію з зовнішніми сутностями. Ці

зовнішні сутності - користувачі та сервер. Система аналізу тексту отримує текстові дані від користувачів у різних форматах, включаючи JSON, TXT та DOCX-файли. Потім система аналізує ці текстові дані та надає результат, який залежить від запитів користувача.

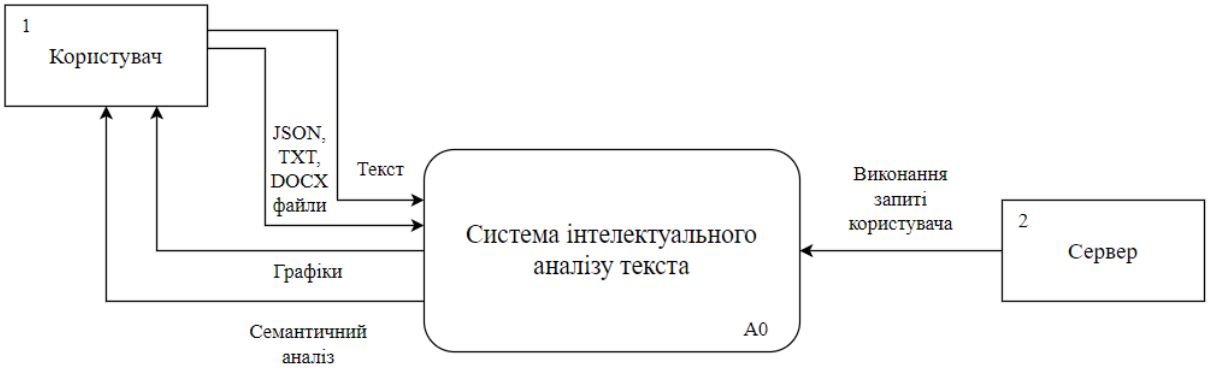


Рисунок 2.2 – Контекстна діаграма потоків даних

На рисунку 2.3 зображена загальна діаграма потоків даних. На ній:

1. БД текстів. Система використовує базу даних текстів, яка містить текстові дані, доступні для обробки.
2. Користувач. Користувач заходить на сайт і вводить текстові дані.
3. Введення текстових даних. Введені текстові дані обробляються системою.
4. Вибір функцій. Користувач вибирає функцію, яку система має виконати з текстовими даними.
5. Виконання функції. Система виконує вибрану функцію, обробляючи текстові дані.
6. Формування результатів. Система формує результати обробки текстових даних.
7. Результати. Користувач отримує результати обробки текстових даних.

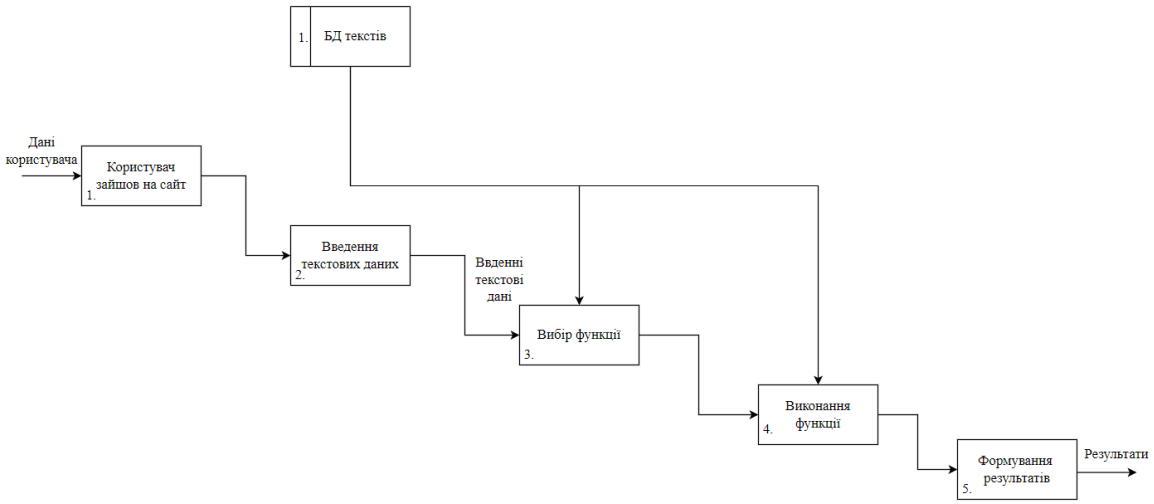


Рисунок 2.3 – Загальна діаграма потоків даних

На рисунку 2.4 зображена деталізовано блок-схема яка описує процес обробки даних, який складається з 5 основних кроків:

1. Перевірка введення даних. Цей компонент використовується для перевірки правильності введених даних.

2. Відображення повідомлення про помилку. Якщо введені дані некоректні, система відображає повідомлення про помилку.

3. Валідація даних для вибраної функції. Цей компонент використовується для перевірки того, чи підходять дані для вибраної функції.

4. Обробка даних. Цей компонент використовується для обробки текстових даних. Обробка даних включає в себе такі дії, як токенізація, лематизація, стоп-слова, POS-тегинг, синтаксичний аналіз та семантичний аналіз.

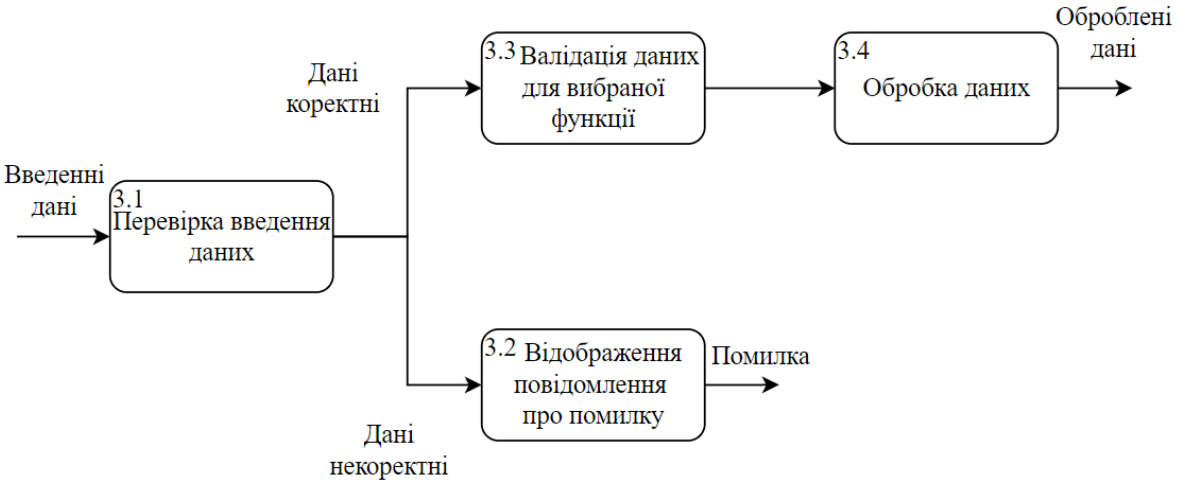


Рисунок 2.4 – Деталізована діаграма потоків даних для вибору функції

На рисунку 2.5 зображено загальну функціональну схему ПЗ яка приймає текстові дані з різних джерел та з ними працює.

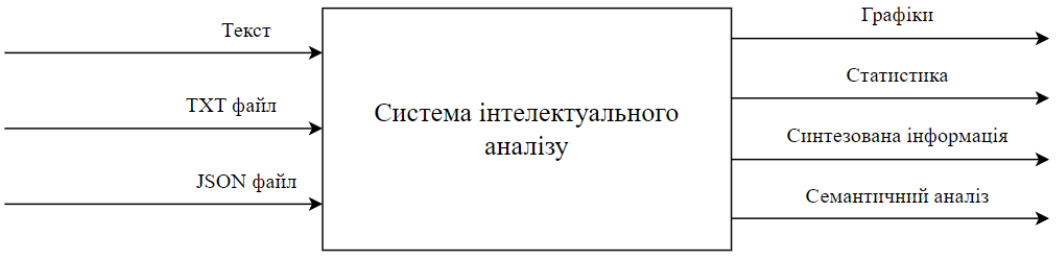


Рисунок 2.5 – Загальна функціональна схема ПЗ

На вхід система може отримувати текст який користувач може ввести в текстове поле, також можна завантажувати TXT файли та JSON (також їх обробляючи). На виході система має візуальні графіки, статистику по тексту, синтезовану інформацію (тема тексту, цільова аудиторія) та семантичний аналіз. Вихід можна також завантажити у вигляді TXT файлу.

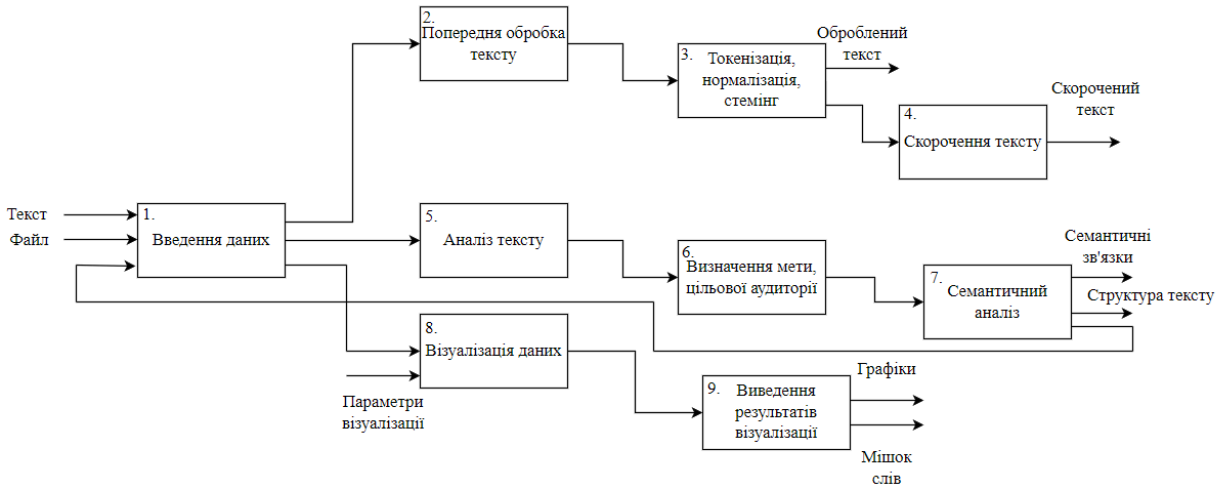


Рисунок 2.6 – Фунціональна схема ПЗ

На рисунку 2.6 зображено функціональну схему програмного забезпечення інтелектуального аналізу тексту. Вона складається з 9 блоків та описує загальний функціонал системи.

Основні етапи роботи ПЗ:

1. Введення даних: користувач вводить текст, який буде оброблено.
2. Попередня обробка тексту: текст проходить через токенізацію, нормалізацію та стемінг.
3. Аналіз тексту: на цьому етапі визначається мета тексту та його цільова аудиторія.
4. Семантичний аналіз: ПЗ виявляє семантичні зв'язки між словами та фразами в тексті.
5. Скорочення тексту: текст може бути скорочений за допомогою спеціальних алгоритмів.
6. Візуалізація даних: текст візуалізується у вигляді графіків.
7. Виведення результатів візуалізації: користувач може бачити результати візуалізації даних.

На вхід система отримує текст або файл, на вихід користувач отримує скорочений текст (прибирання «води» з тексту, викладення основної його суті), сематичні зв’язки в тексті, та його структуру.

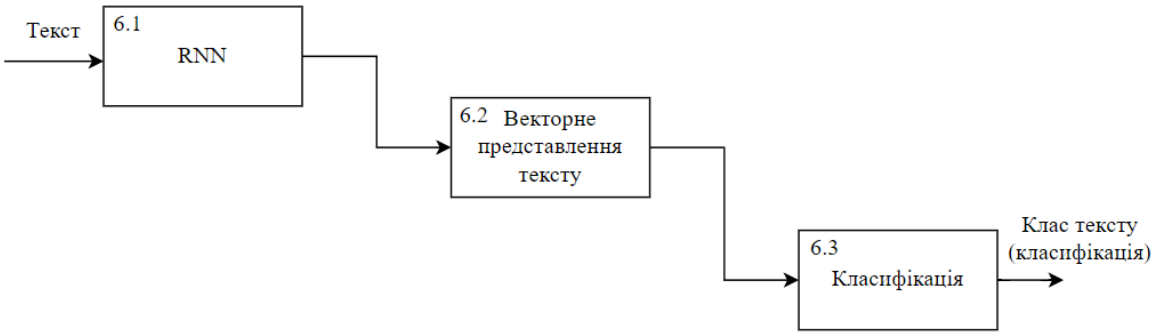


Рисунок 2.7 – Деталізована діаграма визначення мети та цільової аудиторії

На рисунку 2.7 зображена деталізована діаграма визначення мети та цільової аудиторії. Спочатку йде блок самої нейромережі (RNN – рекурентна), Після цього текст потрібно перетворити на векторну форму, яка може бути зрозуміла для нейромережі. Це може бути векторне представлення слів, як TF-IDF вектори, word embeddings, і т.д. Результати виходу з нейромережі повинні бути оброблені для визначення мети тексту та визначення його класу (класифікація). Це може включати класифікацію тексту на категорії та кластеризацію його на групи.

2.2 Розробка та докладний опис алгоритму роботи програми

2.2.1 Опис функціональних та не функціональних вимог до системи

Вимоги можна класифікувати на функціональні та не функціональні. Функціональні вимоги описують, що саме повинна робити система, які функції вона повинна виконувати. Не функціональні вимоги визначають якість, ефективність, безпеку та інші характеристики системи, а не її конкретні функції.

Описані функціональні вимоги для системи інтелектуального аналізу тексту містяться в таблиці 2.1.

Таблиця 2.1 — Функціональні вимоги до системи

№	**Вимога**
1	Семантичний аналіз тексту з метою розуміння його змісту та контексту.
2	Система повинна мати можливість виявляти, чи використовувалися GPT технології для написання конкретного тексту. Це може включати виявлення характерних ознак, які вказують на використання цих технологій.
3	Виявлення та корекція лексичних та орфографічних помилок у тексті.
4	Визначення теми тексту та цільової аудиторії, що може бути корисною для маркетингових стратегій.
5	Система повинна мати можливість побудови хмар слів для візуалізації частоти вживання слів у тексті. Це допомагає виділити ключові терміни та поняття, що зустрічаються у тексті.
6	Генерація графіків для візуалізації структури тексту, наприклад, частота появи термінів у тексті.
7	Система повинна автоматично створювати короткий зміст тексту для швидкого ознайомлення з його змістом. Це допомагає користувачеві отримати загальне уявлення про текст без необхідності читати його повністю.

Описані нефункціональні вимоги для системи інтелектуального аналізу тексту містяться в таблиці 2.

Таблиця 2.2 — Нефункціональні вимоги до системи

№	**Вимога**
1	Система повинна забезпечувати ефективну та швидку обробку текстових даних, незалежно від їх обсягу. Це означає, що час відповіді

	на запити користувачів має бути мінімальним, навіть при обробці великих обсягів інформації.
2	Результати аналізу тексту повинні бути точними та достовірними. Система повинна надійно визначати теми текстів, ідентифікувати цільову аудиторію та виявляти лексичні та семантичні особливості з мінімальною кількістю помилок.
3	Система повинна бути здатна обробляти малі та великі обсяги тексту.
4	Інтегрованість. Система повинна мати можливість легко інтегруватися з іншими інформаційними системами.
5	Система повинна мати зручний та зрозумілий для всіх користувачів інтерфейс
6	Система повинна забезпечувати конфіденційність та цілісність оброблюваної інформації.

2.2.2 Технології та інструменти інтелектуального аналізу тексту

Під час розробки системи інтелектуального аналізу тексту буде використано мову Python 3 та усі небхідні для цього бібліотеки та фреймворки.

Natural Language Toolkit (NLTK) надає повний набір бібліотек для таких завдань, як токенізація, стеммінг, лематизація, тегування частин мови та розпізнавання іменованих об'єктів.

SpaCy пропонує потужні можливості для розширеної лінгвістичної обробки, включаючи синтаксичний аналіз залежностей, розпізнавання об'єктів і сегментацію речень.

Gensim - ще одна цінна бібліотека для моделювання тем, аналізу схожості документів і методів вбудовування слів, таких як Word2Vec і Doc2Vec.

scikit-learn пропонує алгоритми машинного навчання для задач класифікації, кластеризації та регресії тексту, а TensorFlow і PyTorch надають

фреймворки для реалізації моделей глибокого навчання для аналізу тексту, включаючи рекурентні нейронні мережі (RNN).

Таблиця 2.3 — Використані при розробці технології

Технологія/Інструмент	**Опис**
Фреймворк	Flask
Бібліотеки	NLTK (Natural Language Toolkit) для обробки тексту
	spaCy для обробки тексту та виявлення сутностей
	Gensim для роботи з векторними моделями
	scikit-learn для машинного навчання
	Matplotlib та Seaborn для візуалізації даних
Система контролю версій	Git
Інтегроване середовище	PyCharm
Система управління базами даних	PostgreSQL

2.2.3 Алгоритми попередньої обробки тексту

Завдання попередньої обробки тексту перед аналізом полягає в підготовці вхідних даних для подальшого ефективного використання в алгоритмах аналізу тексту. Цей етап включає в себе ряд кроків, що спрямовані на очищення, структурування та стандартизацію текстових даних. Для цього можуть бути використані різноманітні методи та інструменти, які допомагають зберегти інформаційну цінність тексту та зменшити об'єм даних для подальшого аналізу.

Основні етапи попередньої обробки тексту на мові програмування Python включають в себе:

Токенізація - сфері обробки природної мови (NLP) означає процес перетворення послідовності тексту на менші частини, відомі як токени. Ці токени можуть бути як маленькими, як символи, так і довгими, як слова. Для цього можна використовувати бібліотеку NLTK (Natural Language Toolkit).

NTLK - це набір бібліотек і програм для символьної та статистичної обробки природної мови (NLP) для англійської мови, написаний мовою програмування Python. Він підтримує функції класифікації, токенізації, стемінгу, тегування, синтаксичного аналізу та семантичного міркування.

Приклад використання для токенізації може бути наступним:

```
1  import nltk
2  from nltk.tokenize import word_tokenize
3
4  text = "Example for tokenization. By Chepurko Maxim"
5  tokens = word_tokenize(text)
6  print(tokens)
```

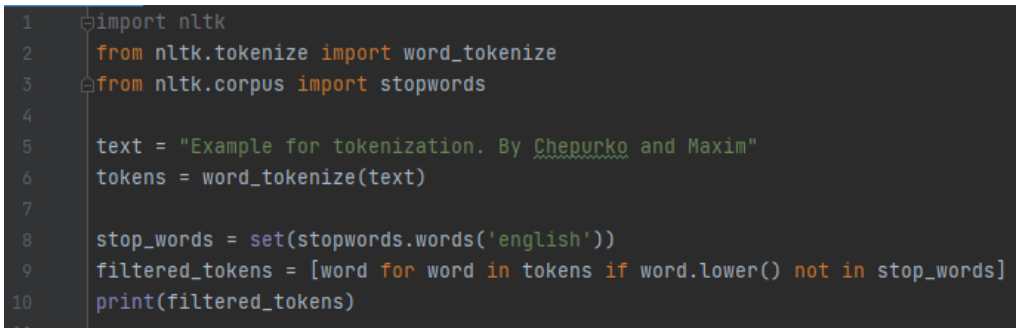
Рисунок 2.8 – Приклад використання токенізації

```
D:\pythonProject16\venv\Scripts\python.exe D:/pythonProject16/main.py
['Example', 'for', 'tokenization', '.', 'By', 'Chepurko', 'Maxim']

Process finished with exit code 0
```

Рисунок 2.9 – Приклад виводу токенізації

Видалення стоп-слів - це процес видалення найбільш поширених слів, які не несуть значущої інформації для аналізу (таких як "і", "в", "на" тощо). Використовуються списки стоп-слів, які можуть бути визначені на основі мови тексту. Для цього можна також використовувати бібліотеку NTLK (corpus).

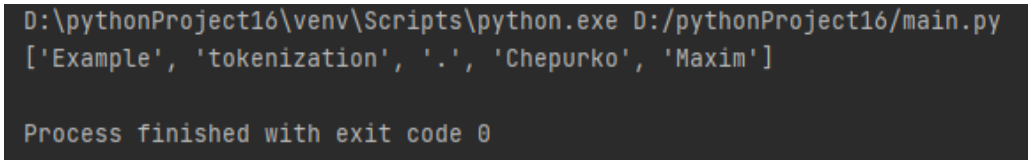


```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

text = "Example for tokenization. By Chepurko and Maxim"
tokens = word_tokenize(text)

stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print(filtered_tokens)
```

Рисунок 2.10 – Приклад використання видалення стоп-слів



```
D:\pythonProject16\venv\Scripts\python.exe D:/pythonProject16/main.py
['Example', 'tokenization', '.', 'Chepurko', 'Maxim']

Process finished with exit code 0
```

Рисунок 2.11– Приклад виводу при видаленні стоп-слів

Лематизація - одна з найпоширеніших технік попередньої обробки тексту, яка використовується в обробці природної мови та машинному навчанні загалом. Лематизація не надто відрізняється від стеммінгу слів у NLP. І при лематизації, і при стеммінгі ми намагаємося звести задане слово до його кореневого слова (до базової форми). У процесі стеммінгу кореневе слово називається основою, а в процесі лематизації воно називається лемою.

Для цього процессу можна використовувати також бібліотеку NTLK, NTLK.stem - інтерфейс для видалення морфологічних афіксів зі слів, залишаючи лише основу слова. Алгоритми вилучення афіксів спрямовані на вилучення афіксів, необхідних для визначення граматичної ролі, часу, деривaційної морфології, залишаючи лише основу слова.

Лекматизація бере слово і розбиває його на леми. Наприклад, дієслово "ходити" може виглядати як "ходжу", "ходить" або "ходив".

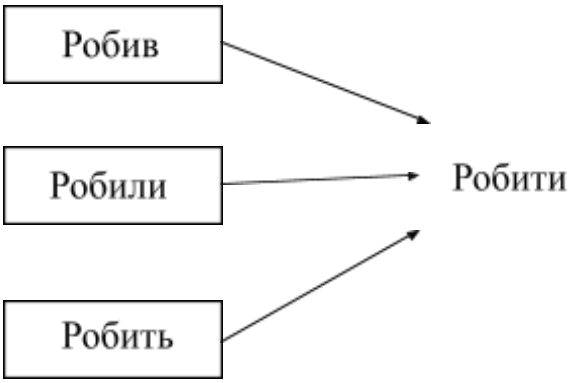


Рисунок 2.12 – Приклад лемматизації

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')

text = "Examples for tokenization. By Chepurko and Maxim. More words for example."
tokens = word_tokenize(text)
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

print(lemmatized_tokens)
```

Рисунок 2.13 – Приклад використання лемматизації

```
['Example', 'tokenization', '.', 'Chepurko', 'Maxim', '.', 'word', 'example', '.']

Process finished with exit code 0
```

Рисунок 2.14 – Приклад виводи після лемматизації

Чистка тексту від символів пунктуації та інших символів. У тексті можуть міститися символи пунктуації, які можуть перешкоджати аналізу. Їх можна видалити за допомогою регулярних виразів.

Регулярний вираз - це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил.

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')

text = "Example for tokenization. By Chepurko and Maxim"
text = re.sub(r'[^\w\s]', '', text)
tokens = word_tokenize(text)

stop_words = set(stopwords.words('english'))

filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

print(lemmatized_tokens)
```

Рисунок 2.15 – Приклад видалення непотрібних символів за допомогою регулярного виразу [^\w\s]

```
['Example', 'tokenization', 'Chepurko', 'Maxim']

Process finished with exit code 0
```

Рисунок 2.16 – Приклад виводу після використання регулярного виразу

2.3 Опис та порівняння алгоритмів аналізу семантики тексту

Семантичний аналіз - це процес вилучення сенсу з тексту. Він дозволяє комп'ютерам розуміти та інтерпретувати речення, абзаци або цілі документи, аналізуючи їхню граматичну структуру та визначаючи зв'язки між окремими словами в певному контексті.

Це важлива підзадача обробки природної мови (NLP) і рушійна сила інструментів машинного навчання, особливо це важливо в інтелектуальному аналізи тексту.

Лексична семантика відіграє важливу роль у семантичному аналізі, дозволяючи машинам розуміти зв'язки між лексичними елементами (словами, фразовими дієсловами тощо), розглянемо приклади (в англійській мові):

Гіпоніми - видові лексеми родової лексеми (гіпероніми), наприклад, апельсин є гіпонімом фрукта (гіперонімом).

Меронімія - логічне розташування тексту і слів, що позначає складову частину або член чогось, напр., сегмент апельсина

Полісемія - зв'язок між значеннями слів або словосполучень, які хоч і дещо відрізняються, але мають спільне основне значення, напр., I read a paper, and I wrote a paper)

Синоніми - слова, що мають те саме або майже те саме значення, що й інші, напр., happy, content, ecstatic, overjoyed

Антоніми - слова, які мають близькі або протилежні значення, наприклад, happy, sad

Омоніми - два слова, які звучать однаково і пишуться однаково, але мають різне значення, наприклад, orange (колір), orange (фрукт).

Семантичний аналіз також враховує знаки та символи (семіотику) і словосполучення (слова, які часто вживаються разом). Автоматизований семантичний аналіз працює за допомогою алгоритмів машинного навчання.

В залежності від типу інформації, яку потрібно отримати з даних, можна використати один з двох методів семантичного аналізу: модель класифікації тексту (яка присвоює тексту заздалегідь визначені категорії) або екстрактор тексту (який витягує з тексту певну інформацію).

2.3.1 Латентний семантичний аналіз (LSA)

Латентний семантичний аналіз (LSA), також відомий як приховане семантичне індексування (LSI), - це метод в обробці природної мови, який виявляє приховану структуру в колекції текстів. Він використовується, зокрема, для зменшення розмірності та пошуку взаємозв'язків між термінами і документами.

Приклад: “mobile“, “phone”, “telephone” - всі ці слова схожі, але якщо ми задамо запит на кшталт " The cell phone has been ringing", то будуть знайдені тільки ті документи, в яких є "cell phone", тоді як документи, що містять "mobile", "phone", "telephone", не будуть знайдені.

Сингулярна декомпозиція (SVD) використовується у методі LSA для аналізу семантики тексту. Вона допомагає зменшити розмірність матриці термін-документ, розкладаючи її на три більш прості компоненти: дві ортогональні матриці та діагональну матрицю з сингулярними значеннями. Це дозволяє отримати векторні представлення слів та документів у просторі меншої розмірності, що полегшує аналіз семантики тексту та виявлення семантичних відносин між ними. Припустимо що:

C = колекція документів.

d = кількість документів.

n = кількість унікальних слів у всій колекції.

M = d X n

SVD розкладає матрицю M, тобто матрицю "слово-документ", на три матриці наступним чином:

$$M = U\sum VT$$

де:

U = розподіл слів у різних контекстах,

$\sum$ = діагональна матриця асоціацій між контекстами,

VT = розподіл контекстів у різних документах.

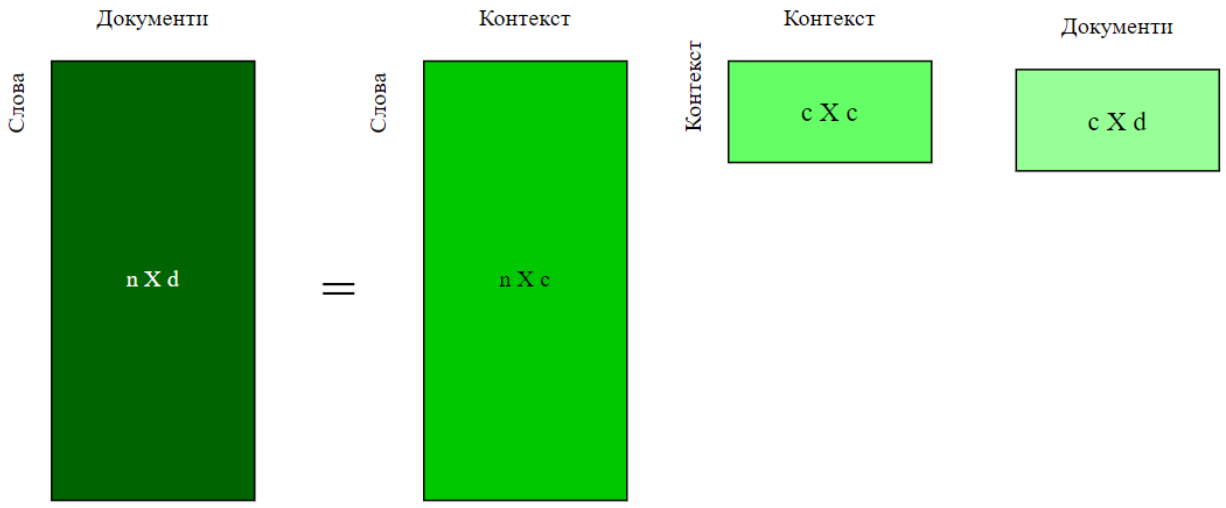


Рисунок 2.17 – Приклад SVD для матриці n x d

У контексті інтелектуального аналізу тексту і методу LSA (Latent Semantic Analysis), сингулярний розклад (SVD) використовується для зменшення розмірності матриці термін-документ і виявлення семантичних відносин між словами та документами.

Розглянемо приклад. Маємо матрицю термін-документ, де кожен рядок представляє слово, а кожний стовпчик представляє документ, і кожна комірка містить частоту зустрічання слова в документі. Наприклад:

	Doc1	Doc2	Doc3
Word1	2	0	1
Word2	1	1	3
Word3	0	2	2

Можна застосувати SVD до цієї матриці, щоб отримати два набори векторів: один для слів і один для документів. Ці вектори представляють "приховані" семантичні властивості слів і документів.

Наприклад, після застосування SVD можна отримати такі вектори для слів та документів:

	Concept1	Concept2
Word1	0.3	0.6
Word2	0.7	0.4
Word3	0.5	0.3

	Doc1	Doc2	Doc3
Concept1	0.4	0.2	0.6
Concept2	0.5	0.6	0.3

Тепер можна використовувати ці вектори для визначення семантичної подібності між словами та документами. Наприклад, слова "Word1" та "Word2" мають більше подібний семантичний зміст, оскільки їхні вектори ближчі один до одного у просторі концепцій.

Тому, що до LSA можна зробити наступні висновки:

1. LSA є класичним методом для аналізу семантики тексту, який базується на матричному розкладі термін-документ.
2. Використовуючи сингулярний розклад (SVD) матриці термін-документ, LSA створює векторні представлення слів та документів у просторі меншої розмірності.
3. Векторні представлення дозволяють виявляти семантичні відношення між словами та документами, що допомагає у розумінні їхнього значення.

2.3.2 Латентиний розподіл Діріхле (LDA)

Latent Dirichlet Allocation (LDA) - це метод тематичного моделювання, який дозволяє виявляти тематичні структури в наборі текстових даних.

Основна ідея LDA полягає в тому, що кожен документ може бути розглянутий як сукупність тем з певними ймовірностями, а кожна тема представлена як розподіл слов з певними ймовірностями.

Розглянемо приклад. Маємо набір документів, що стосуються новин про технології, спорт та культуру. Застосувавши LDA до цих даних, ми можемо отримати такі теми та їхній розподіл слів:

1. Технології: "комп'ютер", "інтернет", "програмне забезпечення"
2. Спорт: "футбол", "баскетбол", "теніс"
3. Культура: "мистецтво", "музика", "література"

Тепер, кожен документ може бути представлений як сукупність цих тем з певними ймовірностями. Наприклад, якщо документ про футбол, ймовірність належності до теми "спорт" буде висока, а імовірність належності до тем "технології" та "культура" буде низькою.

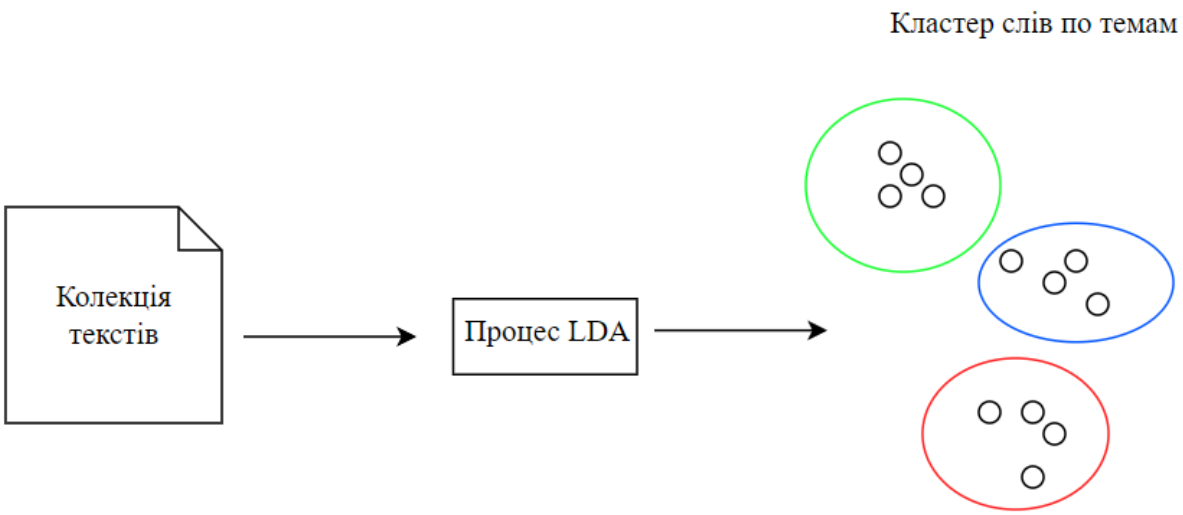


Рисунок 2.18 – Приклад роботи LDA

Таким чином, LDA допомагає розкрити тематичну структуру великих колекцій текстових даних, що дозволяє використовувати цю інформацію для подальшого аналізу, категоризації та розуміння текстів.

Тому, можна зробити висновки що LSA (Latent Semantic Analysis) та LDA (Latent Dirichlet Allocation) - це два різних методи для аналізу семантики тексту. LSA базується на сингулярному розкладі матриці термін-документ,

тоді як LDA використовує ймовірнісний підхід для визначення тематик у текстах.

Основними плюсами LSA є те що він добре підходить для виявлення семантичних відносин між словами та документами та також відносно простий у реалізації та розумінні. Але не дуже ефективний для виявлення тематик у текстах, оскільки не враховує ймовірнісний аспект виникнення слів у документах.

LDA в свою чергу здатний визначати тематичну структуру текстів за допомогою ймовірнісного моделювання та також дозволяє виявляти складні тематичні зв'язки у тексті. Але вимагає більш складного обчислювального підходу та більшої кількості обчислювальних ресурсів.

2.4 Розробка інтерфейсу програми

Інтерфейс веб-сторінки має наступну структуру:

- Header (Шапка):
    - Заголовок "Система інтелектуального аналізу тексту", що також є посиланням на головну сторінку.
    - Навігаційне меню з посиланнями на "Головну", "Про програму", "Контакти" і "FAQ".
- Main (Основний контент):
    - Розділ "Введіть текст для аналізу або завантажте файл":
        - Заголовок "Введіть текст для аналізу або завантажте файл".
        - Текстове поле для введення тексту з можливістю введення або редагування тексту.
        - Поле вибору файлу для завантаження текстового файлу.
        - Кнопка "Аналізувати" для запуску аналізу тексту.
- Розділ "Функціонал":
    - Заголовок "Функціонал".

- Кнопки для виклику різних функцій аналізу: "Попередня обробка тексту", "Аналіз тексту", "Аналіз помилок" (з випадаючим списком лексичних та орфографічних помилок), "Семантичний аналіз" і "Візуалізація результатів".

- Розділ "Результати аналізу":
  - Заголовок "Результати аналізу".
  - Зона для відображення результатів аналізу тексту. Ця область буде використовуватися для відображення виходу з різних функцій аналізу.

Приклад інтерфейсу має наступний вигляд (це лише макет, фінальна версія ПЗ може доповнюватися, покращуватися та виглядати по іншому):

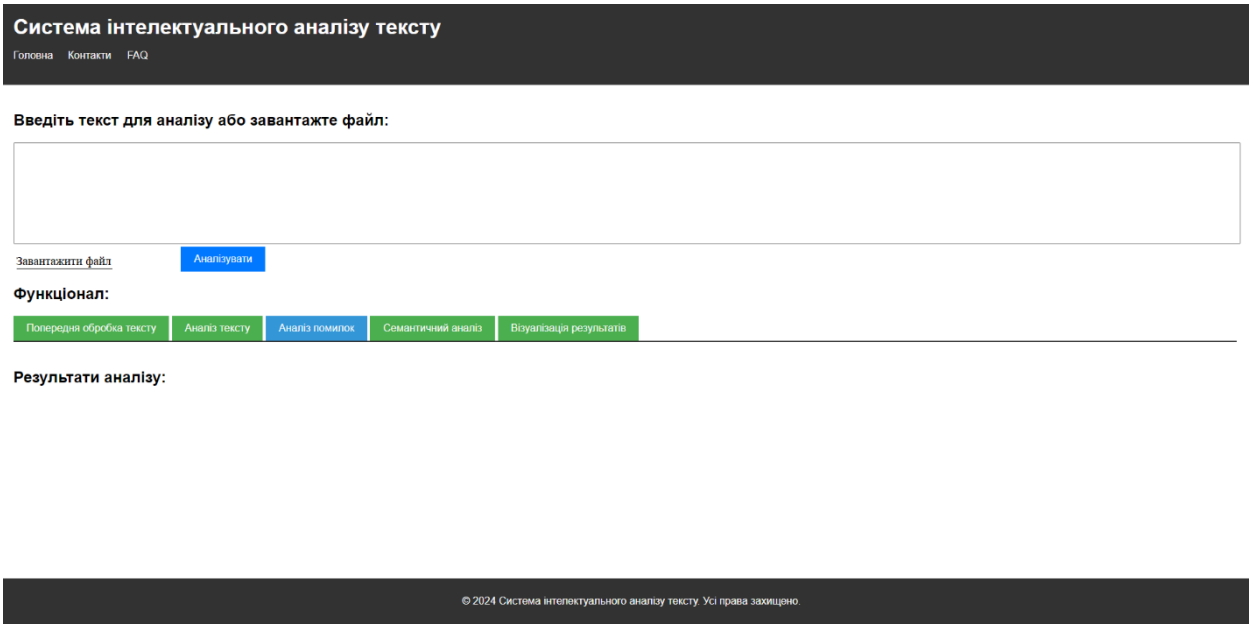


Рисунок 2.19 – Приклад реалізації інтерфейсу

Деякі пункти меню (наприклад «аналіз помилок») будуть мати підпункти, виглядати це має наступним чином:

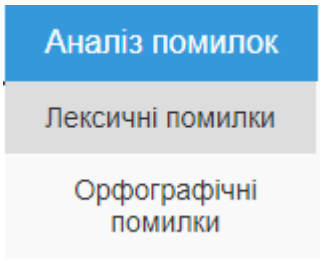


Рисунок 2.20 – Приклад підпунктів основних пунктів меню

Інтерфейс веб-сайту буде адаптуватися під різні розширення та девайси, приклад вигляду інтерфейсу веб-сайту на мобільному пристрої зображено на рисунку 2.21.

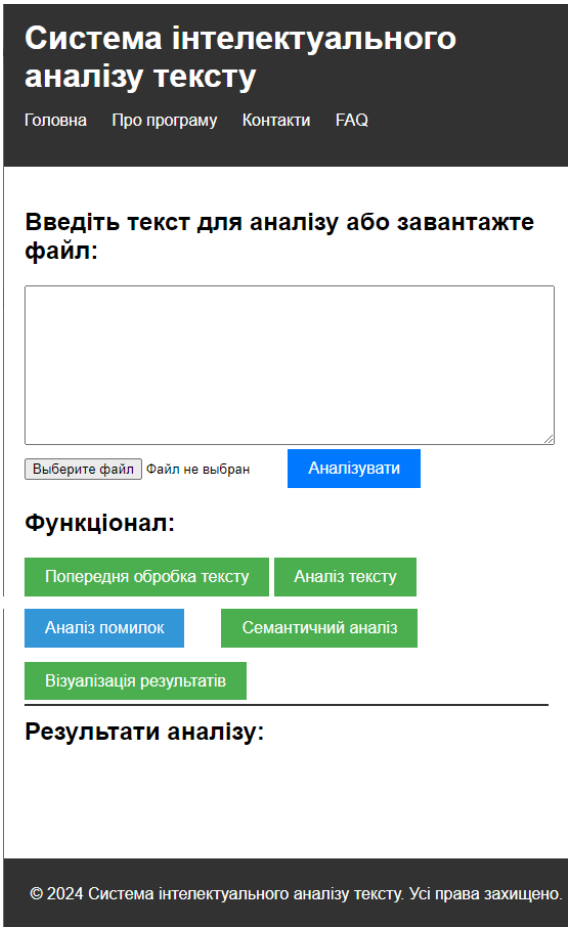


Рисунок 2.21 – Приклад вигляду інтерфейсу веб-сайту при відображенні на екрані мобільного девайсу

Логотип веб-сайту зображений на рисунку 2.22 буде мати наступний вигляд:

Рисунок 2.22 – Логотип веб-сайту системи інтелектуального аналізу

Головна суть логотипу – схожість з мозком. Цей образ символізує складність та глибину аналізу тексту. А також представляє шлях, який система проходить через текст, щоб знайти його сенс, закономірності, помилки і т.д.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Для реалізації програмної частини була вибрана мова програмування Python.

Python - це високорівнева, універсальна, інтерпретована об'єктно-орієнтована мова програмування.

Python відомий тим, що він потужний, швидкий і зрозумілий. Програмуючи на Python можна динамічно вводити змінні без необхідності пояснювати, якою має бути змінна. Вихідний код Python знаходиться у вільному доступі і відкритий для модифікації та повторного використання.

Python широко використовується через його зрозумілий синтаксис та зручність для читання. Часто використовується в аналітиці даних, машинному навчанні (ML) та веб-розробці, Python дає код, який легко читати, розуміти та вивчати.

Саме тому Python обраний для реалізації цього проекту. Через його високий рівень простоти та читабельності коду, що сприяє швидкому розвитку і підтримці проекту на будь-якій стадії.

Крім того, велика спільнота розробників Python та наявність багатофункціональних бібліотек для обробки текстових даних дозволяють швидко і ефективно реалізувати алгоритми аналізу тексту.

В цілому, вибір Python став оптимальним для створення системи інтелектуального аналізу тексту, оскільки він поєднує в собі простоту використання, ефективність та широкий спектр можливостей.

Середа розробки PyCharm була вибрана виходячи з декілької обгрунтованих причин.

По-перше, PyCharm є потужним інтегроване середовищем розробки (IDE) спеціально для Python, що надає широкі можливості автоматичного завершення коду, рефакторингу, вбудованих інструментів аналізу коду та

підтримки віртуальних середовищ. Його інтуїтивний інтерфейс та набір корисних функцій значно полегшують розробку і підтримку проектів Python.

Крім того, PyCharm інтегрується з рядом інших інструментів, таких як системи контролю версій (наприклад, Git), управління пакетами інструментів (наприклад, pip), інструменти для тестування та відлагодження коду. Це забезпечує зручне та ефективне управління проектом з одного місця.

Інтерфейс PyCharm зображений на рисунку 3.1

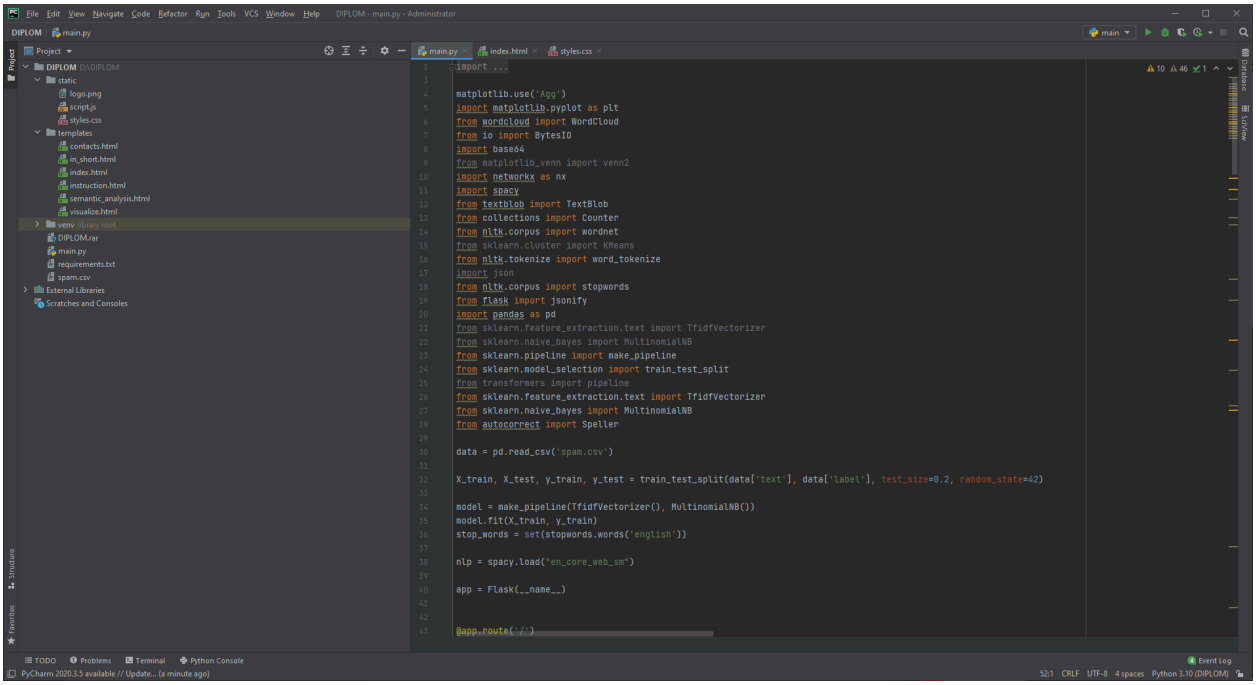


Рисунок 3.1– Інтерфейс середи програмування PyCharm

Python має велику кількість зручних бібліотек, під час реалізації були використані наступні:

Таблиця 3.1 – Використані бібліотеки Python під час розробки

Бібліотека	**Опис**
Flask	Мінімалістичний веб-фреймворк для створення веб-сервера та рендерингу сторінок.
matplotlib	Бібліотека для побудови графіків та діаграм.

WordCloud	Використовується для побудови хмар слів на основі тексту.
io, BytesIO, base64	Використовуються для обробки зображень та їх відображення на веб-сторінці.
matplotlib_venn, networkx	Використовуються для роботи з графами та візуалізації даних.
spacy	Бібліотека для обробки природної мови (Natural Language Processing, NLP).
TextBlob	Бібліотека для обробки тексту та аналізу сентименту.
nltk	Бібліотека для обробки тексту, включаючи токенізацію, стеммінг та інші завдання.
scikit-learn (sklearn)	Бібліотека для машинного навчання та аналізу даних.
pandas	Бібліотека для роботи з даними у вигляді таблиць.
transformers	Бібліотека для роботи з моделями глибокого навчання для обробки природної мови.
autocorrect	Використовується для автоматичного коректування орфографічних помилок.

3.2 Розробка структури проекту

Проект складається з декількох папок, html, js, css, py та csv файлів та зображений на рисунку 3.2.

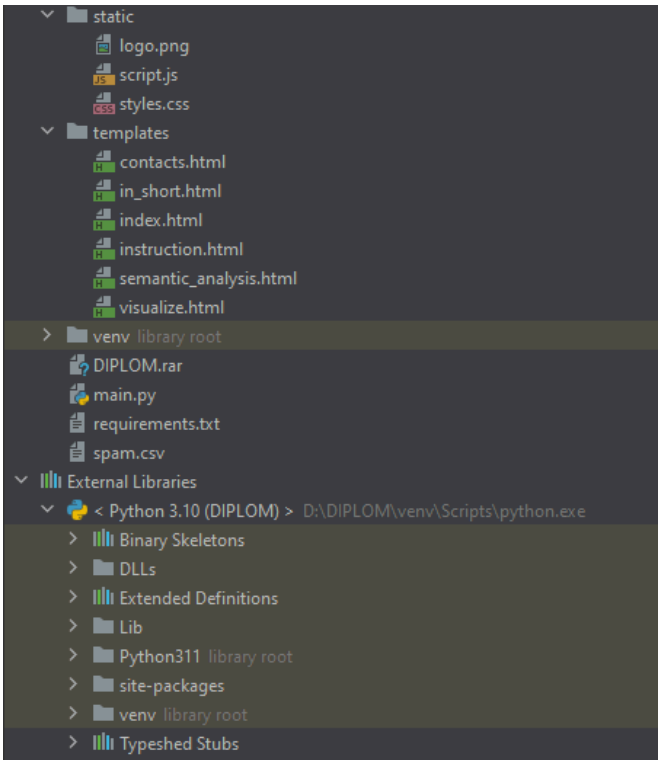


Рисунок 3.2 – Структура проекту ПЗ

Структура проекту виглядає наступним чином:

Static – папка в якій зберігаються основні статичні файли проекту.

- Logo.png – логотип сайту.
- Script.js – файл з усім Javascript кодом.
- Styles.css – файл з каскадними стилями для HTML розмітки.

Templates – папка в якій знаходяться HTML сторінки проекту.

- Contacts.html – сторінка «контактів».
- In_short.html – сторінка необхідна для відображення скороченого та узагальненого тексту.
- Index.html – головна сторінка.
- Instruction.html – сторінка на якій зберігається опис функціоналу веб-сайту.
- Semantic_analysis.html – сторінка необхідна для відображення семантичного аналізу тексту.
- Visualize.html – сторінка на якій відображається візуалізація тексту.

Main.py – головний файл в якому зберігається реалізація основного функціоналу сайту.

Requirements.txt – файл в якому зберігаються назви необхідних бібліотек для запуску проекту.

Spam.csv – датасет необхідний для виявлення спаму в тексті.

Основний файл main.py також має свою структуру.

Спочатку імпортуються необхідні бібліотеки, це зображено на рисунку 3.3.

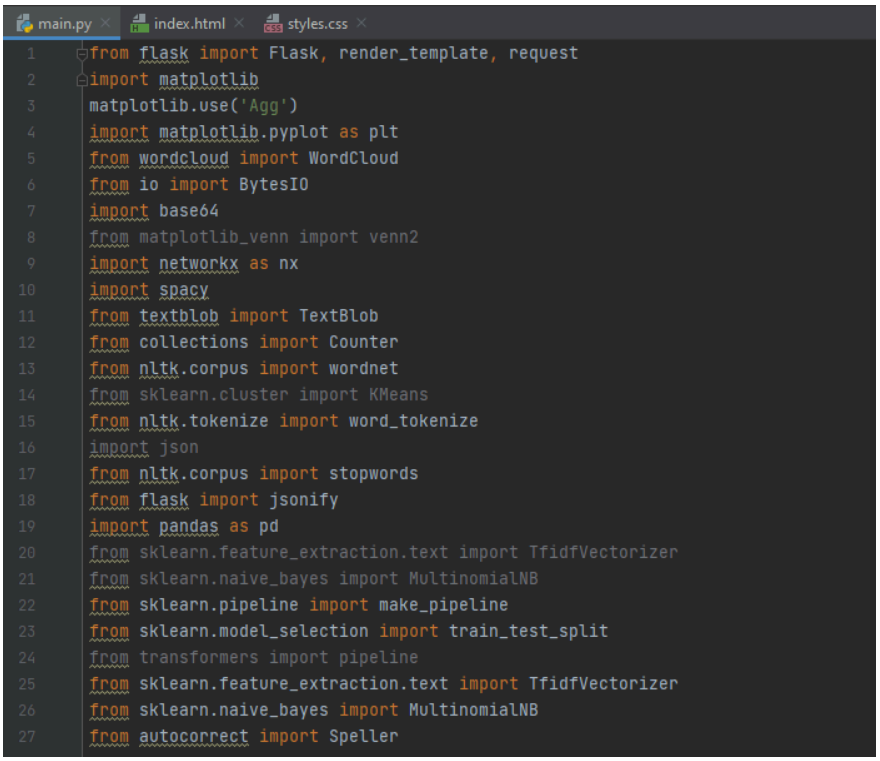


Рисунок 3.3 – Імпортування необхідних бібліотек

Після чого відбувається навчання моделі на датасеті для аналізу тексту.

```
data = pd.read_csv('spam.csv')

X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(X_train, y_train)
stop_words = set(stopwords.words('english'))

nlp = spacy.load("en_core_web_sm")

app = Flask(__name__)
```

Рисунок 3.4 – Навчання на датасеті

Потім кожна сторінка веб-сайту та кожна реалізована функція описана за допомогою команди app.route(«»).

```
@app.route('/fixText', methods=['POST'])
def fix_text():
    data = request.get_json()
    text = data['text']
    corrected_text = spell(text)

    return jsonify({'correctedText': corrected_text})
```

Рисунок 3.5 – Приклад реалізації функції fixText

3.3 Програмна реалізація основних функцій

3.3.1 Програмна реалізація функцій на Python

Основні функції реалізовані під час розробки фінальної версії ПЗ мають наступний вигляд:

Таблиця 3.2 — Короткий опис основних функцій

Функція	Опис
index	Повертає головну сторінку за допомогою шаблону 'index.html'.
contacts	Повертає сторінку контактів за допомогою шаблону 'contacts.html'.

spamHam	Обробляє POST-запит і передбачає, чи є вхідний текст спамом чи ні.
in_short	Обробляє POST-запит і повертає коротке основне речення з вхідного тексту.
fix_text	Обробляє POST-запит і виправляє помилки в тексті.
check_errors	Обробляє POST-запит і перевіряє наявність помилок у вхідному тексті.
instruction	Повертає сторінку з інструкціями за допомогою шаблону 'instruction.html'.
semantic_analysis	Обробляє POST-запит і проводить семантичний аналіз тексту, повертає результати та візуалізацію за допомогою шаблону 'semantic_analysis.html'.
extract_keywords	Обробляє POST-запит і витягує ключові слова з вхідного тексту, повертає топ-3 ключових слова та головну тему.
visualize_text	Обробляє POST-запит і візуалізує текст у вигляді графіків та діаграм, повертає зображення у вигляді HTML рядка.

Розглянемо детальніше логіку реалізації кожної функції.

В самому початку, для подальшого використання деяких функцій потрібно на основі датасету навчити нейронну модель для того щоб вона могла визначати чи є текст спамом чи ні.

Реалізується це наступним чином:

```
data = pd.read_csv('spam.csv')
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(X_train, y_train)
stop_words = set(stopwords.words('english'))
nlp = spacy.load("en_core_web_sm")
```

pd.read_csv('spam.csv') - зчитує дані з CSV файлу з назвою "spam.csv" та завантажує їх у DataFrame. Передбачається, що файл містить колонки "text" з

текстом повідомлень та "label" з мітками, які вказують, чи є повідомлення спамом чи ні.

train_test_split(data['text'], data['label'], test_size=0.2, random_state=42) - розбиває дані на навчальний та тестовий набори. Параметр test_size=0.2 вказує, що 20% даних буде використано для тестування, а решта - для навчання моделі. random_state=42 гарантує відтворюваність розділення даних при кожному запуску програми.

make_pipeline(TfidfVectorizer(), MultinomialNB()) - створюється конвеєр, який поєднує TfidfVectorizer та MultinomialNB модель. TfidfVectorizer перетворює тексти у числові представлення за допомогою TF-IDF, а MultinomialNB - навчається розпізнавати спам.

Після цього потрібно імпортувати деякі елементи необхідні для обробки тексту.

```
stop_words = set(stopwords.words('english'))
nlp = spacy.load("en_core_web_sm")
```

stop_words = set(stopwords.words('english')) - встановлюється множина стоп-слів для англійської мови. Стоп-слова - це слова, які вважаються незначущими для аналізу тексту, такі як "the", "is", "at" і т.д. Вони виключаються з аналізу, оскільки вони зазвичай не несуть значущої інформації про зміст тексту.

nlp = spacy.load("en_core_web_sm") - Ініціалізується обробник природної мови (NLP) для англійської мови з використанням моделі "en_core_web_sm" з бібліотеки spaCy. Цей обробник NLP дозволить виконувати різноманітні завдання обробки тексту, такі як токенізація, розпізнавання частин мови, екстрагування іменованих сутностей та багато іншого.

До кожної сторінки веб-сайту прописаний route:

```
@app.route('/')
```

```
def index():
    return render_template('index.html')

@app.route('/contacts')
def contacts():
    return render_template('contacts.html')
```

Цей код використовує Flask. Він встановлює дві URL-адреси - "/" та "/contacts", кожна з яких відповідає певній сторінці. При запиті до URL-адреси "/" відбувається виклик функції index(), яка відображає сторінку "index.html". Аналогічно, при запиті до URL-адреси "/contacts" викликається функція contacts(), яка відображає сторінку "contacts.html".

Далі буде описано програмну реалізацію основних функцій.

```
@app.route('/spamHam', methods=['POST'])
def spamHam():
    text = request.form['text']
    prediction = model.predict([text])[0]
    return jsonify({'prediction': prediction})
```

Цей код налаштовує обробку POST-запитів до URL-адреси "/spamHam". Коли отримує POST-запит, він отримує текст з форми запиту, передбачає, чи це "спам" чи "не спам" (на основі моделі model spam.csv), та повертає результат у форматі JSON.

Функція скорочення має таку структуру:

```
@app.route('/inShort', methods=['POST'])
def in_short():
    data = request.get_json()
    text = data['text']

    def in_short_logic(text):
        sentences = text.split('.')
        main_sentence = ""
        max_polarity = -1
        for sentence in sentences:
            if len(sentence.strip()) > 0:
```

```
                    polarity = TextBlob(sentence).sentiment.polarity
                    if polarity > max_polarity:
                        main_sentence = sentence
                        max_polarity = polarity

            return main_sentence

        short_text = in_short_logic(text)

        return jsonify({'shortText': short_text})

    spell = Speller(lang='en')
```

Спочатку виконується обробка POST-запитів до URL-адреси "/inShort". Коли сервер отримує POST-запит, він отримує дані у форматі JSON, розпаковує текст з цих даних і передає його до функції in_short_logic().

Функція in_short_logic(text) виконує наступні кроки:

- Розбиває текст на речення, використовуючи роздільник ".".
- Після цього ініціалізує змінні main_sentence (головне речення) та max_polarity (максимальна полярність) зі значенням -1.
- Проходиться по кожному реченню. Якщо речення має довжину більше 0 (тобто не є пустим рядком), обчислює його полярність за допомогою TextBlob та порівнює з максимальною полярністю. Якщо поточна полярність більша за максимальну, змінює головне речення та оновлює максимальну полярність.
- На кінці повертає головне речення з максимальною полярністю.

Логіка реалізації сементичного аналізу тексту має наступний вигляд:

```
@app.route('/semanticAnalysis', methods=['POST'])
def semantic_analysis():
    text = request.form['text']
    doc = nlp(text)
    semantic_results = []
    overall_sentiment = 0
    all_keywords = []

    for sent in doc.sents:
```

```
            sentiment_score = TextBlob(sent.text).sentiment.polarity
                sentiment_label = "Позитивний" if sentiment_score > 0 else
"Негативний" if sentiment_score < 0 else "Нейтральний"
            overall_sentiment += sentiment_score
            entities = [(ent.text, ent.label_) for ent in sent.ents]
             keywords = [token.text for token in sent if not token.is_stop and
token.pos_ in ['NOUN', 'VERB', 'ADJ']]
            semantic_results.append({
                "sentence": sent.text,
                "sentiment": sentiment_label,
                "sentiment_score": sentiment_score,
                "entities": entities,
                "keywords": keywords
            })
            all_keywords.extend(keywords)

                    top_keywords = [keyword for keyword, _ in
Counter(all_keywords).most_common(5)]

          overall_sentiment_label = "Позитивний" if overall_sentiment > 0 else
"Негативний" if overall_sentiment < 0 else "Нейтральний"

                           return render_template('semantic_analysis.html',
results=semantic_results,
                                  overall_sentiment=overall_sentiment_label,
top_keywords=top_keywords)
```

Основні кроки, які відбуваються в цій функції:

- Отримання тексту та створення об'єкта nlp. Починаючи з отримання тексту з POST-запиту, функція використовує бібліотеку spaCy для ініціалізації об'єкта обробника nlp. Цей об'єкт відповідає за обробку тексту та розпізнавання лінгвістичних структур, таких як слова, речення, та сутності.
- Кожне речення тексту проходить через ітераційний цикл, в якому обчислюється настрій речення за допомогою TextBlob, а також виявляються ключові слова. Для цього використовується властивість sentiment.polarity об'єкта TextBlob, яка визначає ступінь позитивності або негативності тексту. Ключові слова визначаються за допомогою

spaCy, враховуючи лише іменники, дієслова та прикметники, і виключаючи займенники та службові слова.

- Результати аналізу кожного речення, включаючи текст речення, настрій, оцінку настрою, виявлені сутності та ключові слова, зберігаються в списку semantic_results. Також розраховується загальний настрій тексту, який обчислюється як сума настроїв кожного речення.
- Після обробки всіх речень тексту, визначаються найпоширеніші ключові слова. Це реалізовано за допомогою лічильника Counter, який підраховує кількість кожного ключового слова у тексті.
- Загальний настрій тексту, список результатів семантичного аналізу кожного речення та найпоширеніші ключові слова передаються у шаблон "semantic_analysis.html" для відображення на веб-сторінці.

Рисунок 3.6 – Кроки семантичного аналізу

Візуалізація реалізована за допомогою перетворення зображення в png формат та виведення його на сторінку за допомогою тега та <img>.

```
plot_url1 = base64.b64encode(img1.getvalue()).decode()
plot_url2 = base64.b64encode(img2.getvalue()).decode()
plot_url3 = base64.b64encode(img3.getvalue()).decode()
plot_url4 = base64.b64encode(img4.getvalue()).decode()
```

```
                    return '<div style="margin-bottom: 20px;"><img
src="{}" style="display:block; margin:auto;"></div><div
style="margin-bottom: 20px;"><img src="{}"
style="display:block; margin:auto;"></div><div style="margin-bottom:
20px;"><img src="{}" style="display:block;
margin:auto;"></div><div><img src="{}"
style="display:block; margin:auto;"></div>'.format(
            plot_url1, plot_url2, plot_url3, plot_url4)
```

Цей код генерує HTML-рядок з чотирма вбудованими зображеннями у форматі PNG. Використовуючи бібліотеку base64, зображення перетворюються у рядки, які потім вставляються безпосередньо в HTML-код як дані URL. Кожне зображення вбудовано в тег <img> з атрибутом src, що містить дані у форматі base64. Всі зображення розташовані у блоках з певними стилізованими властивостями для відображення на сторінці.

Приклади візуалізації зображені на рисунках 3.7, 3.8, 3.9, 3.10.

Рисунок 3.7 – Візуалізація частоти зустрічі слів

Рисунок 3.8 – Візуалізація хмари слів

Рисунок 3.9 – Візуалізація графу слів

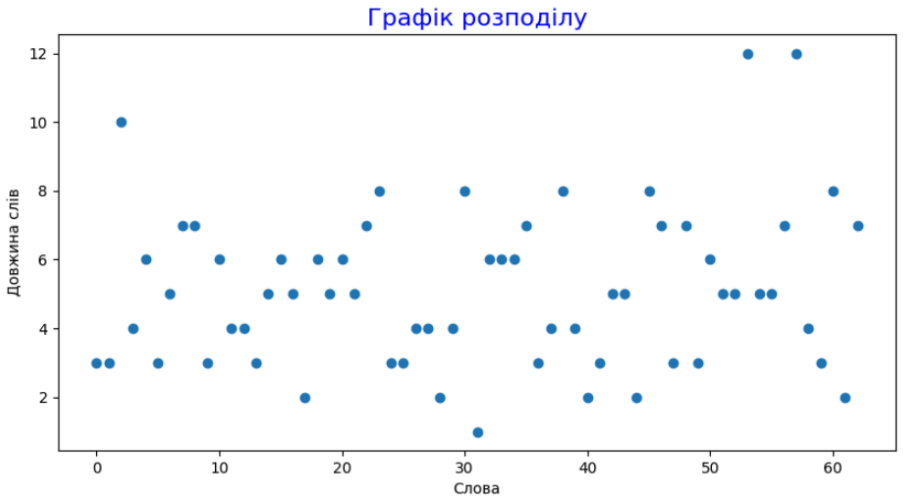


Рисунок 3.10 – Візуалізація графіку розподілу слів

3.3.2 Програмна реалізація функцій на Javascript

Частина функціоналу реалізована на мові програмування Javascript, нижче описано реалізацію:

Таблиця 3.3 — Короткий опис основних функцій реалізованих за допомогою Javascript

Функція	Опис
textAnalysis()	Аналізує введений текст, обчислює кількість символів, слів, унікальних слів, речень, середню довжину слова, частоту вживання слів та знаків пунктуації. Виводить результати на сторінку.
visualizeText()	Відправляє текст на сервер для візуалізації та виводить результати на сторінку за допомогою XMLHttpRequest.
semanticAnalysis()	Відправляє текст на сервер для семантичного аналізу та виводить результати на сторінку за допомогою XMLHttpRequest.
checkErrors()	Відправляє текст на сервер для перевірки помилок, оброблює результати та виводить текст з підкресленими помилковими словами.
extractKeywords()	Відправляє текст на сервер для виділення ключових слів та основної теми, виводить результати на сторінку.
inShort()	Відправляє текст на сервер для стиснення, замінює вміст введеного поля на скорочений текст.
spamHam()	Відправляє текст на сервер для класифікації як спам чи не спам, виводить результат класифікації на сторінку.
fixText()	Відправляє текст на сервер для корекції помилок, замінює вміст введеного поля на виправлений текст.

Для оцінки складності тексту використовується індекс Флеша.

Флеш-індекс — це числова оцінка, яка враховує кількість слів на речення та середню довжину слів у тексті. Наступне рівняння розраховує цей індекс:

FleschIndex=206.835−(1.015×(wordCount/sentenceCount))−(33.6×average WordLength)

Потім, на основі отриманого індексу, визначається рівень читабельності тексту. Якщо індекс менше 30, текст вважається дуже складним для читання. Якщо він між 30 і 49, то текст вважається складним. Якщо він між 50 і 59, то текст має середню складність. Якщо він між 60 і 69, текст вважається легким для читання. І якщо індекс 70 або більше, текст дуже легкий для читання.

Програмна реалізація має наступний вигляд:

```
 var fleschIndex = 206.835 - (1.015 * (wordCount / sentenceCount)) - (33.6 * (averageWordLength));
  console.log("Flesch Index:", fleschIndex);


 var readabilityLevel;
 if (fleschIndex < 30) {
     readabilityLevel = "Текст дуже складний для читання";
 } else if (fleschIndex < 50) {
     readabilityLevel = "Текст складний";
 } else if (fleschIndex < 60) {
     readabilityLevel = "Текст середньої складності";
 } else if (fleschIndex < 70) {
     readabilityLevel = "Текст легкий для читання";
 } else {
     readabilityLevel = "Текст дуже легкий для читання";
 }
```

3.3.3 Програмна реалізація візуальної частини

Візуальна частина реалізована методами HTML та CSS. розглянемо кілька ключових моментів:

- Метатеги <meta> встановлюють кодування символів та масштабування для адаптивності.

```
    <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Vecta - Інтелектуальний аналіз тексту</title>
```

- Зовнішні CSS та JavaScript файли підключаються через <link> та <script> відповідно. Параметризація CSS файлу через url_for забезпечує версіонування.

```
<link rel="stylesheet" type="text/css" href="{{ url_for( 'static',
filename='styles.css', v=1)}}">

<script src="{{ url_for('static', filename='script.js')
}}"></script>
```

- JavaScript код використовується для обробки вибраних файлів користувачем, використовуючи FileReader та бібліотеку Mammoth для обробки файлів .docx.
- <textarea> дозволяє користувачам вводити текст для аналізу. Кнопки нижче цього поля викликають різні функції JavaScript для аналізу та обробки цього тексту.
- Результати аналізу відображаються у відповідному з id "analysis-results", а також результат класифікації може бути відображений у відповідному <h3> з id "classification-result".

Було використано CSS для надання стилю елементам.

CSS розшифровується як Cascading Style Sheets (каскадні таблиці стилів). Це мова кодування, яка визначає зовнішній вигляд та макет веб-сайту. Поряд з HTML, CSS має фундаментальне значення для веб-дизайну. Без нього веб-сайти так і залишилися б звичайним текстом на білому фоні.

Приклад використання CSS:

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}
```

```
header {
    background-color: #333;
    color: #fff;
    padding: 20px 0;
}

header h1 {
    margin: 0;
}
```

Body. Встановлює шрифт тексту для всього тіла сторінки на Arial або будь-який замінник sans-serif, а також встановлює нульові значення для зовнішнього відступу та внутрішнього відступу, щоб усунути можливість прокрутки або порожній простір навколо сторінки.

Header. Встановлює кольори тла та тексту для заголовка сторінки. Фон встановлюється на темно-сірий (#333), а колір тексту - на білий. Також задається внутрішній відступ від верхнього та нижнього краю висотою 20 пікселів та нульовий відступ зліва та справа.

header h1. Встановлює нульовий зовнішній відступ для заголовка <h1> всередині елементу <header>, щоб усунути пробіли від країв заголовка до країв елемента <header>.

3.4 Методика роботи користувача

Веб-сайт має наступну структуру:

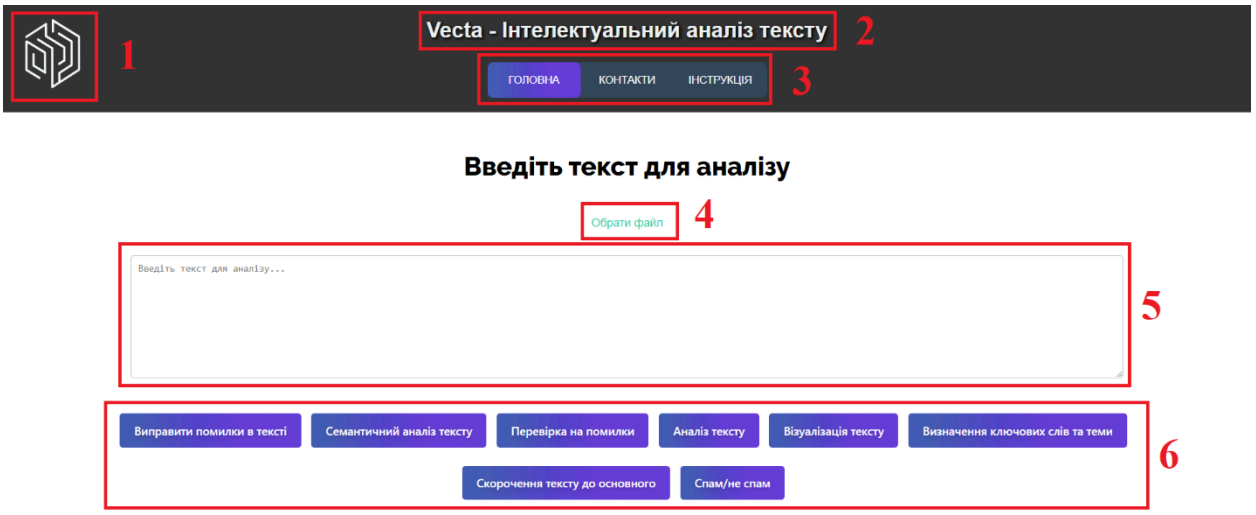


Рисунок 3.11 – Структура веб-сайту

Де:

1 – Логотип.

2 – Назва.

3 – Горизонтальне меню.

4 – Завантаження файлу.

5 – Поле для вводу текста.

6 – Кнопки функціоналу.

Користувачі можуть вводити текст або завантажувати файли з текстом, щоб виконати такі завдання, як:

- Перевірка тексту на граматичні помилки
- Семантичний аналіз тексту
- Аналіз тексту
- Візуалізація тексту
- Визначення ключових слів та теми
- Скорочення тексту до основного
- Визначення спаму/не спаму

Щоб використовувати веб-сайт Vecta, потрібно виконати наступні дії:

1. Запустити сайт на веб-сервері.

2. У полі «Введіть текст для аналізу» потрібно ввести текст якій необхідно проаналізувати.
3. Можна також завантажити файл з текстом, натиснувши кнопку Обрати файл.
4. Вибрати інструмент, який потрібно використати, з меню в центрі екрану
5. Натиснути кнопку.
6. Результати аналізу буде відображено на екрані.

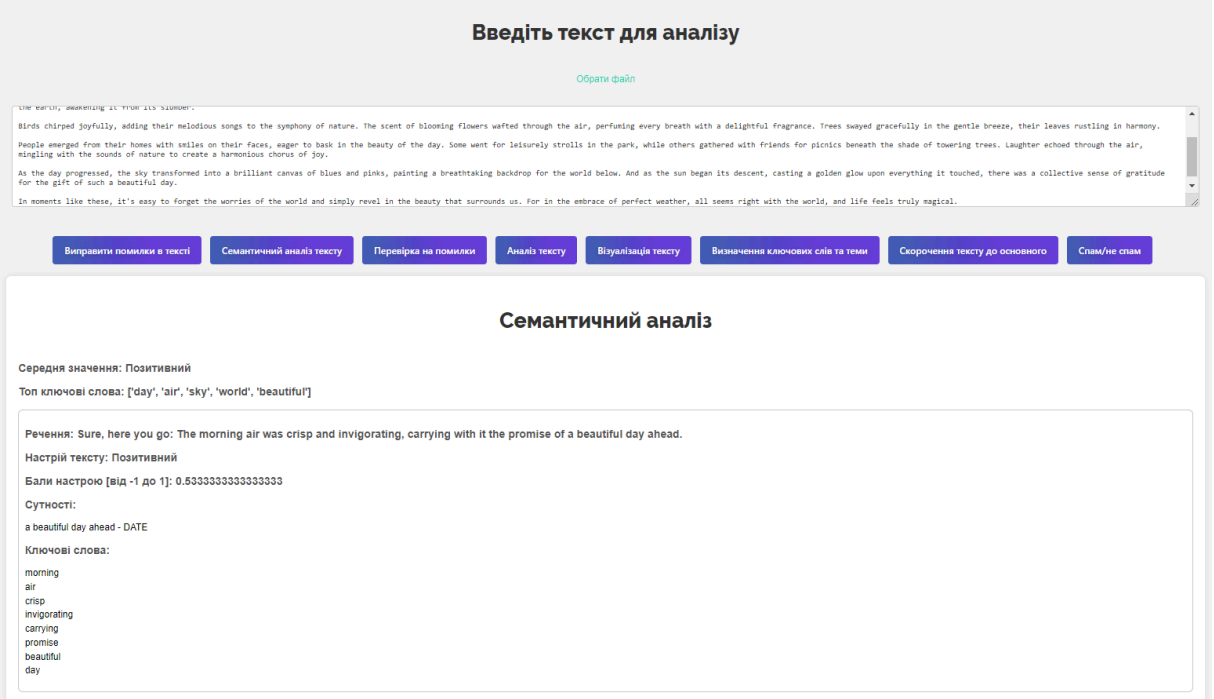


Рисунок 3.12 – Приклад виводу семантичного аналізу на сайті

Для локального запуску сайту на комп’ютері потрібно встановити необхідні бібліотеки з файлу requirements.txt за допомогою команди pip install -r requirements.txt. Після чого якщо всі файли встановлені корректно можна запустити сервер Flask.

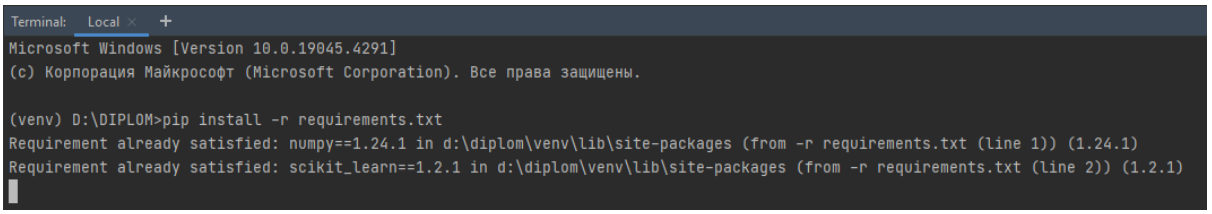


Рисунок 3.13 – Запуск команди pip install -r requirements.txt

Після запуску сайту потрібно перейти за IP адресою яка виведена в консолі. (по стандарту 127.0.0.1:5000).

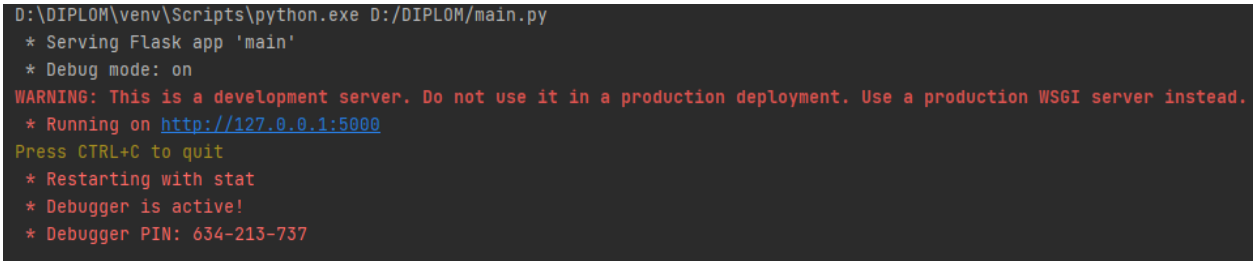


Рисунок 3.14 – Виведення в консолі при запуску сайту

ВИСНОВКИ

У даній кваліфікаційній роботі було проведено дослідження та розроблено систему інтелектуального аналізу тексту, використовуючи сучасні нейромережеві методи. Метою дослідження було створення системи з можливістю проведення семантичного аналізу, виявлення лексичних та орфографічних помилок, визначення теми тексту та цільової аудиторії, а також побудова хмар слів, графіків та короткого змісту.

У процесі дослідження було проаналізовано існуючі методи та алгоритми машинного навчання для обробки природної мови. На основі цього аналізу була розроблена архітектура системи, що включає модулі попередньої обробки тексту, класифікації тексту та видобутку інформації.

Розроблена система була протестована на тестових наборах даних. Результати показали, що розроблена система досягає високої точності у проведенні семантичного аналізу, у визначенні теми тексту та скорочення тексту до основного.

Практичне значення розробленої системи полягає у її можливості вирішення різних завдань, пов'язаних з аналізом тексту, таких як автоматична категоризація документів, аналіз відгуків, виявлення спаму, видобуток інформації з текстів, аналіз на помилки та скорочення текстів.

У додаток до розробки системи інтелектуального аналізу тексту, був реалізований веб-сайт з використанням мови програмування Python та фреймворку Flask.

Цей веб-сайт надає користувачам можливість взаємодії з системою аналізу тексту через зручний інтерфейс. На веб-сайті користувачі можуть відправляти тексти для аналізу, переглядати результати аналізу і отримувати зручний звіт.

Веб-інтерфейс також може включати додаткові функції, такі як можливість завантаження текстових файлів для аналізу, можливість налаштування параметрів аналізу та зручні інструменти візуалізації

результатів, наприклад, побудова графіків чи хмар слів на основі аналізованого тексту.

Реалізація веб-сайту на базі Python та Flask дозволяє забезпечити зручний доступ користувачів до функціоналу системи аналізу тексту через веб-браузер, що робить процес взаємодії з системою більш простим та доступним.

У підсумку, дана кваліфікаційна робота розкрила актуальність інтелектуального аналізу тексту в сучасному інформаційному середовищі. Шляхом дослідження і розробки системи на основі сучасних нейромережевих методів було досягнуто поставлених цілей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mohd, M., Javeed, S., , N., Wani, M., & Khanday, H. (2022). Sentiment analysis using lexico-semantic features. *Journal of Information Science*. https://doi.org/10.1177/01655515221124016.

2. Rogachev, A., Melikhova, E., & Atamanov, G. (2021). Building Artificial Neural Networks for NLP Analysis and Classification of Target Content. , 383-387. https://doi.org/10.2991/ASSEHR.K.210225.058.

3. Shorten, C., Khoshgoftaar, T., & Furht, B. (2021). Text Data Augmentation for Deep Learning. *Journal of Big Data*, 8. https://doi.org/10.1186/s40537-021-00492-0.

4. Maulud, D., Zeebaree, S., Jacksi, K., Sadeeq, M., & Sharif, K. (2021). State of Art for Semantic Analysis of Natural Language Processing. *Qubahan Academic Journal*. https://doi.org/10.48161/QAJ.V1N2A44.

5. Weiwei Zhang, Guangyu Zhai, Binbin Zhong, Xiaoyi Kong (2024). Text Semantic Analysis Algorithm Based on LDA Model and Doc2vec. Book Intelligent Computing Technology and Automation. https://www.researchgate.net/publication/377942645_Text_Semantic_Analysis_Algorithm_Based_on_LDA_Model_and_Doc2vec.

6. Salveter, S. (2021). Natural Language Processing. *A First Course in Artificial Intelligence*. https://doi.org/10.2174/9781681088532121010004.

7. Coburn, J. (2020). Getting Started with Flask. *Build Your Own Car Dashboard with a Raspberry Pi*, 143 - 170. https://doi.org/10.1007/978-1-4842-6080-7_6.

8. Engel, A., Wang, Z., Sarwate, A., Choudhury, S., & Chiang, T. (2022). TorchNTK: A Library for Calculation of Neural Tangent Kernels of PyTorch Models. *ArXiv*, abs/2205.12372. https://doi.org/10.48550/arXiv.2205.12372.

9. Shiya, V., & Sharmila, K. (2021). Language Processing and Python. *Advances in Computational Intelligence and Robotics*. https://doi.org/10.4018/978-1-7998-7728-8.ch006.

10. Zhao, S., & Li, A. (2023). An Natural Language Processed Web Application that Interpret and Convert English to Python Code. *Computer Science, Engineering and Applications*. https://doi.org/10.5121/csit.2023.130506.

11. Shah, M., Shenoy, R., & Shankarmani, R. (2021). Natural Language to Python Source Code using Transformers. *2021 International Conference on Intelligent Technologies (CONIT)*, 1-4. https://doi.org/10.1109/CONIT51480.2021.9498268.

12. Sarkar, D. (2019). Python for Natural Language Processing. *Text Analytics with Python*. https://doi.org/10.1007/978-1-4842-4354-1_2.

ДОДАТОК А – Код програми

main.py

```
from flask import Flask, render_template, request
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from io import BytesIO
import base64
from matplotlib_venn import venn2
import networkx as nx
import spacy
from textblob import TextBlob
from collections import Counter
from nltk.corpus import wordnet
from sklearn.cluster import KMeans
from nltk.tokenize import word_tokenize
import json
from nltk.corpus import stopwords
from flask import jsonify
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from transformers import pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from autocorrect import Speller

data = pd.read_csv('spam.csv')

X_train, X_test, y_train, y_test = train_test_split(data['text'],
data['label'], test_size=0.2, random_state=42)

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(X_train, y_train)
stop_words = set(stopwords.words('english'))

nlp = spacy.load("en_core_web_sm")

app = Flask(__name__)
```

```
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/contacts')
def contacts():
    return render_template('contacts.html')


@app.route('/spamHam', methods=['POST'])
def spamHam():
    text = request.form['text']
    prediction = model.predict([text])[0]
    return jsonify({'prediction': prediction})


@app.route('/inShort', methods=['POST'])
def in_short():
    data = request.get_json()
    text = data['text']

    def in_short_logic(text):
        sentences = text.split('.')
        main_sentence = ""
        max_polarity = -1
        for sentence in sentences:
            if len(sentence.strip()) > 0:
                polarity = TextBlob(sentence).sentiment.polarity
                if polarity > max_polarity:
                    main_sentence = sentence
                    max_polarity = polarity

        return main_sentence

    short_text = in_short_logic(text)

    return jsonify({'shortText': short_text})


spell = Speller(lang='en')
```

```
@app.route('/fixText', methods=['POST'])
def fix_text():
    data = request.get_json()
    text = data['text']
    corrected_text = spell(text)

    return jsonify({'correctedText': corrected_text})


@app.route('/checkErrors', methods=['POST'])
def check_errors():
    data = request.get_json()
    text = data['text']

    tokens = word_tokenize(text)

    tokens = [word for word in tokens if word.lower() not in stop_words]

    errors = []
    for token in tokens:
        if not wordnet.synsets(token):
            errors.append(token)

    return jsonify({'errors': errors})


@app.route('/instruction')
def instruction():
    return render_template('instruction.html')


@app.route('/semanticAnalysis', methods=['POST'])
def semantic_analysis():
    text = request.form['text']
    doc = nlp(text)
    semantic_results = []
    overall_sentiment = 0
    all_keywords = []

    for sent in doc.sents:
        sentiment_score = TextBlob(sent.text).sentiment.polarity
```

```
            sentiment_label = "Позитивний" if sentiment_score > 0 else
"Негативний" if sentiment_score < 0 else "Нейтральний"
        overall_sentiment += sentiment_score
        entities = [(ent.text, ent.label_) for ent in sent.ents]
           keywords = [token.text for token in sent if not token.is_stop and
token.pos_ in ['NOUN', 'VERB', 'ADJ']]
        semantic_results.append({
            "sentence": sent.text,
            "sentiment": sentiment_label,
            "sentiment_score": sentiment_score,
            "entities": entities,
            "keywords": keywords
        })
        all_keywords.extend(keywords)

                top_keywords = [keyword for keyword, _ in
Counter(all_keywords).most_common(5)]

      overall_sentiment_label = "Позитивний" if overall_sentiment > 0 else
"Негативний" if overall_sentiment < 0 else "Нейтральний"

    return render_template('semantic_analysis.html', results=semantic_results,
                                    overall_sentiment=overall_sentiment_label,
top_keywords=top_keywords)


@app.route('/extractKeywords', methods=['POST'])
def extract_keywords():
    text = request.form['text']
    doc = nlp(text)
       keywords = [token.text for token in doc if not token.is_stop and
token.pos_ in ['NOUN', 'VERB', 'ADJ']]
    max_keyword_count = 0
    main_theme = None
    for sent in doc.sents:
        keyword_count = sum(1 for token in sent if token.text in keywords)
        if keyword_count > max_keyword_count:
            max_keyword_count = keyword_count
            main_theme = sent.text

    return jsonify({'keywords': keywords[:3], 'main_theme': main_theme})
```

```
@app.route('/visualizeText', methods=['POST'])
def visualize_text():
    text = request.form['text']

    word_counts = {}
    for word in text.split():
        word_counts[word] = word_counts.get(word, 0) + 1

    plt.figure(figsize=(10, 5))
    plt.bar(word_counts.keys(), word_counts.values())
    plt.xlabel('Слова')
    plt.ylabel('Частота')
    plt.title('Частота зустрічі слів', fontsize=16, color='blue')
    plt.xticks(rotation=45)
    plt.tick_params(axis='x', which='both', bottom=False, labelbottom=False)

    img1 = BytesIO()
    plt.savefig(img1, format='png')
    img1.seek(0)

                    wordcloud     =     WordCloud(width=800,     height=400,
background_color='white').generate(text)

    img2 = BytesIO()
    wordcloud.to_image().save(img2, format='PNG')
    img2.seek(0)

    plt.figure(figsize=(10, 5))
    x = range(len(text.split()))
    y = [len(word) for word in text.split()]
    plt.scatter(x, y)
    plt.xlabel('Слова')
    plt.ylabel('Довжина слів')
    plt.title('Графік розподілу', fontsize=16, color='blue')

    img3 = BytesIO()
    plt.savefig(img3, format='png')
    img3.seek(0)

    G = nx.Graph()
    words = text.split()
    for i in range(len(words) - 1):
        word1 = words[i]
```

```
        word2 = words[i + 1]
        if G.has_edge(word1, word2):
            G[word1][word2]['weight'] += 1
        else:
            G.add_edge(word1, word2, weight=1)

    plt.figure(figsize=(10, 5))

    pos = nx.spring_layout(G, k=0.15, iterations=20)

                nx.draw(G,    pos,    with_labels=True,    font_weight='bold',
node_color='skyblue', node_size=1500, edge_color='black',
            linewidths=1, font_size=6)

    img4 = BytesIO()
    plt.savefig(img4, format='png')
    img4.seek(0)

    plot_url1 = base64.b64encode(img1.getvalue()).decode()
    plot_url2 = base64.b64encode(img2.getvalue()).decode()
    plot_url3 = base64.b64encode(img3.getvalue()).decode()
    plot_url4 = base64.b64encode(img4.getvalue()).decode()

                    return     '<div     style="margin-bottom:     20px;"><img
src="{}"  style="display:block;  margin:auto;"></div><div
style="margin-bottom:        20px;"><img        src="{}"
style="display:block;      margin:auto;"></div><div      style="margin-bottom:
20px;"><img        src="{}"        style="display:block;
margin:auto;"></div><div><img                    src="{}"
style="display:block; margin:auto;"></div>'.format(
        plot_url1, plot_url2, plot_url3, plot_url4)


if __name__ == '__main__':
    app.run(debug=True)
```

Script.js

```
function textAnalysis() {
    var text = document.getElementById("text-input").value;
    var characterCount = text.length;

    var wordCount = text.split(/\s+/).filter(function(word) {
        return word.length > 0;
    }).length;

    var uniqueWords = [...new Set(text.split(/\s+/))].filter(function(word) {
        return word.length > 0;
    }).length;

    var sentenceCount = text.split(/[.!?]+/).filter(function(sentence) {
        return sentence.trim().length > 0;
    }).length;

    var averageWordLength = text.split(/\s+/).reduce(function(total, word) {
        return total + word.length;
    }, 0) / wordCount;

    var wordFrequency = {};
    text.split(/\s+/).forEach(function(word) {
        if (word.length > 0) {
            wordFrequency[word] = (wordFrequency[word] || 0) + 1;
        }
    });

    var topWords = Object.keys(wordFrequency).sort(function(a, b) {
        return wordFrequency[b] - wordFrequency[a];
    }).slice(0, 5);

    var punctuationFrequency = {};
    var punctuation = /[!"#$%&'()*+,-./:;<=>?@[\\\]^_`{|}~]/g;
    text.split('').forEach(function(char) {
        if (punctuation.test(char)) {
              punctuationFrequency[char] = (punctuationFrequency[char] || 0) +
1;
        }
    });
```

```
    var fleschIndex = 206.835 - (1.015 * (wordCount / sentenceCount)) - (33.6
* (averageWordLength));
    console.log("Flesch Index:", fleschIndex);


    var readabilityLevel;
    if (fleschIndex < 30) {
        readabilityLevel = "Текст дуже складний для читання";
    } else if (fleschIndex < 50) {
        readabilityLevel = "Текст складний";
    } else if (fleschIndex < 60) {
        readabilityLevel = "Текст середньої складності";
    } else if (fleschIndex < 70) {
        readabilityLevel = "Текст легкий для читання";
    } else {
        readabilityLevel = "Текст дуже легкий для читання";
    }

var analysisResults = {
    "Кількість літер": characterCount,
    "Кількість слів": wordCount,
    "Кількість унікальних слів": uniqueWords,
    "Кількість речень": sentenceCount,
    "Середня довжина слова": averageWordLength.toFixed(2),
    "Найчастіше використані слова": topWords.map(function(word) {
        return word + " (" + wordFrequency[word] + ")";
    }).join(", "),
                                    "Кількість        знаків        пунктуації":
Object.keys(punctuationFrequency).map(function(punctuation) {
        return punctuation + " (" + punctuationFrequency[punctuation] + ")";
    }).join(", "),
    "Рівень складності читання": readabilityLevel
};

var analysisHTML = "<h2>Результати аналізу тексту:</h2><table>";
for (var key in analysisResults) {
       analysisHTML  +=  "<tr><td><strong>"  +  key  +  ":</strong></td><td>"  +
analysisResults[key] + "</td></tr>";
}
analysisHTML += "</table>";

document.getElementById("analysis-results").innerHTML = analysisHTML;
```

```
}

        function visualizeText() {
            var text = document.getElementById('text-input').value;
            var xhr = new XMLHttpRequest();
            xhr.open('POST', '/visualizeText', true);
                                            xhr.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');
            xhr.onreadystatechange = function() {
                if (xhr.readyState == 4 && xhr.status == 200) {
                        document.getElementById('analysis-results').innerHTML =
xhr.responseText;
                }
            };
            xhr.send('text=' + encodeURIComponent(text));
        }


function semanticAnalysis() {
    var text = document.getElementById("text-input").value;
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "/semanticAnalysis", true);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
                        document.getElementById("analysis-results").innerHTML  =
xhr.responseText;
        }
    };
    xhr.send("text=" + encodeURIComponent(text));
}

function checkErrors() {
    var text = document.getElementById('text-input').value.trim();

    fetch('/checkErrors', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ text: text })
    })
    .then(response => response.json())
```

```
    .then(data => {
        var resultDiv = document.getElementById('analysis-results');
        resultDiv.innerHTML = '';

        var textWithErrors = '';
        var tokens = text.split(' ');

        var errorHeader = document.createElement('h3');
        errorHeader.textContent = 'Помилки:';
        resultDiv.appendChild(errorHeader);

        tokens.forEach(token => {
            if (data.errors.includes(token)) {
                    textWithErrors += '<span class="error-word">' + token +
'</span> ';
            } else {
                textWithErrors += token + ' ';
            }
        });

        resultDiv.style.paddingLeft = '20px';

        resultDiv.innerHTML += textWithErrors;
    })
    .catch(error => {
        console.error('Error:', error);
    });
}

function extractKeywords() {
    var text = document.getElementById('text-input').value;
    fetch('/extractKeywords', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
        },
        body: 'text=' + encodeURIComponent(text),
    })
    .then(response => response.json())
    .then(data => {
        var resultsDiv = document.getElementById('analysis-results');
```

```
                    resultsDiv.innerHTML = '<h3>Ключові слова:</h3>' +
data.keywords.join(', ') + '<h3>Основна тема:</h3>' + (data.main_theme ?
data.main_theme : 'Не знайдено');
    })
    .catch(error => console.error('Error:', error));
}

function inShort() {
    var text = document.getElementById('text-input').value;

    fetch('/inShort', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ text: text }),
    })
    .then(response => response.json())
    .then(data => {
        var shortText = data.shortText;
        document.getElementById('text-input').value = shortText;
    })
    .catch(error => {
        console.error('Error:', error);
    });
}

function spamHam() {
    var text = document.getElementById('text-input').value;

    fetch('/spamHam', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
        },
        body: 'text=' + encodeURIComponent(text),
    })
    .then(response => response.json())
    .then(data => {
                                        var classificationResult =
document.getElementById('classification-result');
        if (data.prediction === 'spam') {
            classificationResult.textContent = 'Це спам!';
```

```
        } else {
            classificationResult.textContent = 'Це не спам.';
        }
    })
    .catch(error => {
        console.error('Помилка:', error);
    });
}

function fixText() {
    const text = document.getElementById('text-input').value;

    fetch('/fixText', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ text: text })
    })
    .then(response => response.json())
    .then(data => {
        const correctedText = data.correctedText;
        document.getElementById('text-input').value = correctedText;
    })
    .catch(error => console.error('Error:', error));
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vecta - Інтелектуальний аналіз тексту</title>
<link rel="stylesheet" type="text/css" href="{{ url_for( 'static',
filename='styles.css', v=1)}}">

<script src="{{ url_for('static', filename='script.js') }}"></script>

</head>
<body>
    <header>
       <img src="{{ url_for('static', filename='logo.png') }}" alt="Логотип
Vecta" class="logo">
        <h1>Vecta - Інтелектуальний аналіз тексту</h1>
        <nav>
    <a href="/">Головна</a>
    <a href="contacts">Контакти</a>
    <a href="instruction">Інструкція</a>
                <div class="animation start-home"></div>
        </nav>
    </header>
    <main>
        <h2 class="maintextinput">Введіть текст для аналізу</h2>
                 <label class="file-upload">
            <span>Обрати файл</span>
            <input type="file" id="file-input" accept=".txt, .docx, .json">
        </label>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/mammoth/1.5.0/mammoth.browser.min.
js"></script>
<script>
              document.getElementById('file-input').addEventListener('change',
function(event) {
        const file = event.target.files[0];
        const reader = new FileReader();
        reader.onload = function(e) {
            const contents = e.target.result;
            if (file.name.endsWith('.txt') || file.name.endsWith('.json')) {
```

```
                document.getElementById('text-input').value = contents;
            } else if (file.name.endsWith('.docx')) {
                mammoth.extractRawText({arrayBuffer: contents})
                    .then(function(result) {
                                document.getElementById('text-input').value =
result.value;
                    })
                    .catch(function(err) {
                        console.log(err);
                        alert("Ошибка при обработке DOCX файла.");
                    });
            } else {
                alert("Неподдерживаемый формат файла.");
            }
        };
        if (file.name.endsWith('.txt') || file.name.endsWith('.json')) {
            reader.readAsText(file);
        } else {
            reader.readAsArrayBuffer(file);
        }
    });
</script>
        <div class="text-input">
                    <textarea id="text-input" placeholder="Введіть текст для
аналізу..."></textarea>
            <div class="buttons">
                            <button onclick="fixText()">Виправити помилки в
тексті</button>
                        <button onclick="semanticAnalysis()">Семантичний аналіз
тексту</button>
                <button onclick="checkErrors()">Перевірка на помилки</button>
                <button onclick="textAnalysis()">Аналіз тексту</button>
                <button onclick="visualizeText()">Візуалізація тексту</button>
                  <button onclick="extractKeywords()">Визначення ключових слів
та теми</button>
                            <button onclick="inShort()">Скорочення тексту до
основного</button>
                <button onclick="spamHam()">Спам/не спам</button>
            </div>
        </div>

        <div id="analysis-results"></div>
<h3 id="classification-result"></h3>
```

```
    </main>
    <script src="script.js"></script>
</body>
</html>
```

Style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

header {
    background-color: #333;
    color: #fff;
    padding: 20px 0;
}

header h1 {
    margin: 0;
}

.logo {
    width: 100px;
    position: absolute;
    left: 20px;
      }

nav ul {
    list-style-type: none;
    padding: 0;
}

nav ul li {
    display: inline;
    margin-right: 20px;
}

nav ul li a {
    color: #fff;
    text-decoration: none;
}

main {
    padding: 20px;
}
```

```
    .text-input {
        text-align: center;
    }
    .buttons button {
          background-image: linear-gradient(92.88deg, #455EB5 9.16%, #5643CC
43.89%, #673FD7 64.72%);
      border-radius: 8px;
      border-style: none;
      box-sizing: border-box;
      color: #FFFFFF;
      cursor: pointer;
      margin-top: 25px;
      flex-shrink: 0;
                                  font-family:     "Inter     UI","SF     Pro
Display",-apple-system,BlinkMacSystemFont,"Segoe
UI",Roboto,Oxygen,Ubuntu,Cantarell,"Open Sans","Helvetica Neue",sans-serif;
      font-size: 12px;
      font-weight: 500;
      height: 3rem;
      padding: 0 0.6rem;
      text-align: center;
      text-shadow: rgba(0, 0, 0, 0.25) 0 3px 8px;
      transition: all .5s;
      user-select: none;
      -webkit-user-select: none;
      touch-action: manipulation;
    }


    .buttons button:hover {
      box-shadow: rgba(80, 63, 205, 0.5) 0 1px 30px;
      transition-duration: .3s;
    }

    @media (min-width: 768px) {
      .buttons button {
        padding: 0 2.6rem;
      }
    }

    nav {
        margin: 27px auto 0;
        position: relative;
```

```
    width: 390px;
    height: 50px;
    background-color: #34495e;
    border-radius: 8px;
    font-size: 0;
}

nav a {
    line-height: 50px;
    height: 100%;
    font-size: 15px;
    display: inline-block;
    position: relative;
    z-index: 1;
    text-decoration: none;
    text-transform: uppercase;
    text-align: center;
    color: white;
    cursor: pointer;
}



nav .animation {
    position: absolute;
    height: 100%;
    top: 0;
    z-index: 0;
    transition: all .5s ease 0s;
    border-radius: 8px;
}

a:nth-child(1) {
    width: 130px;
}

a:nth-child(2) {
    width: 130px;
}

a:nth-child(3) {
    width: 130px;
}
```

```
    nav .start-home, a:nth-child(1):hover~.animation {
        width: 130px;
        left: 0;
          background-image:  linear-gradient(92.88deg,  #455EB5  9.16%,  #5643CC
43.89%, #673FD7 64.72%);
    }

    nav .start-contacts, a:nth-child(2):hover~.animation {
        width: 130px;
        left: 130px;
          background-image:  linear-gradient(92.88deg,  #455EB5  9.16%,  #5643CC
43.89%, #673FD7 64.72%);
    }

    nav .start-instruction, a:nth-child(3):hover~.animation {
        width: 130px;
        left: 260px;
          background-image:  linear-gradient(92.88deg,  #455EB5  9.16%,  #5643CC
43.89%, #673FD7 64.72%);
    }

        .function {
            margin-bottom: 20px;
        }
        .function h1 {
            color: #333;
        }
        .function p {
            color: #666;
        }


    h1 {
        margin: 40px 0 40px;
        text-align: center;
        font-size: 30px;
        color: #ecf0f1;
        text-shadow: 2px 2px 4px #000000;
    }

    p {
        position: absolute;
```

```
    bottom: 20px;
    width: 100%;
    text-align: center;
    color: #ecf0f1;
    font-size: 16px;
}

span {
    color: #2BD6B4;
}

.text-input textarea {
    width: 80%;
    height: 150px;
    margin-bottom: 20px;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

.text-input .buttons button {
    margin-right: 10px;
    padding: 10px 20px;
    font-size: 16px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 50px 0;
    position: fixed;
    bottom: 0;
    width: 100%;
}

table {
    width: 70%;
        margin: auto;
    border-collapse: collapse;
```

```
        margin-top: 20px;
    }

div.table-title {
   display: block;
  margin: auto;
  max-width: 600px;
  padding:5px;
  width: 100%;
}

.table-title h3 {
   color: #fafafa;
   font-size: 30px;
   font-weight: 400;
   font-style:normal;
   font-family: "Roboto", helvetica, arial, sans-serif;
   text-shadow: -1px -1px 1px rgba(0, 0, 0, 0.1);
   text-transform:uppercase;
}



.table-fill {
  background: white;
  border-radius:3px;
  border-collapse: collapse;
  height: 320px;
  margin: auto;
  max-width: 600px;
  padding:5px;
  width: 100%;
  box-shadow: 0 5px 10px rgba(0, 0, 0, 0.1);
  animation: float 5s infinite;
}

th {
  color:#D5DDE5;;
        background-color: #333;
text-align: center;
  border-bottom:4px solid #9ea7af;
  border-right: 1px solid #343a45;
  font-size:23px;
  font-weight: 100;
```

```
  padding:24px;
  text-align:left;
  text-shadow: 0 1px 1px rgba(0, 0, 0, 0.1);
  vertical-align:middle;
}

th:first-child {
    text-align: center;
  border-top-left-radius:3px;
}

th:last-child {
    text-align: center;
  border-top-right-radius:3px;
  border-right:none;
}

tr {
  border-top: 1px solid #C1C3D1;
  border-bottom-: 1px solid #C1C3D1;
  color:#666B85;
  font-size:16px;
  font-weight:normal;
  text-shadow: 0 1px 1px rgba(256, 256, 256, 0.1);
}

tr:hover td {
  background:#4E5066;
  color:#FFFFFF;
  border-top: 1px solid #22262e;
}

tr:first-child {
  border-top:none;
}

tr:last-child {
  border-bottom:none;
}

tr:nth-child(odd) td {
  background:#EBEBEB;
}
```

```
tr:nth-child(odd):hover td {
  background:#4E5066;
}

tr:last-child td:first-child {
  border-bottom-left-radius:3px;
}

tr:last-child td:last-child {
  border-bottom-right-radius:3px;
}

td {
  background:#FFFFFF;
  padding:20px;
  text-align:left;
  vertical-align:middle;
  font-weight:300;
  font-size:18px;
  text-shadow: -1px -1px 1px rgba(0, 0, 0, 0.1);
  border-right: 1px solid #C1C3D1;
}

td:last-child {
  border-right: 0px;
}

th.text-left {
  text-align: left;
}

th.text-center {
  text-align: center;
}

th.text-right {
  text-align: right;
}

td.text-left {
  text-align: left;
}
```

```
td.text-center {
  text-align: center;
}

td.text-right {
  text-align: right;
}

.funcdesc{
    text-align: center;
}

h2 {
    color: black;
    font-family: 'Raleway', sans-serif;
    font-size: 36px;
    font-weight: 800;
    line-height: 72px;
    margin: 10px 500px 24px;
    text-align: center;
    padding: 10px;
}

.file-upload {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    padding-bottom: 25px;

}

.file-upload span {
    margin-bottom: 10px;
}
.file-upload input[type="file"] {
    display: none;
}

.error-word {
    color: #ff0000;
}
```