НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ** (підпис)	_____ **Белла ГОЛУБ** (підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: « Програмне забезпечення для автоматичного створення текстових документів »

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

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

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

Виконав _____ Денис **ПОСТУМЕНТ**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

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

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Постументу Денису Андрійовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для автоматичного створення текстових документів»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Обробка текстових документів форматів TXT, DOCX та PDF, автоматичне створення короткого викладу тексту, використання локального режиму та технологій штучного інтелекту для аналізу документів.
Перелік питань, що підлягають дослідженню:
Аналіз предметної області, моделювання предметної області , проєктування програмної системи, реалізація системи автоматичного резюмування тексту, тестування та аналіз роботи системи
Перелік графічних документів (за необхідності).
_____

Дата видачі завдання «___» _____ 2025 р.

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

Завдання взяв до виконання _____ **Денис ПОСТУМЕНТ**

ЗМІСТ

ВСТУП

У сучасному світі обсяг текстової інформації постійно зростає. Користувачі щодня працюють із великою кількістю документів, серед яких навчальні матеріали, технічна документація, статті, звіти та інші текстові файли. Опрацювання великих обсягів тексту потребує значних витрат часу, тому виникає потреба у використанні програмних засобів для автоматичного аналізу документів та створення їх короткого викладу.

Сучасні технології штучного інтелекту та методи обробки природної мови дозволяють автоматизувати процес резюмування тексту та значно спростити роботу користувача з інформацією. Використання таких систем дає можливість швидко отримати основний зміст документа без необхідності повного прочитання тексту. Особливо актуальними подібні рішення є для студентів, викладачів, працівників IT-сфери та користувачів, які регулярно працюють із великими текстовими файлами.

Актуальність дослідження визначається необхідністю автоматизації процесу опрацювання текстових документів, зменшення часу на аналіз інформації та підвищення зручності роботи з електронними документами. Використання локальних алгоритмів резюмування та сучасних AI-технологій дозволяє створювати короткий виклад тексту, зберігаючи основний зміст документа та ключові ідеї.

Метою бакалаврської кваліфікаційної роботи є розробка програмного забезпечення для автоматичного створення короткого викладу текстових документів із використанням локальних алгоритмів аналізу тексту та технологій штучного інтелекту. Розроблена система повинна забезпечувати завантаження текстових документів, аналіз вмісту, створення короткого викладу тексту та збереження результатів у різних форматах.

Об'єктом дослідження є процес автоматизованого опрацювання текстових документів із використанням сучасних інформаційних технологій.

Предметом дослідження є методи, моделі та програмні засоби автоматичного створення короткого викладу текстових документів на основі мови програмування Python, локальних алгоритмів аналізу тексту та систем штучного інтелекту.

Для досягнення поставленої мети були визначені такі завдання:

- проаналізувати існуючі системи автоматичного резюмування тексту та їх основні особливості;
- визначити функціональні та нефункціональні вимоги до програмного забезпечення;
- спроєктувати архітектуру програмної системи та структуру бази даних;
- розробити програмний застосунок для роботи з текстовими документами;
- реалізувати підтримку форматів TXT, DOCX та PDF;
- реалізувати локальний режим створення короткого викладу тексту;
- інтегрувати AI-режим із використанням системи Ollama;
- реалізувати систему збереження історії обробки документів;
- провести тестування програмного забезпечення та перевірити його працездатність.

Тестування програмного застосунку проводилось шляхом функціонального тестування основних модулів системи та перевірки коректності обробки текстових документів різних форматів.

Практичне значення одержаних результатів полягає у створенні програмного забезпечення STYSLO AI для автоматичного створення короткого викладу текстових документів із використанням локальних алгоритмів аналізу тексту та технологій штучного інтелекту. Розроблена система дозволяє автоматизувати процес аналізу великих текстових

документів, скоротити час опрацювання інформації та спростити роботу користувачів із навчальними матеріалами, документацією та іншими електронними файлами.

Програмне забезпечення підтримує роботу з форматами TXT, DOCX та PDF, локальний режим обробки тексту, AI-аналіз через систему Ollama, а також збереження результатів у різних форматах. Результати роботи можуть бути використані у навчальній діяльності, роботі з документацією та для подальшого розвитку систем автоматичного резюмування тексту.

Структура роботи: бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі виконано аналіз предметної області та існуючих програмних рішень. У другому розділі описано моделювання предметної області та UML-діаграми системи. У третьому розділі представлено проєктування та реалізацію програмного забезпечення. Четвертий розділ містить результати тестування системи та аналіз її роботи.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному цифровому середовищі користувачі постійно працюють із великою кількістю текстової інформації. Значна частина даних зберігається у вигляді електронних документів - навчальних матеріалів, технічної документації, статей, звітів та інших текстових файлів. Через постійне збільшення обсягів інформації виникає проблема швидкого ознайомлення зі змістом документів та виділення основних ідей тексту.

Однією з головних проблем під час роботи з текстовими документами є необхідність витрачати значний час на читання великих обсягів інформації. Користувачі часто працюють із файлами, які містять десятки або навіть сотні сторінок тексту. У таких випадках виникає потреба швидко отримати короткий виклад документа без необхідності повного ознайомлення з усім текстом. Особливо актуальною ця проблема є для студентів, викладачів, працівників IT-сфери та користувачів, які регулярно працюють із документацією та навчальними матеріалами.

Традиційний спосіб створення короткого викладу тексту виконується вручну та потребує значних витрат часу. Користувач повинен самостійно аналізувати текст, виділяти основні думки та формувати узагальнений зміст документа. При роботі з великими файлами це ускладнює процес опрацювання інформації та знижує ефективність роботи з документами.

Сучасні технології дозволяють автоматизувати процес аналізу тексту за допомогою алгоритмів обробки природної мови та систем штучного інтелекту. Такі системи дають можливість автоматично визначати основні частини документа, аналізувати зміст тексту та формувати короткий виклад без прямого втручання користувача. Використання подібних програмних рішень

дозволяє суттєво скоротити час роботи з текстовими файлами та спростити процес ознайомлення з інформацією.

Незважаючи на існування різних онлайн-сервісів для резюмування тексту, багато з них мають певні недоліки. Частина систем працює лише через веб-інтерфейс, вимагає стабільного доступу до мережі Інтернет або має обмеження щодо обсягу тексту. Деякі сервіси не підтримують локальну обробку документів або не забезпечують достатньої якості створеного короткого викладу. Також окремі рішення не підтримують роботу з різними форматами файлів або мають обмежений функціонал.

Аналіз предметної області показує, що використання програмного забезпечення для автоматичного створення короткого викладу текстових документів є актуальним та перспективним напрямком. Впровадження подібних систем дозволяє автоматизувати процес аналізу тексту, зменшити витрати часу на опрацювання інформації та підвищити зручність роботи з електронними документами.

Предметом даної бакалаврської кваліфікаційної роботи є процес автоматичного створення короткого викладу текстових документів із використанням локальних алгоритмів аналізу тексту та технологій штучного інтелекту. Розроблена система призначена для роботи з текстовими файлами різних форматів та автоматичного формування стислого змісту документа.

Предметна область охоплює процеси завантаження, аналізу та обробки текстових документів. Система повинна підтримувати роботу з файлами форматів TXT, DOCX та PDF, виконувати аналіз тексту та формувати короткий виклад документа залежно від обраного режиму роботи.

Важливою частиною предметної області є використання локального режиму аналізу тексту та AI-режиму із застосуванням системи Ollama. Локальний режим дозволяє формувати короткий виклад документа за допомогою

алгоритмів частотного аналізу тексту без використання зовнішніх сервісів. AI-режим використовує можливості мовних моделей для більш глибокого аналізу змісту документа та створення більш якісного короткого викладу тексту.

Система також забезпечує можливість збереження результатів у різних форматах, ведення історії обробки документів та підтримку декількох мов інтерфейсу. Для реалізації програмного забезпечення використовуються сучасні технології та бібліотеки мови програмування Python, а також база даних SQLite для збереження інформації про результати роботи системи.

Детальний опис основних сутностей предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Документ	Назва Файлу	Назва текстового документа
	Формат файлу	Тип документа (TXT, DOCX, PDF)
	Вміст документа	Текстова інформація документа
	Розмір документа	Кількість символів або слів
Короткий виклад	Текст резюме	Автоматично сформований короткий виклад
	Мова	Мова створеного резюме
	Довжина резюме	Розмір короткого викладу
	Режим роботи	Локальний режим або AI-режим
Система резюмування	Назва системи	Назва програмного забезпечення
	Тип обробки	Спосіб аналізу тексту
	Історія обробки	Дані про раніше оброблені документи
	Формат збереження	Формат експорту результату

1.2 Огляд існуючих рішень

Зі збільшенням кількості електронних документів та обсягів текстової інформації значно зріс попит на програмні засоби для автоматичного аналізу тексту та створення короткого викладу документів. На сучасному ринку існує велика кількість онлайн-сервісів та AI-платформ, які дозволяють автоматизувати процес резюмування тексту, спростити роботу з документацією та пришвидшити пошук основної інформації у великих текстах.

Одним із найбільш популярних інструментів для роботи з текстом є ChatGPT від компанії OpenAI. Даний сервіс використовує технології штучного інтелекту для аналізу текстової інформації та генерації відповідей на запити користувачів. Система дозволяє створювати короткий виклад тексту, аналізувати документи, пояснювати інформацію та виконувати різні завдання, пов’язані з обробкою природної мови.

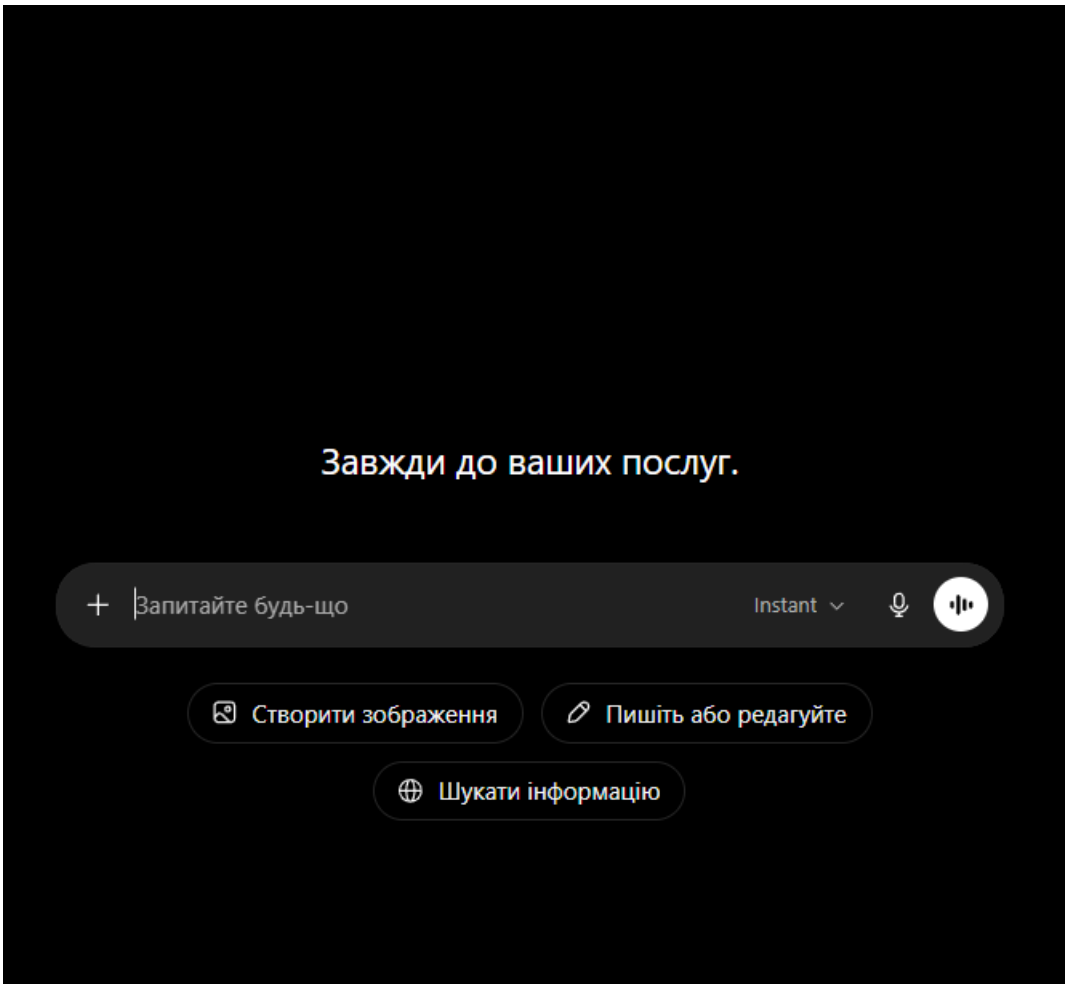


Рис. 1.1 - Інтерфейс системи ChatGPT

Основною перевагою ChatGPT є висока якість аналізу тексту та можливість працювати з великими обсягами інформації. Сервіс дозволяє формувати короткий виклад документа, виділяти ключові ідеї та створювати узагальнений зміст тексту. Крім цього, система підтримує декілька мов та забезпечує зручний інтерфейс взаємодії з користувачем.

Водночас використання ChatGPT має певні обмеження. Для роботи системи необхідне постійне підключення до мережі Інтернет, а також частина функцій доступна лише у платній версії сервісу. Крім того, під час роботи з конфіденційними документами можуть виникати питання щодо безпеки та передачі даних на зовнішні сервери.

Ще одним популярним рішенням у сфері автоматичного аналізу тексту є Grammarly. Даний сервіс призначений для перевірки граматики, аналізу тексту

та покращення якості написання документів. Окрім перевірки помилок, система також підтримує функції скорочення та узагальнення тексту.

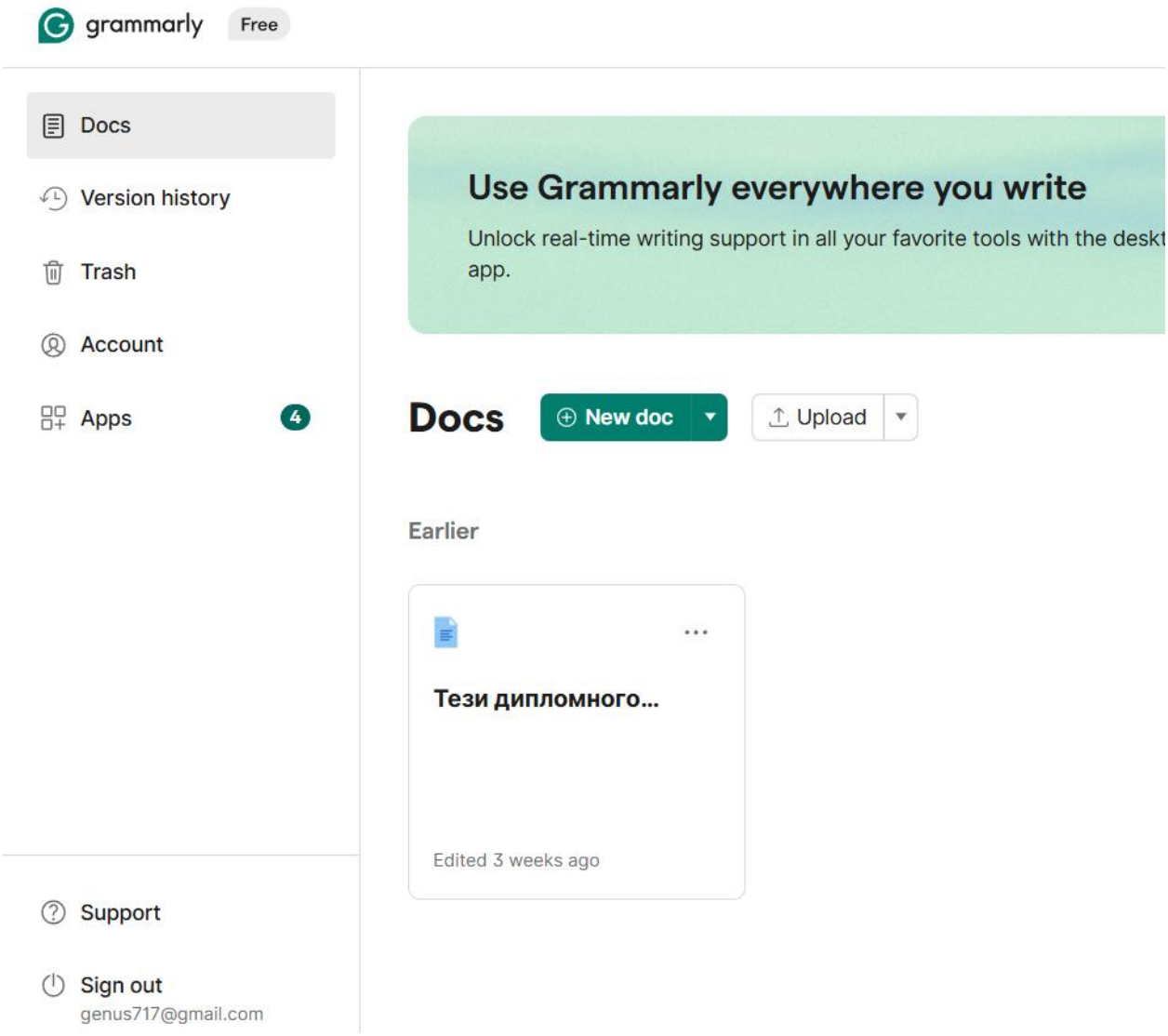


Рис. 1.2 - Інтерфейс системи Grammarly

Grammarly використовує алгоритми штучного інтелекту для аналізу структури тексту та покращення його читабельності. Система дозволяє автоматично визначати складні речення, пропонувати коротші формулювання та допомагає користувачу швидше працювати з текстовими документами.

Перевагою Grammarly є простота використання та інтеграція з браузерами, текстовими редакторами та онлайн-сервісами. Проте функціонал системи переважно орієнтований на перевірку тексту, а не на повноцінне автоматичне

створення короткого викладу великих документів. Також розширені можливості доступні лише у платній версії сервісу.

Серед локальних AI-рішень для роботи з текстом окрему увагу варто приділити системі Ollama. Дана платформа дозволяє запускати мовні моделі локально на комп'ютері користувача без необхідності використання зовнішніх серверів. Ollama підтримує інтеграцію різних AI-моделей та може використовуватись для аналізу тексту, генерації відповідей і створення короткого викладу документів.

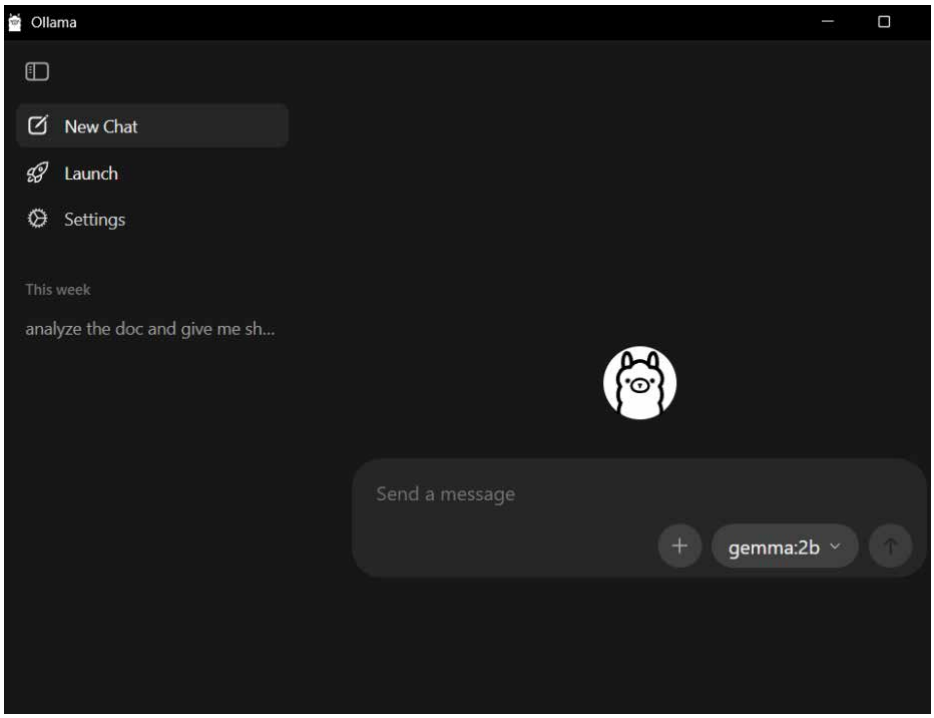


Рис. 1.3 - Робота системи Ollama

Основною перевагою Ollama є можливість локальної роботи з документами, що підвищує безпеку даних та дозволяє використовувати систему без постійного підключення до Інтернету. Крім цього, користувач може самостійно обирати AI-моделі для аналізу тексту та інтегрувати їх у власні програмні застосунки.

Недоліком Ollama є необхідність встановлення додаткових моделей та підвищені вимоги до апаратного забезпечення комп'ютера. Також для інтеграції системи у власне програмне забезпечення потрібні базові знання програмування та роботи з API.

Отже, аналіз існуючих рішень показує, що сучасні системи автоматичного резюмування тексту мають широкий функціонал та активно використовують технології штучного інтелекту. Проте частина сервісів має обмеження щодо локальної роботи, підтримки форматів файлів або доступності окремих функцій. Це підтверджує актуальність розробки власного програмного забезпечення для автоматичного створення короткого викладу текстових документів із підтримкою локального режиму роботи та AI-аналізу документів.

1.3 Постановка завдання

Організація систем автоматичного опрацювання текстових документів базується на необхідності швидкого аналізу великих обсягів інформації та формування короткого викладу тексту. У сучасних умовах користувачі часто працюють із документами значного розміру, що ускладнює процес пошуку основної інформації та потребує додаткових витрат часу на ознайомлення зі змістом тексту.

Основним завданням системи є автоматизація процесу аналізу текстових документів та створення їх короткого викладу. Розроблене програмне забезпечення повинно забезпечувати завантаження документів різних форматів, обробку тексту, визначення основних частин документа та формування стислого змісту з використанням локальних алгоритмів і технологій штучного інтелекту.

Ключові параметри, які повинна підтримувати система:

- обробка текстових документів форматів TXT, DOCX та PDF;
- аналіз змісту документа та визначення ключових фрагментів тексту;
- створення короткого викладу документа залежно від обраного режиму роботи;

- підтримка локального режиму аналізу тексту;
- підтримка AI-режиму з використанням системи Ollama;
- збереження результатів обробки у форматах TXT, DOCX та PDF;
- ведення історії обробки документів;
- підтримка декількох мов роботи системи.

Програмний застосунок використовує локальну базу даних SQLite для збереження інформації про оброблені документи, результати аналізу та історію роботи користувача. Збереження даних дозволяє повторно переглядати результати створених коротких викладів та забезпечує зручність роботи із системою.

Процес роботи системи організований у декілька етапів. Після завантаження документа програма виконує зчитування тексту та аналіз його структури. Далі залежно від обраного режиму система або використовує локальний алгоритм частотного аналізу тексту, або виконує AI-аналіз документа через інтеграцію з Ollama. Після завершення обробки користувач отримує готовий короткий виклад документа, який може бути збережений у необхідному форматі.

У випадку роботи з великими документами система автоматично розділяє текст на окремі частини для стабільної обробки. Це дозволяє уникнути перевантаження системи та забезпечує коректну роботу застосунку навіть із файлами значного розміру.

Інтерфейс користувача побудований таким чином, щоб забезпечити просту та зрозумілу взаємодію із системою. Основні елементи інтерфейсу дозволяють користувачу обирати режим роботи, мову, довжину короткого викладу, відкривати документи та зберігати результати обробки. Крім цього, система містить окремий модуль для перегляду історії обробки документів.

З архітектурної точки зору програмне забезпечення складається з окремих модулів для роботи з файлами, аналізу тексту, AI-обробки, бази даних та

графічного інтерфейсу користувача. Такий підхід дозволяє спростити подальший супровід системи та забезпечує можливість розширення функціоналу програмного застосунку.

Таким чином, розроблена система автоматичного створення короткого викладу текстових документів дозволяє спростити процес роботи з великими обсягами текстової інформації, скоротити час аналізу документів та підвищити ефективність опрацювання текстових даних.

1.4 Функціональні та нефункціональні вимоги

Фу Функціональні вимоги визначають основні можливості програмного забезпечення та описують дії, які повинна виконувати система автоматичного створення короткого викладу текстових документів.

Функціональні вимоги:

- завантаження текстових документів форматів TXT, DOCX та PDF;
- зчитування та аналіз текстового вмісту документа;
- автоматичне створення короткого викладу тексту;
- підтримка локального режиму резюмування документів;
- підтримка AI-режиму з використанням системи Ollama;
- можливість вибору довжини короткого викладу;
- підтримка декількох мов роботи системи;
- автоматичний поділ великих документів на частини для стабільної обробки;
- збереження результатів у форматах TXT, DOCX та PDF;
- ведення історії обробки документів;
- відображення статистики документа, зокрема кількості символів, слів та речень;

- відображення прогресу обробки документа у режимі реального часу;
- перевірка наявності необхідних AI-моделей для роботи AI-режиму;
- обробка помилок під час роботи із файлами або AI-модулями;
- очищення робочої області та повторна обробка документів.

Нефункціональні вимоги визначають характеристики системи, які впливають на її стабільність, швидкодію та зручність використання.

Нефункціональні вимоги:

- швидка обробка текстових документів малого та середнього обсягу;
- стабільна робота системи під час обробки великих файлів;
- простий та зрозумілий графічний інтерфейс користувача;
- підтримка роботи без підключення до Інтернету у локальному режимі;
- коректна робота із файлами різних форматів;
- збереження результатів обробки без втрати даних;
- можливість масштабування та розширення функціоналу системи;
- підтримка багатомовності інтерфейсу та результатів обробки;
- безпечне локальне збереження інформації у базі даних SQLite;
- надійна робота AI-режиму при інтеграції з Ollama;
- підтримка багатопотокової обробки для уникнення зависання інтерфейсу;
- відображення повідомлень про помилки та некоректну роботу системи;
- сумісність із сучасними операційними системами Windows.

Порівняння існуючих систем автоматичного створення короткого викладу тексту

Система	**Локальний режим**	**Підтримка ШІ**	**Складність розгортання**	Вимоги до ресурсів
ChatGPT	Ні	Так	Низька	Низькі

Ollama	Так	Так	Середня	Високі
STYSLO AI	Так	Так	Низька	Середні

Проведений аналіз існуючих рішень показав, що більшість сучасних систем автоматичного створення короткого викладу тексту орієнтовані на використання хмарних сервісів та потребують постійного підключення до мережі Інтернет. Такі системи забезпечують високий рівень підтримки технологій штучного інтелекту, проте мають обмеження щодо локальної роботи та контролю над обробкою даних.

На відміну від комерційних сервісів, програмне забезпечення STYSLO AI підтримує локальний режим роботи, інтеграцію з системою Ollama та дозволяє виконувати автоматичний аналіз текстових документів без передачі даних на зовнішні сервери. Це забезпечує більшу автономність роботи, підвищує рівень конфіденційності та спрощує використання системи в умовах обмеженого або відсутнього доступу до мережі Інтернет.

1.5 Вимоги до інтерфейсу користувача

Інтерфейс користувача програмного забезпечення повинен забезпечувати просту та зрозумілу взаємодію із системою автоматичного створення короткого викладу текстових документів. Основні елементи інтерфейсу мають бути розташовані логічно та забезпечувати швидкий доступ до функцій завантаження файлів, вибору режиму роботи, налаштування параметрів обробки тексту та збереження результатів.

Головне вікно програми повинно містити кнопки для відкриття документів, запуску процесу обробки тексту, перегляду історії роботи та збереження результатів у різних форматах. Користувач повинен мати можливість швидко перемикатися між локальним режимом та AI-режимом обробки документів, а також обирати мову та довжину короткого викладу тексту.

Інтерфейс повинен підтримувати роботу з файлами форматів TXT, DOCX та PDF. Після відкриття документа система повинна автоматично відображати процес обробки тексту та повідомляти користувача про стан виконання операції. Для цього у програмі передбачено використання прогрес-бара та текстових повідомлень про поточний етап аналізу документа.

Користувач повинен мати можливість переглядати результат обробки безпосередньо у вікні програми, а також зберігати створений короткий виклад у форматах TXT, DOCX та PDF. Інтерфейс також повинен забезпечувати доступ до історії раніше оброблених документів із можливістю перегляду детальної інформації про результати аналізу.

Важливою частиною інтерфейсу є система повідомлень про помилки та некоректну роботу окремих модулів. У випадку відсутності AI-моделей або помилок під час роботи системи користувач повинен отримувати зрозумілі повідомлення з описом проблеми та рекомендаціями щодо її вирішення.

Інтерфейс програмного застосунку повинен мати сучасний вигляд, підтримувати зручну навігацію та забезпечувати комфортну роботу користувача з великими текстовими документами. Особлива увага приділяється простоті використання, читабельності тексту та зручному розташуванню елементів керування системою.

Всі вимоги до інтерфейсу спрямовані на забезпечення швидкої та ефективної роботи користувача із системою автоматичного створення короткого викладу текстових документів.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

Unified Modeling Language (UML) - це універсальна мова моделювання, яка використовується для опису, проєктування та візуалізації програмних систем. UML дозволяє формалізувати структуру програмного забезпечення, відобразити взаємодію між окремими компонентами системи та забезпечити зрозуміле представлення архітектури програмного продукту.

Використання UML у процесі розробки програмного забезпечення дає можливість спростити аналіз предметної області, визначити основні функціональні можливості системи та виявити взаємозв’язки між її компонентами ще на етапі проєктування. UML-діаграми дозволяють візуально представити роботу системи, що значно полегшує процес розробки, тестування та подальшого супроводу програмного забезпечення.

Мова UML містить різні типи діаграм, які поділяються на структурні та поведінкові. Структурні діаграми використовуються для опису архітектури системи, її класів, компонентів, бази даних та фізичної структури програмного забезпечення. До них належать діаграми класів, компонентів, пакетів та розгортання. Поведінкові діаграми призначені для моделювання логіки роботи системи та взаємодії між її елементами. До таких діаграм належать діаграми прецедентів, діяльності та послідовності.

У процесі розробки програмного забезпечення STYSLO AI UML використовується для моделювання функціональних можливостей системи автоматичного створення короткого викладу текстових документів. Використання UML-діаграм дозволяє формалізувати процеси завантаження документів, аналізу тексту, створення короткого викладу та збереження результатів обробки.

Діаграма прецедентів використовується для визначення основних сценаріїв взаємодії користувача із системою. Вона дозволяє відобразити функціональні можливості програмного забезпечення, зокрема відкриття документів, запуск аналізу тексту, вибір режиму обробки та збереження результатів.

Діаграма діяльності демонструє послідовність виконання основних процесів системи та дозволяє відобразити алгоритм роботи програмного забезпечення під час обробки текстових документів. За допомогою даної діаграми можна показати етапи завантаження документа, аналізу тексту, формування короткого викладу та виведення результату користувачу.

Діаграма послідовності використовується для моделювання взаємодії між окремими компонентами системи у часі. Вона демонструє порядок обміну повідомленнями між користувачем, графічним інтерфейсом, модулем аналізу тексту, базою даних SQLite та AI-сервісом Ollama.

Діаграма класів дозволяє відобразити структуру програмної системи, основні класи, їх атрибути, методи та взаємозв’язки між ними. Використання даної діаграми забезпечує краще розуміння внутрішньої структури програмного забезпечення та спрощує процес реалізації системи.

Для опису структури програмних модулів використовується діаграма компонентів. Вона дозволяє відобразити взаємодію між окремими частинами системи, такими як графічний інтерфейс, модуль роботи з файлами, модуль AI-аналізу, база даних та модуль експорту результатів.

Діаграма розгортання використовується для моделювання фізичної архітектури програмного забезпечення. Вона демонструє розміщення основних компонентів системи на комп’ютері користувача та взаємодію програмного застосунку із локальним AI-сервісом Ollama.

Таким чином, використання UML у процесі розробки програмного забезпечення STYSLO AI дозволяє забезпечити структуроване представлення

системи, покращити розуміння її архітектури та спростити процес подальшої реалізації й супроводу програмного продукту.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма прецедентів (Use-Case Diagram) є одним із основних інструментів UML, який використовується для моделювання функціональних можливостей програмної системи та взаємодії користувача із програмним забезпеченням. Основною метою діаграми прецедентів є відображення сценаріїв використання системи та визначення дій, які може виконувати користувач під час роботи із програмним застосунком.

Для програмного забезпечення STYSLO AI діаграма прецедентів використовується для опису основних функцій системи автоматичного створення короткого викладу текстових документів. Діаграма дозволяє визначити взаємодію користувача із системою під час завантаження документа, запуску аналізу тексту, створення короткого викладу та збереження результатів обробки.

Основним актором системи є користувач, який взаємодіє з програмним забезпеченням через графічний інтерфейс. Користувач має можливість відкривати текстові документи, обирати режим роботи системи, запускати процес аналізу тексту, переглядати результати обробки та зберігати створений короткий виклад у різних форматах.

Розроблена діаграма прецедентів використання представлена на рис. 2.1.

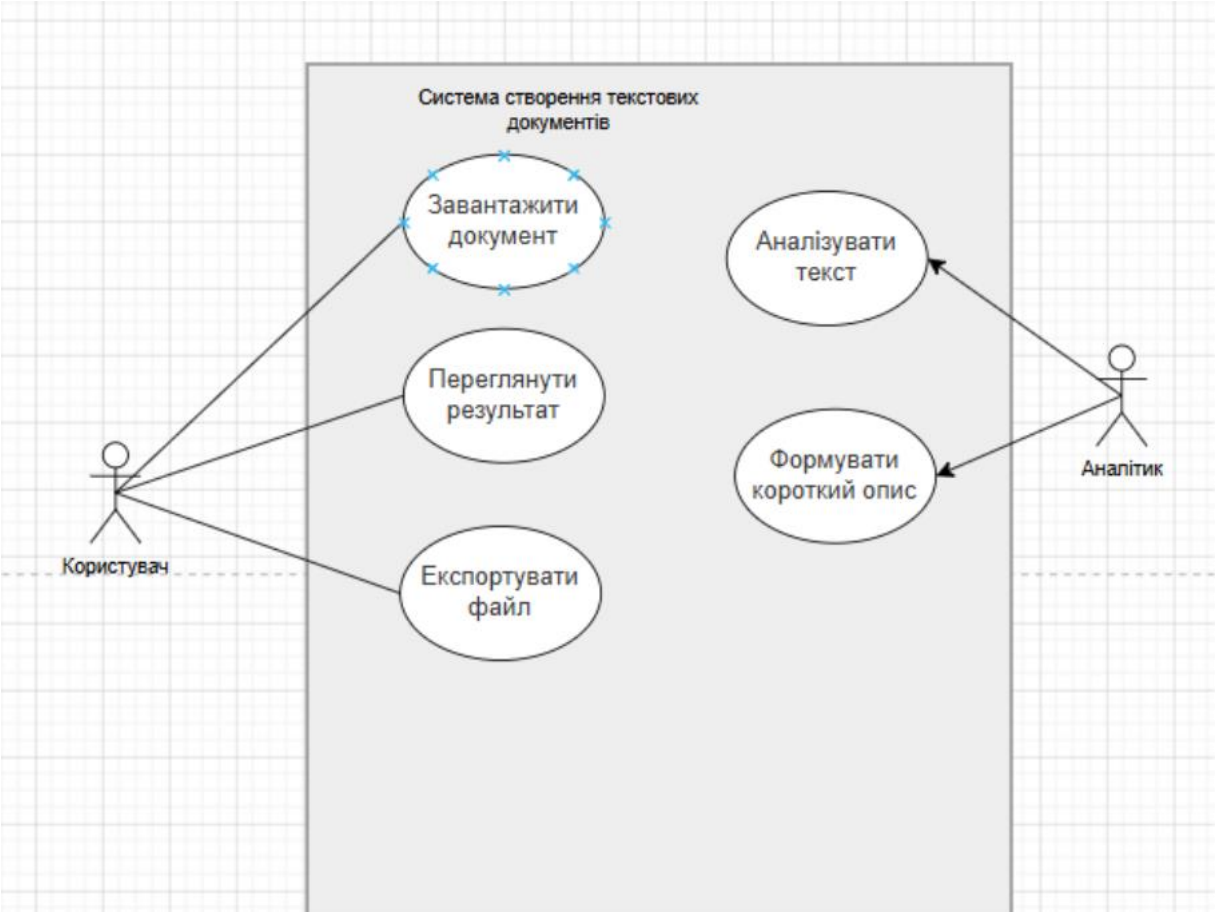


Рис. 2.1 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- «Користувач»;
- «Аналітик».

Актор «Користувач» включає такі прецеденти:

- завантажити документ;
- переглянути результат;
- експортувати файл.

Актор «Аналітик» включає такі прецеденти:

- аналізувати текст;
- формувати короткий опис.

Прецеденти системи певним чином пов’язані між собою та утворюють послідовний процес обробки текстового документа.

Розглянемо детальніше основні сценарії використання системи.

Варіант використання: Завантажити документ

Актор: Користувач

Опис: Даний сценарій описує процес відкриття текстового документа для подальшого аналізу системою автоматичного створення короткого викладу тексту. Користувач обирає файл на комп’ютері, після чого система перевіряє формат документа та виконує його зчитування. Після успішного завантаження документ передається на подальший аналіз.

Основний потік:

- користувач відкриває програму STYSLO AI;
- користувач натискає кнопку відкриття файлу;
- система відкриває вікно вибору документа;
- користувач обирає файл формату TXT, DOCX або PDF;
- система перевіряє формат документа;
- система виконує зчитування тексту;
- документ передається на аналіз.

Альтернативні потоки:

- якщо файл має непідтримуваний формат, система виводить повідомлення про помилку;
- якщо документ пошкоджений або недоступний, система повідомляє про неможливість його відкриття;
- якщо файл не містить тексту, система повідомляє про відсутність даних для аналізу.

Використання сценарію «Завантажити документ» дозволяє користувачу підготувати текстовий файл для подальшої автоматичної обробки та створення короткого викладу.

Розглянемо ще один сценарій використання.

Варіант використання: Формувати короткий опис

Актор: Аналітик

Опис: Даний сценарій описує процес аналізу текстового документа та автоматичного створення короткого викладу. Після отримання документа система виконує аналіз тексту за допомогою локального алгоритму або AI-режиму та формує стислий опис документа.

Основний потік:

- аналітик отримує текст документа;
- система запускає процес аналізу тексту;
- виконується обробка документа;
- формується короткий опис;
- результат передається користувачу;
- користувач переглядає отриманий результат;
- користувач може експортувати файл у необхідному форматі.

Альтернативні потоки:

- якщо під час аналізу виникає помилка, система виводить повідомлення користувачу;
- якщо AI-модель недоступна, система пропонує використати локальний режим;
- якщо документ має занадто великий обсяг, система виконує обробку тексту частинами.

Використання сценарію «Формувати короткий опис» дозволяє автоматизувати процес аналізу текстових документів та швидко отримувати стислий виклад основного змісту документа.

2.2.2 Діаграма послідовності. Діаграма послідовності використовується для моделювання взаємодії між окремими учасниками системи у часовому порядку. Вона дозволяє відобразити послідовність обміну повідомленнями між користувачем та компонентами програмного забезпечення під час виконання певного сценарію роботи системи.

Для програмного забезпечення STYSLO AI діаграма послідовності використовується для опису процесу автоматичного аналізу текстового документа та формування короткого викладу тексту. Вона демонструє взаємодію між користувачем та модулем аналізу системи під час обробки документа.

На діаграмі послідовності виділяються такі основні учасники:

- «Користувач»;
- «Аналітик».

Під об’єктом «Аналітик» у даному випадку розуміється модуль аналізу тексту, який виконує обробку документа та формування короткого викладу.

Розроблена діаграма послідовності представлена на рис. 2.2.

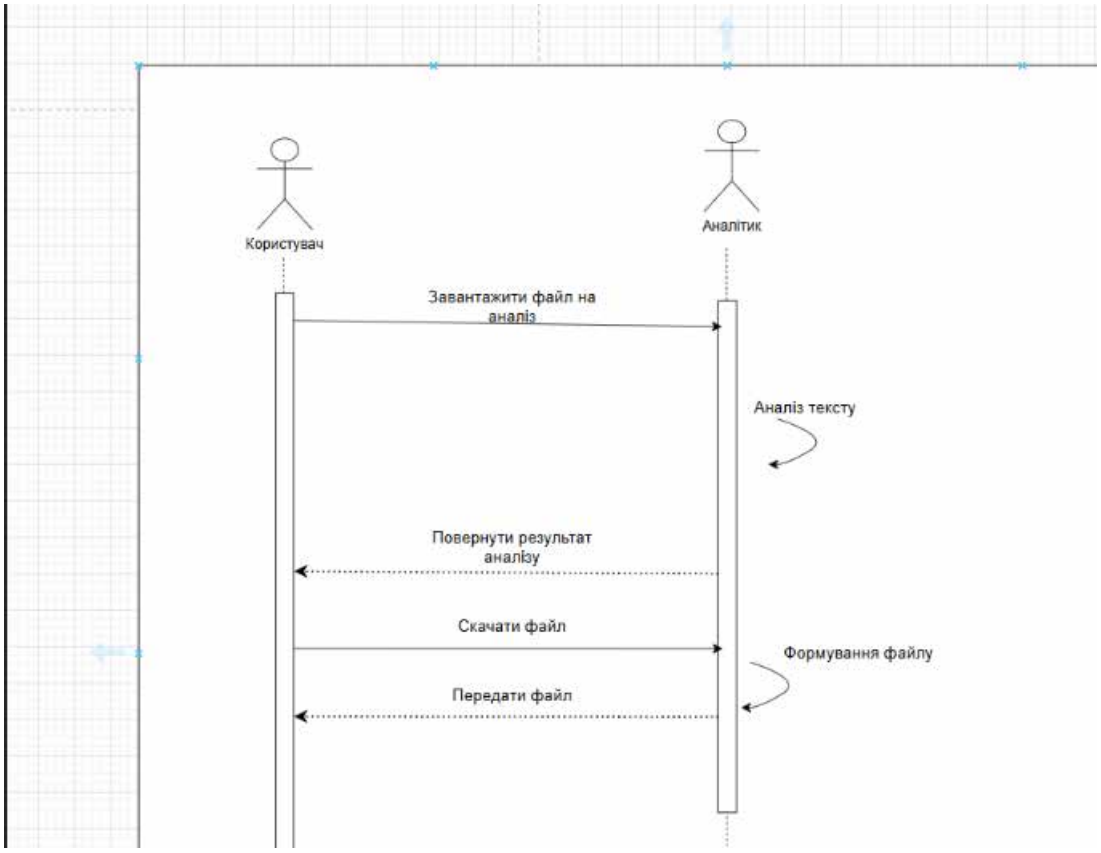


Рис. 2.2 Діаграма послідовності

Дана діаграма послідовності описує процес взаємодії користувача із системою автоматичного створення короткого викладу текстових документів. Вона побудована таким чином, щоб показати послідовність обміну повідомленнями між користувачем та модулем аналізу тексту під час обробки документа.

Спочатку користувач завантажує файл для подальшого аналізу. Після отримання документа система запускає процес аналізу тексту, під час якого виконується обробка вмісту документа та визначення основних частин тексту. На цьому етапі модуль аналізу виконує автоматичне формування короткого викладу документа.

Після завершення аналізу система повертає користувачу результат обробки. Користувач може переглянути сформований короткий виклад тексту та оцінити результат роботи системи. Це демонструє основну функцію програмного забезпечення - автоматичне створення стислого опису

документа без необхідності ручного аналізу тексту.

Наступним етапом є експорт результату. Користувач виконує команду скачування файлу, після чого система формує готовий документ та передає його користувачу у вибраному форматі.

Таким чином, діаграма послідовності демонструє логіку роботи програмного забезпечення STYSLO AI та відображає основні етапи взаємодії користувача із системою під час аналізу текстового документа, формування короткого викладу та збереження результату обробки.

2.2.3 Діаграма активності. Діаграма активності використовується для моделювання логіки роботи програмної системи та послідовності виконання окремих процесів. Вона дозволяє відобразити порядок дій користувача та системи, а також показати взаємозв’язки між окремими етапами обробки інформації.

На відміну від діаграми послідовності, яка демонструє взаємодію між об’єктами у часовому порядку, діаграма активності зосереджується на логіці виконання процесів, умовах переходу між етапами та альтернативних варіантах роботи системи.

У програмному забезпеченні STYSLO AI діаграма активності використовується для моделювання процесу автоматичного створення короткого викладу текстових документів. Вона дозволяє наочно показати послідовність виконання основних операцій системи - від завантаження документа до отримання готового результату.

Діаграма активності включає початкову точку процесу, дії користувача та системи, умовні переходи між окремими етапами роботи, а також завершення процесу після отримання результату обробки.

Розроблена діаграма активності представлена на рис. 2.3

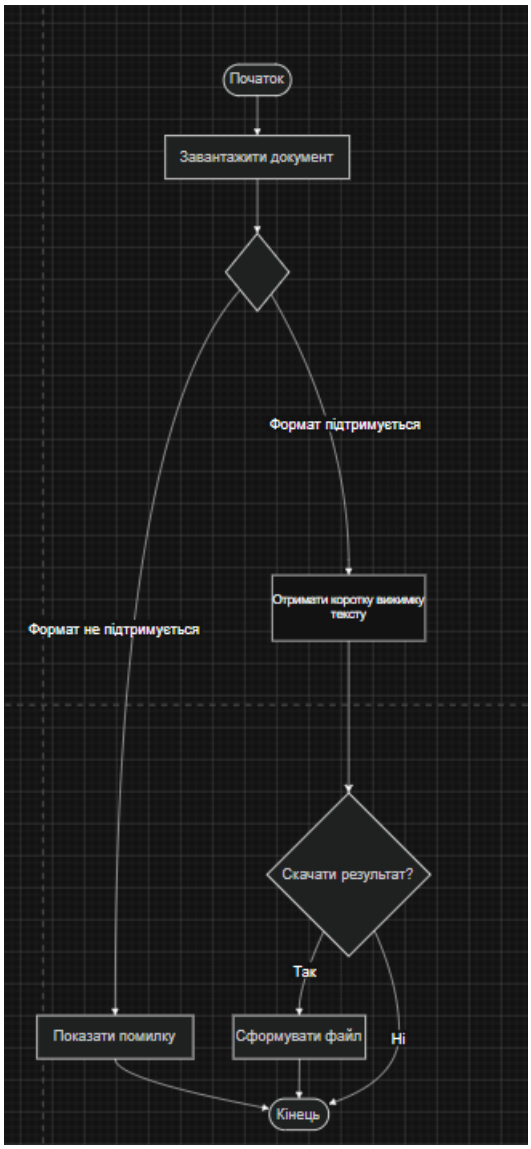


Рис. 2.3 Діаграма активності

Дана діаграма активності демонструє логіку роботи програмного забезпечення STYSLO AI під час аналізу текстового документа та створення короткого викладу.

На початку роботи користувач відкриває програму та завантажує файл для подальшого аналізу. Після цього система виконує перевірку формату документа та зчитування текстової інформації.

Далі запускається процес аналізу тексту. На цьому етапі система визначає основні частини документа та виконує формування короткого викладу тексту. Залежно від обраного режиму роботи може використовуватись

локальний алгоритм аналізу або AI-режим із використанням сервісу Ollama.

Після завершення аналізу система формує результат та відображає його користувачу. Користувач має можливість переглянути створений короткий виклад та виконати експорт результату у необхідному форматі.

Діаграма також відображає альтернативні сценарії роботи системи. Наприклад, якщо документ має некоректний формат або містить помилки, система повідомляє користувача про проблему та завершує процес обробки. У випадку виникнення помилок під час AI-аналізу система може запропонувати використання локального режиму роботи.

Таким чином, діаграма активності дозволяє наочно відобразити логіку функціонування програмного забезпечення STYSLO AI та демонструє основні етапи автоматичного створення короткого викладу текстових документів.

2.3 Абстракції предметної області

Абстракції предметної області в UML використовуються для узагальненого опису основних сутностей системи, їх властивостей та взаємозв’язків. Вони дозволяють виділити ключові компоненти програмного забезпечення та спростити розуміння структури системи без детального опису технічної реалізації.

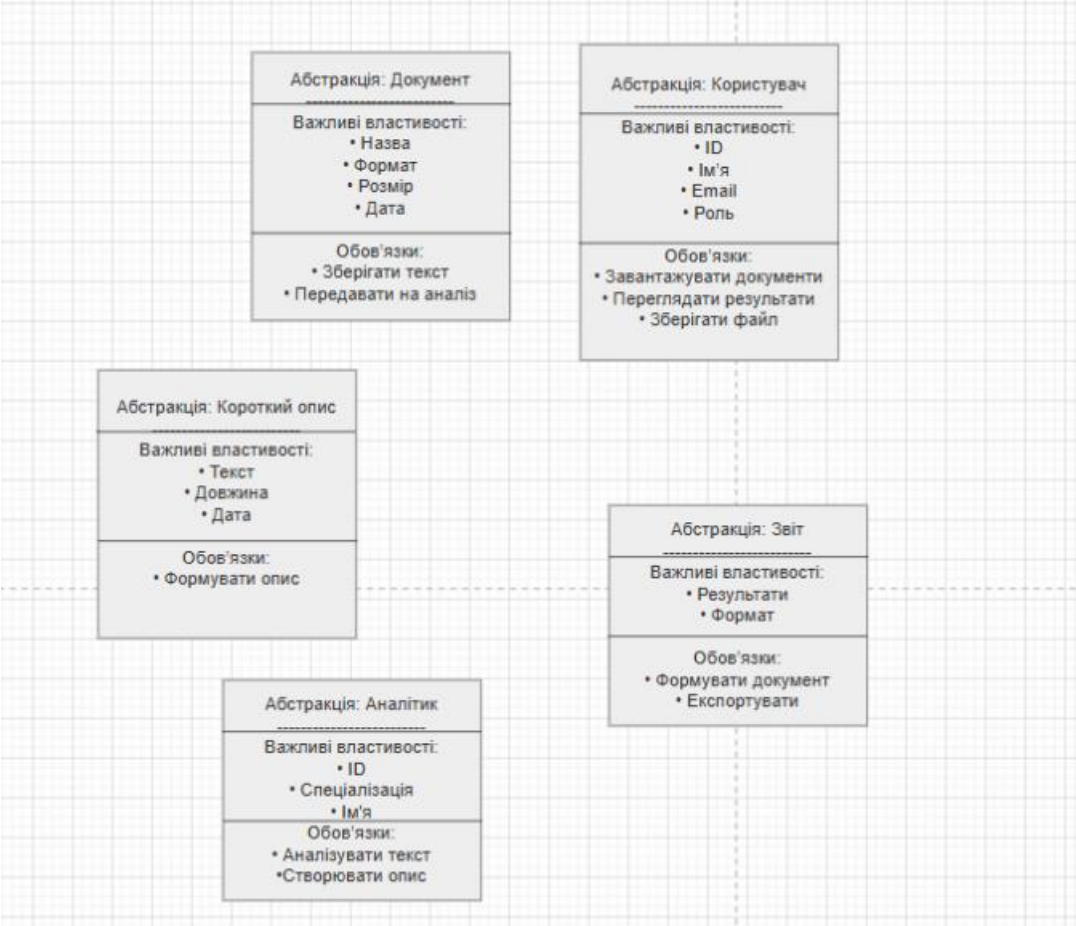
Основною метою використання абстракцій є формалізація предметної області та визначення головних елементів, які беруть участь у роботі програмного забезпечення. Завдяки цьому стає можливим побудувати логічну модель системи, визначити основні функції окремих компонентів та описати їх взаємодію між собою.

Абстракції дозволяють спростити складну структуру програмного забезпечення та зосередитися лише на ключових елементах системи. Вони

використовуються як основа для подальшого створення UML-діаграм, проєктування архітектури системи та реалізації програмного продукту.

У програмному забезпеченні STYSLO AI абстракції предметної області використовуються для опису основних сутностей, пов’язаних із процесом автоматичного створення короткого викладу текстових документів.

Розроблені абстракції предметної області представлені на рис. 2.4.



У системі виділено дві основні абстракції:

- «Документ»;
- «Короткий опис».

Абстракція «Документ» описує текстовий файл, який завантажується користувачем у систему для подальшого аналізу. Дана сутність містить основні характеристики документа, зокрема назву файлу, формат документа,

текстовий вміст та розмір файлу.

Основними функціями абстракції «Документ» є:

- завантаження документа у систему;
- перевірка формату файлу;
- зчитування текстового вмісту;
- передача документа на аналіз.

Абстракція «Короткий опис» описує результат роботи системи після аналізу текстового документа. Вона містить сформований короткий виклад тексту, параметри обробки та інформацію про режим роботи системи.

Основними функціями абстракції «Короткий опис» є:

- формування короткого викладу тексту;
- збереження результату обробки;
- експорт результату у файл;
- відображення результату користувачу.

Дані абстракції тісно пов’язані між собою, оскільки короткий опис формується на основі текстового документа, який був завантажений користувачем у систему. Саме взаємодія цих двох сутностей забезпечує основну функціональність програмного забезпечення STYSLO AI.

Використання абстракцій предметної області дозволяє структурувати основні елементи системи, спростити процес моделювання та забезпечити більш зрозуміле представлення логіки роботи програмного забезпечення.

2.4 Діаграма класів

Діаграма класів (Class Diagram) є одним із основних інструментів UML для моделювання статичної структури програмної системи. Вона використовується для відображення основних класів системи, їх атрибутів,

методів та взаємозв’язків між окремими компонентами програмного забезпечення.

Діаграма класів дозволяє отримати загальне уявлення про внутрішню структуру системи та визначити, як окремі класи взаємодіють між собою. Використання даної діаграми спрощує процес проєктування програмного забезпечення, дозволяє уникнути дублювання функціоналу та забезпечує більш зрозумілу організацію програмного коду.

Класи на діаграмі представляються у вигляді прямокутників, які містять назву класу, його атрибути та методи. Зв’язки між класами дозволяють показати взаємодію між окремими компонентами системи, передачу даних та залежності між об’єктами.

Під час розробки програмного забезпечення STYSLO AI діаграма класів використовується для моделювання основних компонентів системи автоматичного створення короткого викладу текстових документів.

Розроблена діаграма класів представлена на рис. 2.5.

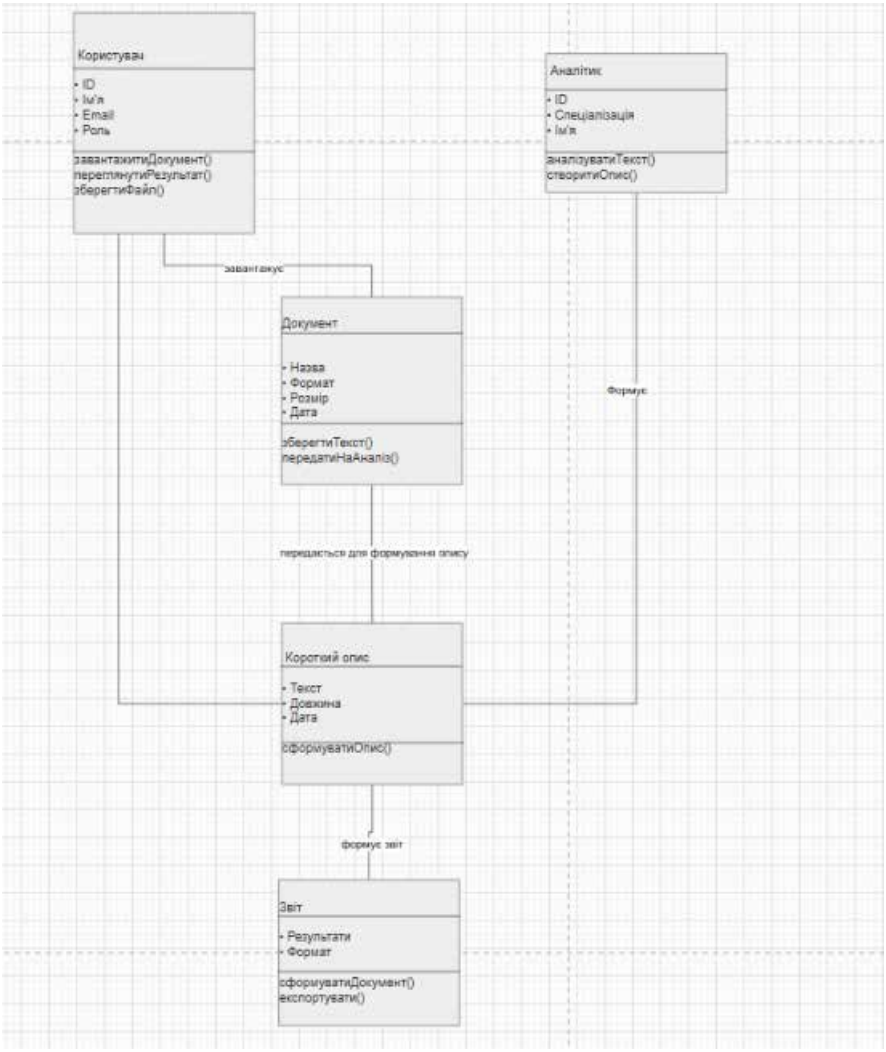


Рис. 2.5 Діаграма класів

Дана діаграма класів моделює структуру програмного забезпечення STYSLO AI та демонструє взаємодію між основними компонентами системи.

Одним із центральних класів є клас User, який описує користувача системи. Даний клас містить основні атрибути користувача та забезпечує взаємодію із програмним застосунком. Користувач має можливість завантажувати документи, запускати аналіз тексту та зберігати результати обробки.

Клас Document відповідає за роботу з текстовими файлами. Він містить атрибути, пов’язані з документом, зокрема назву файлу, формат документа та текстовий вміст. Основними функціями класу є відкриття документа, зчитування тексту та передача даних на аналіз.

Клас Summary описує результат роботи системи після обробки документа. Він містить короткий виклад тексту, параметри аналізу та інформацію про

режим роботи системи. Даний клас забезпечує формування, відображення та експорт результатів обробки.

Для реалізації AI-аналізу використовується клас OllamaService. Він відповідає за взаємодію із локальним AI-сервісом Ollama та забезпечує передачу тексту для подальшої AI-обробки. Даний клас використовується під час роботи AI-режиму системи.

Клас Database забезпечує роботу з локальною базою даних SQLite. Його основними функціями є збереження історії обробки документів, запис результатів аналізу та отримання інформації з бази даних.

Окремо на діаграмі представлений клас FileManager, який відповідає за імпорт та експорт файлів різних форматів. Він забезпечує підтримку роботи з документами TXT, DOCX та PDF.

Зв’язки між класами демонструють взаємодію компонентів системи. Користувач працює з документами та результатами обробки, модуль аналізу використовує дані документа для формування короткого викладу, а результати зберігаються у базі даних SQLite.

Діаграма класів демонструє логічну структуру програмного забезпечення STYSLO AI та дозволяє краще зрозуміти організацію системи автоматичного створення короткого викладу текстових документів. Використання даної діаграми спрощує процес подальшої реалізації, тестування та супроводу програмного забезпечення.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

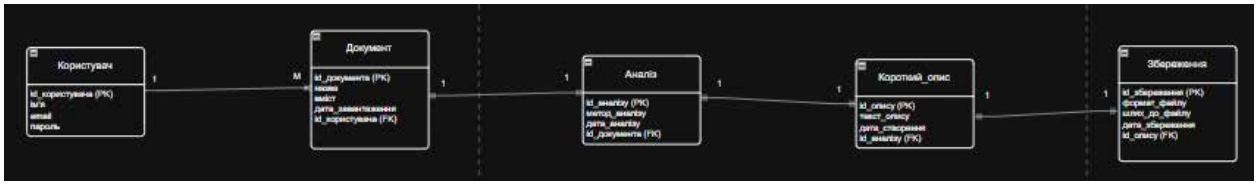
Логічне моделювання бази даних є важливим етапом розробки програмного забезпечення, оскільки воно дозволяє визначити структуру збереження інформації, зв’язки між даними та організацію роботи системи. Для побудови логічної моделі використовуються ER-діаграми, які наочно відображають основні сутності предметної області, їх атрибути та взаємозв’язки між окремими компонентами системи.

ER-діаграма (Entity-Relationship Diagram) є графічним представленням структури бази даних. Вона використовується для моделювання інформаційної системи та дозволяє визначити основні таблиці, ключові поля та зв’язки між сутностями.

Основними елементами ER-діаграми є:

- сутності, які представляють об’єкти предметної області;
- атрибути, що описують характеристики сутностей;
- зв’язки між таблицями;
- первинні та зовнішні ключі.

Використання ER-діаграм дозволяє спростити процес проєктування бази даних, забезпечити цілісність інформації та уникнути дублювання даних. Крім цього, логічна модель даних використовується як основа для подальшої реалізації бази даних у програмному забезпеченні.



Розроблена ER-діаграма бази даних представлена на рис. 3.1.

Рис. 3.1 - ER-діаграма бази даних STYSLO AI

Дана ER-діаграма моделює структуру бази даних програмного забезпечення STYSLO AI для автоматичного створення короткого викладу текстових документів. Вона демонструє основні сутності системи, їх атрибути та взаємозв’язки між окремими таблицями.

Однією з основних сутностей системи є таблиця «Користувач». Вона містить інформацію про користувача системи, зокрема ідентифікатор користувача, ім’я та пароль. Дана таблиця використовується для авторизації користувачів та забезпечення доступу до функціоналу програмного забезпечення.

Таблиця «Документ» призначена для збереження інформації про завантажені текстові файли. Вона містить назву документа, дату завантаження та посилання на користувача, який виконав завантаження файлу. Один користувач може працювати з декількома документами, що реалізовано через відповідний зв’язок між таблицями.

Таблиця «Аналіз» використовується для збереження результатів аналізу текстового документа. Вона містить тип аналізу, дату виконання аналізу та посилання на документ, який був оброблений системою. Дана сутність відповідає за процес автоматичного аналізу тексту та формування короткого викладу.

Таблиця «Короткий опис» містить сформований системою короткий виклад документа. У таблиці зберігається текст короткого опису, дата створення та посилання на відповідний аналіз документа. Це дозволяє пов’язати результат обробки із конкретним процесом аналізу тексту.

Таблиця «Збереження» використовується для збереження інформації про експорт результатів роботи системи. Вона містить формат файлу, дату збереження та посилання на короткий опис документа. Завдяки цьому система підтримує експорт результатів у різних форматах.

Зв’язки між таблицями забезпечують цілісність даних та дозволяють організувати послідовний процес роботи системи - від завантаження документа користувачем до формування короткого викладу та збереження результату.

Таким чином, ER-діаграма демонструє логічну структуру бази даних програмного забезпечення STYSLO AI та відображає основні етапи роботи системи автоматичного створення короткого викладу текстових документів.

3.2 Вибір системи управління базою даних та її реалізація

При розробці програмного забезпечення STYSLO AI важливим етапом стало проєктування та реалізація бази даних. База даних використовується для збереження інформації про користувачів, завантажені документи, результати аналізу тексту та історію роботи системи. Від правильного вибору системи управління базами даних залежить стабільність роботи програмного забезпечення, швидкість обробки інформації та надійність збереження даних.

Система управління базами даних (СУБД) забезпечує створення, збереження, редагування та отримання інформації із бази даних. Існує велика кількість СУБД, які відрізняються за способом організації даних, продуктивністю та сферою використання.

Реляційні СУБД використовують таблиці для збереження інформації та підтримують мову SQL для роботи з даними. До найбільш відомих реляційних СУБД належать MySQL, PostgreSQL, Microsoft SQL Server та SQLite. Дані системи підтримують зв’язки між таблицями, забезпечують цілісність інформації та дозволяють виконувати складні запити до бази даних.

Для програмного забезпечення STYSLO AI було обрано SQLite як

систему управління базами даних. Основною причиною вибору SQLite є простота використання, відсутність необхідності встановлення окремого серверу бази даних та зручна інтеграція із мовою програмування Python.

SQLite є вбудованою реляційною СУБД, яка зберігає всю базу даних у вигляді одного файлу. Це дозволяє спростити розгортання програмного забезпечення та забезпечує локальну роботу системи без необхідності використання зовнішніх серверів.

Під час вибору СУБД були враховані такі критерії:

- підтримка реляційної моделі даних;
- простота інтеграції із Python;
- можливість локального збереження даних;
- невисокі вимоги до ресурсів комп’ютера;
- стабільність та надійність роботи;
- підтримка SQL-запитів;
- простота резервного копіювання бази даних.

Використання SQLite дозволяє забезпечити швидке зчитування та запис інформації, а також спрощує процес перенесення програмного забезпечення між різними комп’ютерами.

Для моделювання структури бази даних було використано ER-діаграму, створену у середовищі Draw.io. Побудована модель дозволила визначити основні сутності системи, їх атрибути та зв’язки між окремими таблицями.

Таким чином було створено структуру бази даних програмного забезпечення STYSLO AI, яка представлена на рис. 3.2.

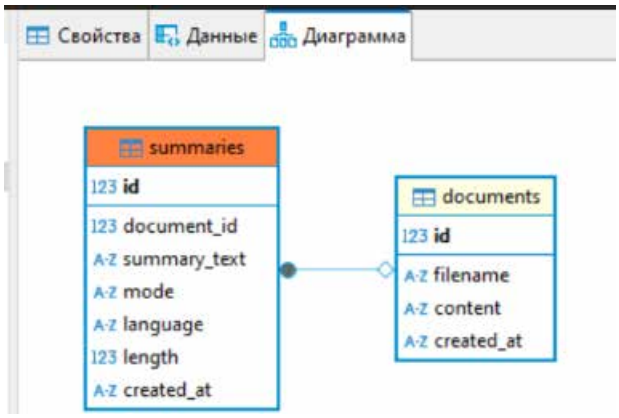


Рис. 3.2 - Структура бази даних STYSLO AI

У структурі бази даних реалізовано декілька основних таблиць.

Таблиця «Користувач» містить інформацію про користувачів системи. У ній зберігаються ідентифікатор користувача, ім’я та пароль. Дана таблиця використовується для авторизації користувачів та забезпечення доступу до функцій програмного забезпечення.

Таблиця «Документ» призначена для збереження інформації про текстові файли, які завантажуються користувачем у систему. Вона містить назву документа, дату завантаження та зовнішній ключ для зв’язку із користувачем.

Таблиця «Аналіз» використовується для збереження інформації про процес аналізу документа. Вона містить тип аналізу, дату виконання обробки та посилання на відповідний документ.

Таблиця «Короткий опис» містить результати автоматичного створення короткого викладу тексту. У ній зберігається текст короткого опису, дата створення та зовнішній ключ для зв’язку із таблицею аналізу.

Таблиця «Збереження» використовується для збереження інформації про експорт результатів роботи системи. Вона містить формат файлу, дату збереження та посилання на відповідний короткий опис документа.

Між таблицями реалізовані зв’язки типу «один-до-багатьох» та «один-до-одного». Один користувач може завантажувати декілька документів, кожен документ проходить аналіз, після чого система формує короткий опис та забезпечує можливість його збереження у різних форматах.

Для роботи із базою даних у програмному забезпеченні використовується бібліотека sqlite3 мови програмування Python. Вона забезпечує створення таблиць, виконання SQL-запитів, запис та отримання інформації із бази даних.

Використання SQLite у програмному забезпеченні STYSLO AI дозволило забезпечити стабільну локальну роботу системи, спростити процес збереження інформації та реалізувати підтримку історії обробки текстових документів.

.

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення визначає структуру системи, принципи взаємодії між окремими компонентами та організацію обробки даних. Правильно побудована архітектура дозволяє забезпечити стабільність роботи системи, спрощує підтримку програмного забезпечення та створює можливість для подальшого розширення функціональних можливостей.

Існує декілька основних підходів до побудови архітектури програмного забезпечення, серед яких монолітна архітектура, клієнт-серверна модель, багаторівнева архітектура та мікросервісна архітектура. Кожен із цих підходів має свої переваги та особливості використання залежно від типу програмного забезпечення.

Під час розробки програмного забезпечення STYSLO AI було використано багаторівневу архітектуру, яка дозволяє розділити систему на окремі функціональні компоненти. Такий підхід забезпечує більш зрозумілу структуру програмного забезпечення та спрощує взаємодію між окремими модулями системи.

Розроблена архітектура програмного забезпечення представлена на рис. 3.3.

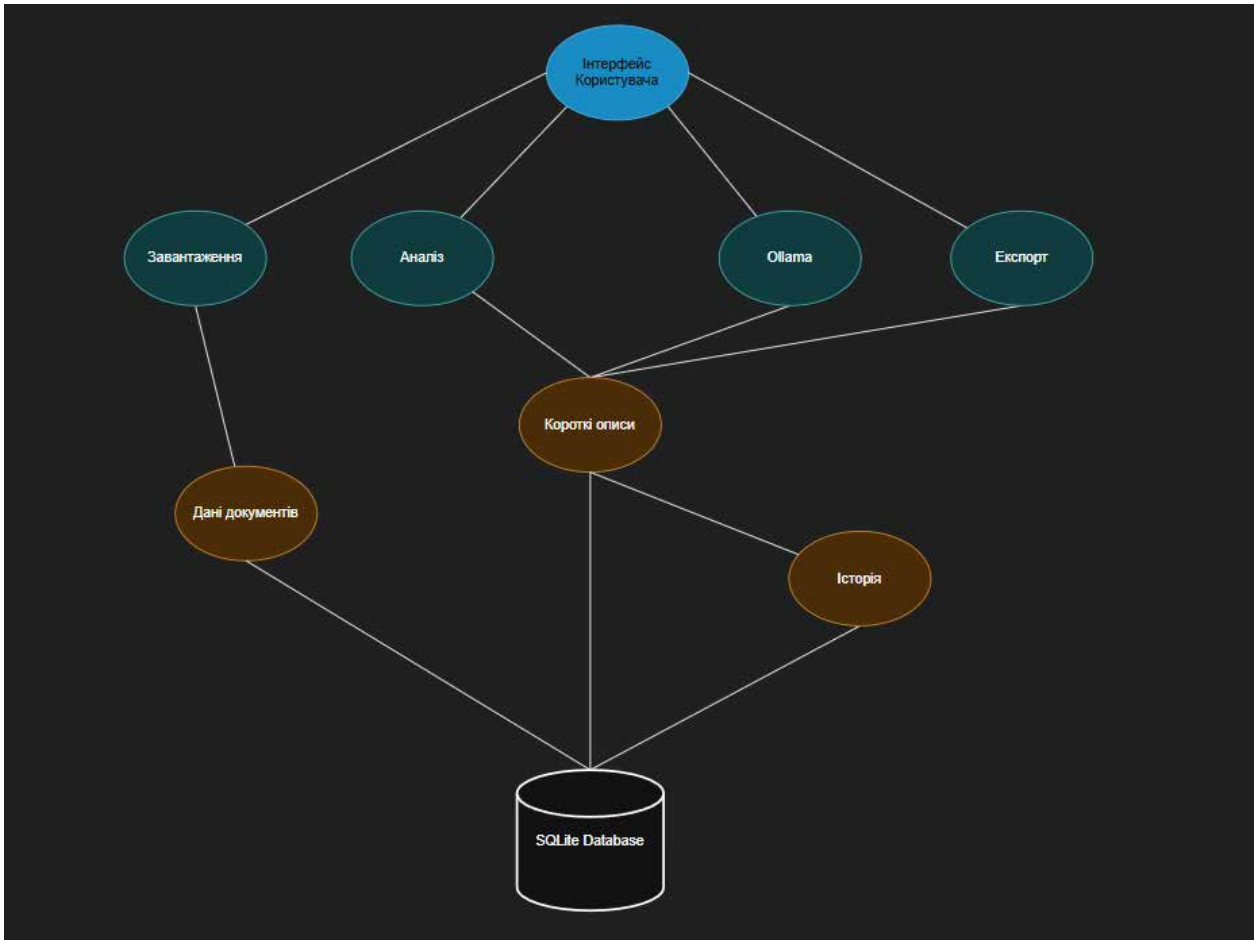


Рис. 3.3 - Архітектура програмного забезпечення STYSLO AI

Дана архітектура складається з декількох основних рівнів та функціональних модулів, які забезпечують роботу системи автоматичного створення короткого викладу текстових документів.

На верхньому рівні розташований інтерфейс користувача. Саме через нього користувач взаємодіє із програмним забезпеченням. Інтерфейс забезпечує можливість завантаження документів, запуску аналізу тексту, використання AI-режиму Ollama та експорту результатів у різних форматах.

Другий рівень складається з основних функціональних модулів системи. Модуль «Завантаження» відповідає за відкриття та обробку текстових файлів, модуль «Аналіз» забезпечує виконання аналізу тексту та формування короткого викладу документа. Модуль «Ollama» використовується для AI-аналізу тексту за допомогою локальної мовної моделі. Модуль «Експорт» забезпечує збереження результатів роботи системи у форматах TXT, DOCX та PDF.

Третій рівень відповідає за обробку та збереження даних системи. Компонент «Дані документів» використовується для роботи із завантаженими файлами та текстовим вмістом документів. Компонент «Короткі описи» містить результати аналізу тексту та сформовані короткі виклади документів. Компонент «Історія» забезпечує збереження інформації про попередні результати роботи системи.

На нижньому рівні знаходиться база даних SQLite Database, яка використовується для локального збереження інформації. У базі даних зберігаються відомості про документи, результати аналізу тексту та історія роботи користувача із системою.

Зв’язки між компонентами демонструють процес взаємодії окремих модулів системи. Інтерфейс користувача забезпечує доступ до функціональних компонентів, модулі аналізу та експорту працюють із короткими описами документів, а результати роботи системи зберігаються у базі даних SQLite.

Використання багаторівневої архітектури дозволило забезпечити чітке розділення функцій між окремими компонентами програмного забезпечення, спростити структуру системи та створити основу для подальшого розвитку програмного забезпечення STYSLO AI.

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. Діаграма пакетів у UML є інструментом для візуалізації високорівневої організації програмного забезпечення та відображення логічних залежностей між різними частинами системи. Пакет у такій діаграмі – це групування класів, інтерфейсів або інших елементів моделі, яке дозволяє структурувати програму на окремі логічні компоненти та приховати деталі реалізації всередині пакетів.

Основна мета діаграми пакетів – продемонструвати структуру системи на рівні модулів і взаємозв'язки між ними, що особливо корисно для великих проєктів, де багато класів та компонентів. Вона показує, які частини системи залежать одна від одної, і допомагає визначити точки інтеграції, а також спростити підтримку та масштабування програмного забезпечення [18].

У діаграмі пакетів зв'язки між пакетами можуть бути різних типів: залежності, імпорти, реалізації або асоціації. Це дозволяє розробникам швидко оцінити, які модулі впливають на інші, і визначити області, які можна модифікувати без ризику порушення роботи системи. Крім того, діаграма пакетів допомагає визначити логічні блоки для командної роботи, оскільки кожен пакет можна призначити конкретній групі розробників.

Діаграми пакетів використовуються як для проєктування нових систем, так і для документування існуючих, забезпечуючи узгоджене уявлення про структуру програмного забезпечення на високому рівні. Вони добре доповнюють діаграми класів, оскільки дозволяють бачити не окремі класи, а їх логічні об'єднання, і тим самим спрощують управління складними проєктами.

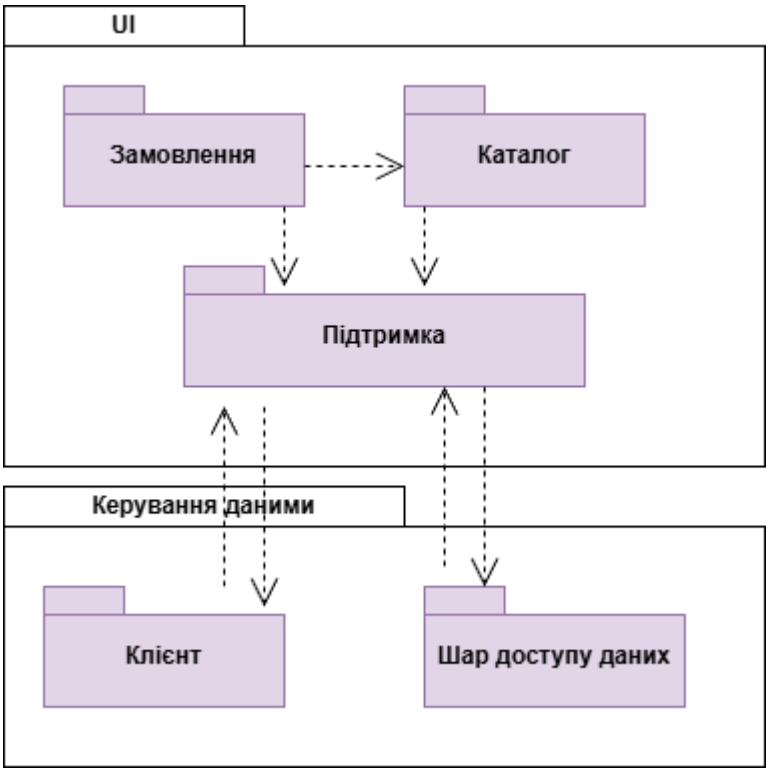


Рис. 3.4 Діаграма пакетів

Ця діаграма пакетів показує, як програмна система розділена на окремі модулі та рівні, кожен з яких відповідає за свою частину функціональності. У верхньому шарі знаходиться UI, тобто інтерфейс користувача. Тут виділені пакети «Замовлення» та «Каталог», які забезпечують роботу з оформленням покупок і переглядом товарів. Між ними є залежність: модуль замовлень може звертатися до каталогу, щоб отримати інформацію про товари.

Обидва ці пакети пов’язані з центральним модулем Підтримка, який виступає посередником між інтерфейсом і нижчими рівнями системи. Він відповідає за обробку звернень, інтеграцію бізнес-логіки та передачу даних далі.

У нижньому шарі «Керування даними» розташовані два пакети: Клієнт і Шар доступу даних. Вони отримують інформацію від модуля підтримки. «Клієнт» відповідає за взаємодію з користувачем у контексті даних, а «Шар доступу даних» забезпечує роботу з базою даних, тобто фактичне збереження та отримання інформації.

Таким чином, архітектура побудована за принципом багатошаровості: інтерфейс користувача звертається до бізнес-логіки, бізнес-логіка працює з клієнтськими даними та шаром доступу, а той у свою чергу взаємодіє з базою даних. Це дозволяє чітко розділити відповідальність між пакетами й зробити систему більш зрозумілою та масштабованою.

3.5 Вибір інструментарію для створення програмного забезпечення

Для розробки програмного було здійснено ретельний вибір інструментів та технологій, які дозволяють створити сучасну, надійну та масштабовану систему. Основними критеріями при виборі програмного забезпечення були функціональні можливості, сумісність з веб-технологіями, популярність серед розробників, активна підтримка спільноти, наявність документації, а також здатність забезпечити ефективну інтеграцію з різними сервісами та API. Такі вимоги обумовлені необхідністю реалізації чат-бота, який здатний в режимі реального часу взаємодіяти з користувачами, обробляти їх запити та вести облік замовлень у структурованій формі.

Серверна частина додатку реалізується на мові PHP, що дозволяє обробляти запити клієнтів, виконувати бізнес-логіку та працювати з базою даних. Для організації коду, управління бізнес-логікою та забезпечення масштабованості використовується фреймворк Symfony. Він надає багаторівневу архітектуру додатку, яка дозволяє відокремити логіку презентації, бізнес-процеси та управління даними, підвищує повторне використання компонентів і полегшує підтримку та розвиток проєкту у майбутньому. Symfony також забезпечує безпеку додатку, обробку помилок, систему роутингу запитів та інтеграцію з зовнішніми сервісами.

Для збереження та обробки даних обрана реляційна система управління базами даних MySQL, яка забезпечує надійне зберігання інформації, підтримку складних запитів та узгодженість даних. MySQL

дозволяє ефективно працювати з великими обсягами інформації, зберігати історію взаємодій користувачів, облік замовлень та повідомлень, а також здійснювати аналітику дій користувачів. Для проєктування структури бази даних, візуалізації зв’язків між таблицями та подальшої генерації SQL-коду застосовується MySQL Workbench. Це дозволяє розробникам отримати наочну картину структури даних та полегшує процес внесення змін у базу даних [19].

Інтерактивна взаємодія користувача з системою забезпечується через Telegram API, що дозволяє створити чат-бота, який у режимі реального часу отримує повідомлення від користувачів, обробляє їх та надсилає відповіді. API Telegram дозволяє реалізувати функціонал автоматичного оброблення запитів клієнтів, відправку повідомлень про статус замовлення, створення кнопок та меню для зручної навігації, а також інтегрувати бота з іншими сервісами для підвищення ефективності взаємодії.

Розробка та написання коду ведеться у середовищі Visual Studio Code, яке підтримує PHP, HTML, CSS та JavaScript, а також надає інтеграцію з системами контролю версій, що дозволяє організувати ефективну роботу над проєктом, вести облік змін та спільну розробку. Вибір цього інструменту обумовлений його легкістю, високою функціональністю, наявністю численних плагінів та можливістю налаштування середовища відповідно до потреб розробника.

Використання цього комплексу інструментів і технологій забезпечує створення повнофункціональної, стабільної та масштабованої системи для автоматизації підтримки клієнтів у сфері електронної комерції. Комбінація PHP, Symfony, MySQL, Telegram API, MySQL Workbench та Visual Studio Code дозволяє інтегрувати всі рівні додатку — від інтерфейсу користувача до бази даних — у єдину систему, що забезпечує швидку, безпечну та надійну роботу чат-бота. Усі вибрані засоби дозволяють реалізувати основні функції системи, такі як реєстрація та автентифікація користувачів, облік замовлень,

ведення історії взаємодій та обробку запитів у режимі реального часу, а також забезпечують подальший розвиток і масштабування проєкту.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної та коректної роботи програмного забезпечення STYSLO AI були визначені основні вимоги до апаратного та програмного забезпечення. Дані вимоги враховують особливості роботи системи, обробку текстових документів, використання локальної AI-моделі Ollama та збереження результатів аналізу у базі даних SQLite.

Апаратні вимоги системи залежать від режиму роботи програмного забезпечення. Для локального аналізу тексту достатньо персонального комп’ютера або ноутбука із середніми технічними характеристиками. У випадку використання AI-режиму Ollama необхідна більша кількість оперативної пам’яті та вищий рівень продуктивності процесора.

Мінімальні апаратні вимоги для роботи системи:

- процесор Intel Core i3 або аналогічний;
- оперативна пам’ять не менше 4 ГБ;
- вільне місце на диску не менше 2 ГБ;
- стабільне підключення до Інтернету для завантаження моделей Ollama;
- операційна система Windows 10 або новіша.

Рекомендовані апаратні вимоги:

- процесор Intel Core i5 або вище;
- оперативна пам’ять від 8 ГБ;
- SSD-накопичувач для швидшої роботи системи;
- стабільне підключення до Інтернету;
- відеокарта з підтримкою сучасних технологій обробки даних.

Програмні вимоги включають операційну систему, середовище розробки та необхідні програмні компоненти для роботи системи.

Для розробки програмного забезпечення використовувалися:

- мова програмування Python;
- бібліотеки Python для роботи з текстом та файлами;
- SQLite для локального збереження даних;
- Ollama для AI-аналізу тексту;
- Visual Studio Code та PyCharm як середовище розробки;
- Draw.io для створення UML-діаграм та моделювання системи.

Для роботи програмного забезпечення необхідно встановити Python версії 3.10 або новішої, а також додаткові бібліотеки, які використовуються у проєкті. Програма підтримує роботу із текстовими файлами форматів TXT, DOCX та PDF.

Система також потребує встановлення Ollama для використання AI-режиму аналізу тексту. Ollama забезпечує локальну роботу мовної моделі без необхідності використання зовнішніх хмарних сервісів. Це дозволяє забезпечити швидшу обробку тексту та підвищити рівень конфіденційності даних користувача.

Для збереження інформації використовується база даних SQLite, яка не потребує встановлення окремого серверу баз даних та зберігає всі дані у локальному файлі.

Узгодження апаратних та програмних вимог дозволяє забезпечити стабільну роботу програмного забезпечення STYSLO AI, коректну обробку текстових документів та підтримку AI-функціоналу системи.

Також було створено діаграму розгортання для візуального представлення структури розгортання програмного забезпечення (рис. 4.1).

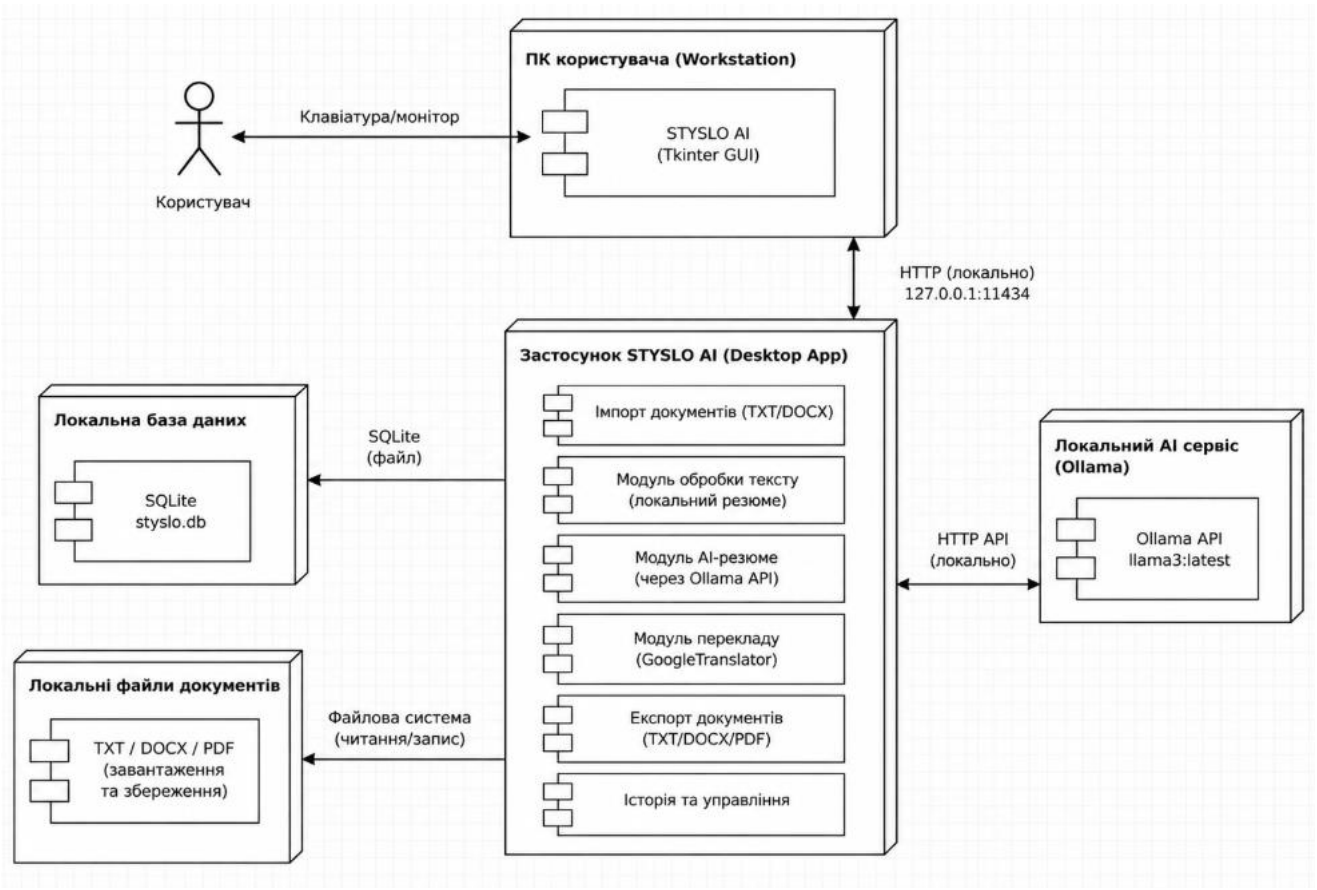


Рис. 4.1 Діаграма розгортання

Діаграма розгортання демонструє структуру взаємодії основних компонентів програмного забезпечення STYSLO AI у локальному середовищі.

Користувач взаємодіє із системою через графічний інтерфейс, реалізований за допомогою бібліотеки Tkinter. Інтерфейс запускається на персональному комп’ютері користувача та забезпечує доступ до основних функцій програмного забезпечення.

Основним компонентом системи є десктопний застосунок STYSLO AI, який виконує обробку текстових документів. До його складу входять модуль імпорту документів, модуль локальної обробки тексту, AI-модуль для створення короткого викладу тексту через Ollama API, модуль перекладу тексту та модуль експорту результатів у формати TXT, DOCX та PDF.

Для AI-аналізу тексту система використовує локальний сервіс Ollama, який працює через локальний HTTP API. Це дозволяє виконувати аналіз тексту без використання зовнішніх хмарних сервісів та забезпечує локальну обробку даних користувача.

Для збереження інформації використовується локальна база даних SQLite, яка зберігається у файлі styslo.db. У базі даних міститься інформація про результати аналізу тексту, короткі описи документів та історію роботи системи.

Система також працює із локальними файлами документів форматів TXT, DOCX та PDF. Завантаження та збереження файлів здійснюється через файлову систему операційної системи.

Використання локальної архітектури дозволяє забезпечити автономну роботу програмного забезпечення STYSLO AI, підвищити швидкість обробки тексту та забезпечити конфіденційність даних користувача.

4.2 Тестування системи

Тестування програмного забезпечення є важливим етапом розробки, який дозволяє перевірити коректність роботи системи, стабільність функціонування окремих модулів та відповідність програмного забезпечення поставленим вимогам.

Для перевірки працездатності програмного забезпечення було проведено функціональне тестування основних модулів системи. Тестування охоплювало перевірку локального режиму роботи, AI-обробки через Ollama, експорту результатів та обробки помилкових ситуацій.

№	Назва тест-кейсу	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Завантаження TXT-файлу	Файл формату TXT	Система відкриває	Файл успішно	Успішно

			файл та зчитує текст	відкрито, текст зчитано	
2	Завантаження DOCX-файлу	Файл формату DOCX	Система відкриває документ та отримує текст	Документ успішно оброблено	Успішно
3	Завантаження PDF-файлу	Файл формату PDF	Система зчитує текст із PDF	Текст із PDF зчитано	Успішно
4	Локальне резюмування	Документ + локальний режим	Система формує короткий виклад без використання AI	Короткий виклад сформовано	Успішно
5	AI-резюмування через Ollama	Документ + AI-режим	Система надсилає текст до Ollama та отримує результат	AI-резюме сформовано	Успішно
6	Відсутність Ollama	AI-режим, Ollama не запущена	Система показує повідомлення про помилку	Виведено повідомлення про недоступність Ollama	Успішно
7	Відсутність AI-моделі	AI-режим без потрібної моделі	Система повідомляє про відсутність моделі	Виведено список відсутніх моделей	Успішно
8	Обробка великого документа	Великий PDF/DOCX	Система обробляє текст частинами	Документ оброблено без зависання	Успішно

9	Експорт у TXT	Сформований короткий виклад	Система зберігає результат у TXT	TXT-файл створено	Успішно
10	Експорт у DOCX	Сформований короткий виклад	Система зберігає результат у DOCX	DOCX-файл створено	Успішно
11	Експорт у PDF	Сформований короткий виклад	Система зберігає результат у PDF	PDF-файл створено	Успішно
12	Відсутність інтернету	Локальний режим без інтернету	Система повинна працювати у локальному режимі	Локальна обробка виконується	Успішно

Під час тестування програмного забезпечення STYSLO AI було перевірено основні функціональні можливості системи, зокрема завантаження документів, аналіз тексту, AI-обробку через Ollama, експорт результатів та роботу локальної бази даних.

Після запуску програмного забезпечення користувач потрапляє у головне вікно системи, де доступні основні функції програми: відкриття документа, вибір режиму аналізу, використання AI-режиму Ollama, вибір мови обробки та експорт результатів у різних форматах (рис. 4.2).

Рис. 4.2 Початкове меню

Для перевірки роботи системи було виконано завантаження текстового документа у форматі DOCX. Після вибору файлу система автоматично зчитує текстовий вміст документа та відображає інформацію про файл, зокрема

кількість символів, слів та речень (рис. 4.3).

Рис. 4.3 - Завантаження текстового документа

Після запуску аналізу система виконує обробку тексту та автоматично формує короткий виклад документа. У процесі роботи відображається індикатор виконання обробки, що дозволяє користувачу контролювати стан виконання аналізу (рис. 4.4).

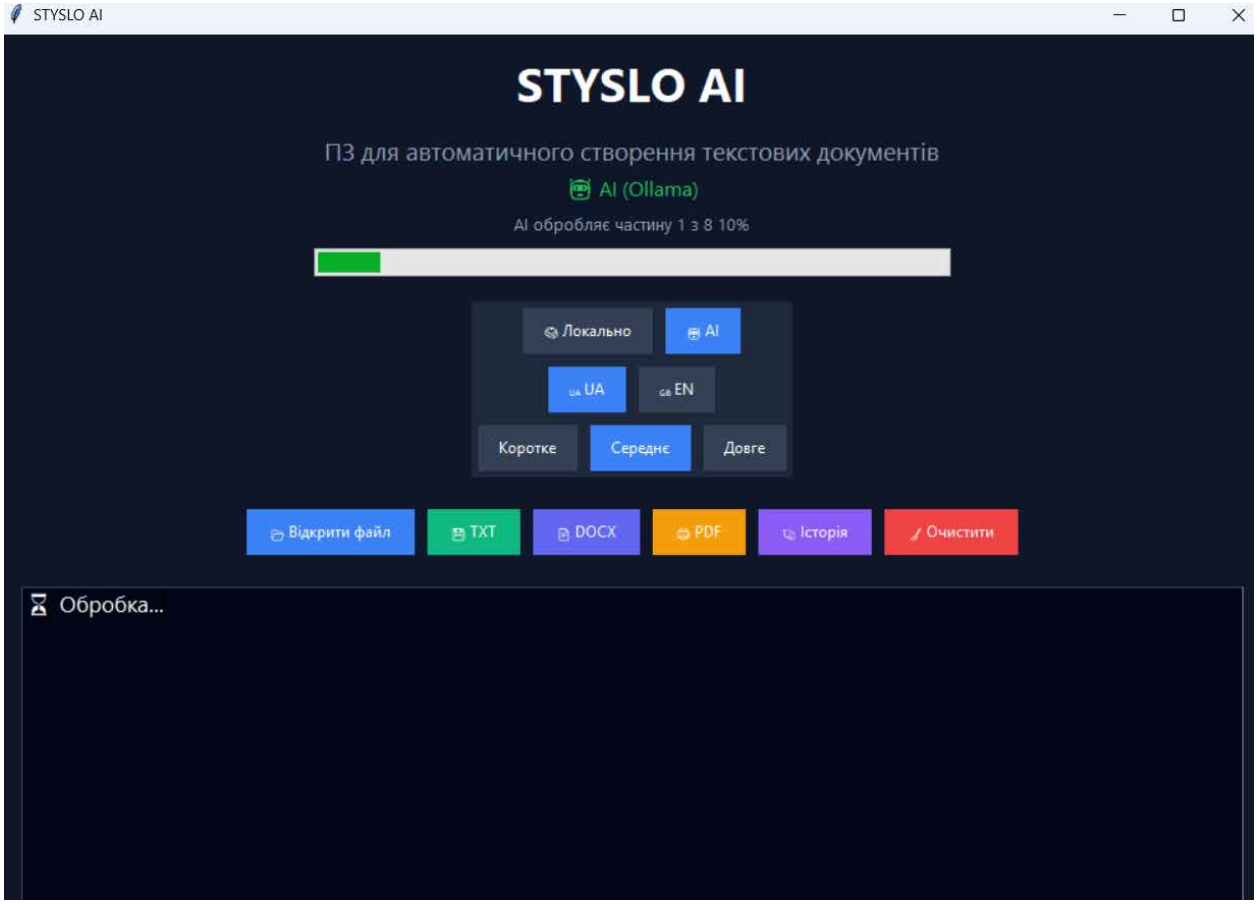


Рис. 4.4 - Процес аналізу тексту

Для перевірки функції експорту було виконано збереження результатів аналізу у формати TXT, DOCX та PDF. Система коректно створює файли та зберігає сформований короткий опис документа у вибраному форматі (рис. 4.5).

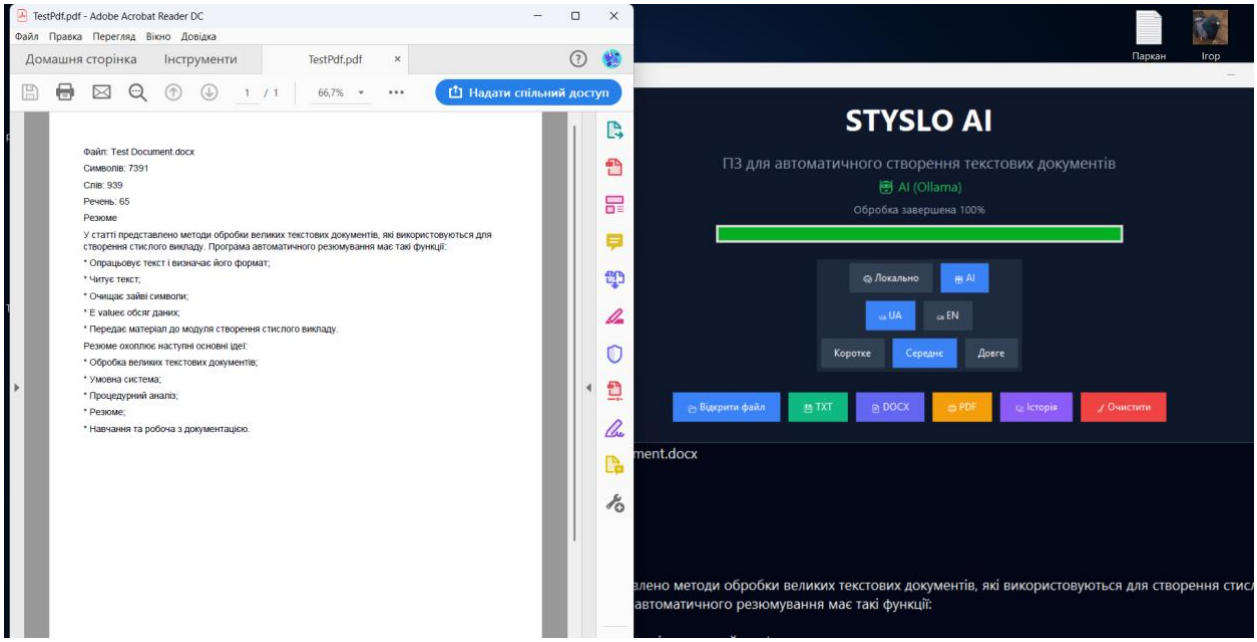


Рис. 4.5 - Експорт результатів аналізу

Також було протестовано роботу локальної бази даних SQLite та модуля історії. Після завершення аналізу інформація про документ та результати обробки автоматично зберігаються у базі даних, що дозволяє користувачу переглядати історію попередніх операцій (рис. 4.6).

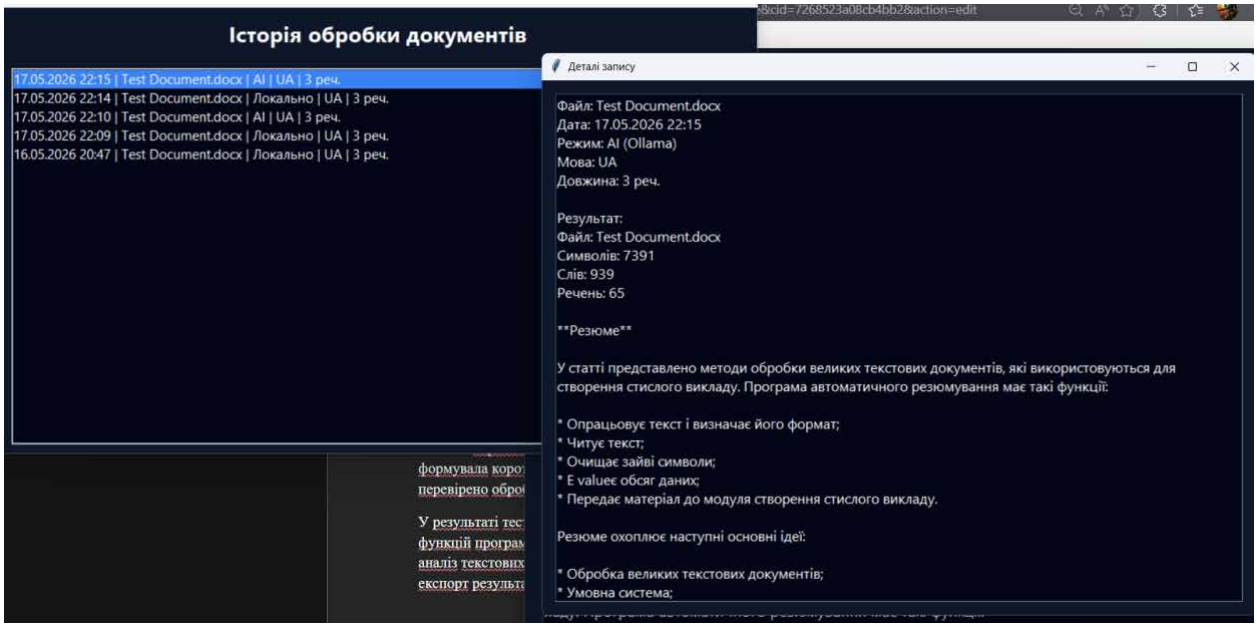


Рис. 4.6 - Перегляд історії обробки документів

Окремо було проведено тестування AI-режиму Ollama. Під час тестування система коректно виконувала взаємодію із локальним AI-сервісом та формувала короткі виклади тексту на основі мовної моделі. Також було перевірено обробку помилок у випадку відсутності підключення до Ollama.

У результаті тестування було підтверджено коректну роботу основних функцій програмного забезпечення STYSLO AI. Система стабільно виконує аналіз текстових документів, підтримує AI-режим обробки, забезпечує експорт результатів та збереження інформації у локальній базі даних SQLite.

ВИСНОВКИ

У даній бакалаврській роботі було виконано розробку програмного забезпечення STYSLO AI для автоматичного створення короткого викладу текстових документів. У процесі виконання роботи було проаналізовано предметну область, визначено основні вимоги до системи та реалізовано програмне забезпечення для обробки текстових документів із використанням локальних AI-технологій.

У роботі було обґрунтовано актуальність використання систем автоматичного аналізу тексту та створення коротких викладів документів. Визначено основні проблеми ручної обробки великих обсягів текстової інформації та необхідність автоматизації даного процесу.

Під час виконання роботи було проведено аналіз існуючих рішень у сфері автоматичного опрацювання тексту та AI-систем для роботи із документами. На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до програмного забезпечення.

Для моделювання системи були використані UML-діаграми, зокрема діаграма прецедентів, діаграма послідовності, діаграма активності, діаграма класів та ER-діаграма бази даних. Це дозволило формалізувати структуру програмного забезпечення, визначити взаємодію між окремими компонентами системи та спростити процес подальшої реалізації програмного продукту.

У рамках роботи було спроєктовано структуру бази даних та реалізовано її за допомогою SQLite. База даних забезпечує збереження інформації про документи, результати аналізу тексту та історію роботи системи.

Для реалізації програмного забезпечення було використано мову програмування Python. Графічний інтерфейс користувача реалізовано за допомогою бібліотеки Tkinter. Для AI-аналізу тексту використовується локальний сервіс Ollama, який дозволяє виконувати автоматичне створення короткого викладу документів без використання зовнішніх хмарних сервісів.

У програмному забезпеченні реалізовано основні функціональні можливості:

- завантаження текстових документів;
- автоматичний аналіз тексту;
- формування короткого викладу документа;

- підтримка AI-режиму через Ollama;
- експорт результатів у формати TXT, DOCX та PDF;
- збереження історії обробки документів.

Також було проведено тестування програмного забезпечення, у результаті якого підтверджено коректну роботу основних функцій системи. Програмне забезпечення стабільно виконує аналіз текстових документів, підтримує локальну AI-обробку та забезпечує збереження результатів роботи.

Результатом виконаної роботи є функціональний програмний застосунок STYSLO AI для автоматичного створення короткого викладу текстових документів. Розроблена система дозволяє спростити процес опрацювання текстової інформації, підвищити швидкість аналізу документів та забезпечити зручну роботу користувача із текстовими файлами.

Таким чином, поставлена мета роботи була досягнута, а отримані результати можуть бути використані для подальшого розвитку систем автоматичного аналізу тексту та інтеграції сучасних AI-технологій у програмне забезпечення для роботи із документами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Documentation – [Електронний ресурс] – Режим доступу: https://docs.python.org/3/
2. SQLite Documentation – [Електронний ресурс] – Режим доступу: https://www.sqlite.org/docs.html
3. Ollama Documentation – [Електронний ресурс] – Режим доступу: https://ollama.com/
4. Tkinter Documentation – [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/tkinter.html
5. Jurafsky D., Martin J.H. Speech and Language Processing. - Pearson, 2021.
6. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. - Pearson, 2016.
7. Larman C. Applying UML and Patterns. - Prentice Hall, 2004.
8. Fowler M. Patterns of Enterprise Application Architecture. - Addison-Wesley, 2002.
9. Date C.J. An Introduction to Database Systems. - Pearson, 2003.
10. Ambler S. The Object Primer: Agile Model-Driven Development with UML 2. - Cambridge University Press, 2004.
11. Draw.io Documentation – [Електронний ресурс] – Режим доступу: https://www.diagrams.net/
12. Real Python - Working with SQLite in Python – [Електронний ресурс] – Режим доступу: https://realpython.com/python-sqlite-sqlalchemy/
13. GeeksforGeeks - Text Summarization using NLP – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/text-summarization-in-nlp/
14. Natural Language Processing with Python. - O’Reilly Media, 2009.
15. W3Schools Python Tutorial – [Електронний ресурс] – Режим доступу: https://www.w3schools.com/python/
16. Visual Studio Code Documentation – [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/docs

17. PyCharm Documentation – [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/pycharm/documentation/
18. MDN Web Docs – HTTP Overview – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP
19. Stack Overflow – Python Questions and Answers – [Електронний ресурс] – Режим доступу: https://stackoverflow.com/
20. OpenAI - Natural Language Processing Overview – [Електронний ресурс] – Режим доступу: https://openai.com/

ДОДАТОК А

Повний Код Додатку Styslo AI

```
import tkinter as tk
from tkinter import messagebox
import webbrowser
from datetime import datetime
from pypdf import PdfReader
from tkinter import filedialog, ttk
import sqlite3
import threading
import re
from collections import Counter
import requests
from docx import Document


from reportlab.platypus import SimpleDocTemplate, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont


from deep_translator import GoogleTranslator


# ---------- БД ----------
conn = sqlite3.connect("styslo.db", check_same_thread=False)
cursor = conn.cursor()


cursor.execute("""
CREATE TABLE IF NOT EXISTS history (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    filename TEXT,
    summary TEXT,
    mode TEXT,
    lang TEXT,
    length INTEGER
)
""")
conn.commit()


try:
    cursor.execute("ALTER TABLE history ADD COLUMN created_at TEXT")
    conn.commit()
except sqlite3.OperationalError:
    pass


last_summary = ""
current_file = ""
is_processing = False


# ---------- ЛОКАЛЬНИЙ ----------
def local_summary(text, n):
```

```
    sentences = re.split(r'[.!?\n]', text)
    sentences = [s.strip() for s in sentences if len(s) > 30]

    words = re.findall(r'\w+', text.lower())
    freq = Counter(words)

    scores = {s: sum(freq.get(w,0) for w in re.findall(r'\w+', s.lower())) for s in sentences}
    best = sorted(scores, key=scores.get, reverse=True)[:n]

    return '. '.join(best)

# ---------- AI перевірка ----------
def get_ollama_models():
    try:
        response = requests.get("http://127.0.0.1:11434/api/tags", timeout=3)
        data = response.json()

        models = []
        for model in data.get("models", []):
            models.append(model.get("name", ""))

        return models

    except:
        return None

def show_ollama_error_window(title, message, show_download=True):
    win = tk.Toplevel(root)
    win.title(title)
    win.geometry("520x300")
    win.configure(bg="#0f172a")
    win.resizable(False, False)

    tk.Label(
        win,
        text=title,
        font=("Segoe UI", 18, "bold"),
        bg="#0f172a",
        fg="white"
    ).pack(pady=(25, 10))

    tk.Label(
        win,
        text=message,
        font=("Segoe UI", 11),
        bg="#0f172a",
        fg="#cbd5e1",
        justify="center",
        wraplength=440
    ).pack(pady=10)

    btn_frame = tk.Frame(win, bg="#0f172a")
```

```
    btn_frame.pack(pady=20)

    if show_download:
        tk.Button(
            btn_frame,
            text="🌐 Відкрити сайт Ollama",
            bg="#3b82f6",
            fg="white",
            padx=14,
            pady=8,
            relief="flat",
            command=lambda: webbrowser.open("https://ollama.com/download")
        ).pack(side="left", padx=8)

    tk.Button(
        btn_frame,
        text="Закрити",
        bg="#ef4444",
        fg="white",
        padx=14,
        pady=8,
        relief="flat",
        command=win.destroy
    ).pack(side="left", padx=8)
def check_ollama_requirements():
    models = get_ollama_models()

    if models is None:
        show_ollama_error_window(
            "Ollama не запущена",
            "Для AI-режиму потрібно встановити та запустити Ollama.\n\n"
            "Після встановлення виконайте команди:\n"
            "ollama pull gemma:2b\n"
            "ollama pull phi3",
            show_download=True
        )
        return False

    required_models = ["gemma:2b", "phi3"]
    missing = []

    for model in required_models:
        if not any(model in m for m in models):
            missing.append(model)

    if missing:
        show_ollama_error_window(
            "Не знайдено AI-моделі",
            "У Ollama відсутні потрібні моделі:\n\n"
            + "\n".join(missing)
            + "\n\nВстановіть їх командами:\n"
            + "\n".join([f"ollama pull {m}" for m in missing]),
            show_download=False
        )
```

```
        return False

    return True
# ---------- AI ----------
def split_text_into_chunks(text, chunk_size=10000):
    words = text.split()
    chunks = []
    current_chunk = ""

    for word in words:
        if len(current_chunk) + len(word) + 1 <= chunk_size:
            current_chunk += word + " "
        else:
            chunks.append(current_chunk.strip())
            current_chunk = word + " "

    if current_chunk.strip():
        chunks.append(current_chunk.strip())

    return chunks



def ollama_generate(prompt, model_name="gemma:2b", num_predict=280):
    response = requests.post(
        "http://127.0.0.1:11434/api/generate",
        json={
            "model": model_name,
            "prompt": prompt,
            "stream": False,
            "options": {
                "temperature": 0,
                "num_predict": num_predict
            }
        },
        timeout=600
    )

    data = response.json()
    return data.get("response", "").strip()



def ollama_summary(text, n, lang):
    try:
        text = text.strip()

        if not text:
            return "❌ Текст документа порожній або не був зчитаний."

        if lang == "en":
            model_name = "phi3"
        else:
            model_name = "gemma:2b"
```

```
# ---------- МАЛІ ДОКУМЕНТИ ----------
if len(text) < 20000:
    root.after(0, lambda: set_progress(30, "AI аналізує документ..."))

    if lang == "ua":
        prompt = f"""
        Проаналізуй увесь документ і створи узагальнене резюме українською мовою.

        Не переказуй документ посторінково.
        Не створюй окремі блоки для кожного розділу або сторінки.
        Не використовуй markdown, списки, зірочки або заголовки.
        Не копіюй речення дослівно.
        Не вигадуй інформацію.

        Сформуй один цілісний абзац, який передає:
        1. головну тему документа;
        2. основні ідеї;
        3. для чого призначена описана система;
        4. які функції або підходи згадуються;
        5. загальний висновок.

        Обсяг: приблизно {n + 3} речень.

        Документ:
        {text}
        """
    else:
        prompt = f"""
        Analyze the whole document and create a general summary ONLY in English.

        You must translate the meaning of the provided document and write the final summary ONLY in English.
        Do not use Ukrainian words.
        Do not summarize it page by page.
        Do not create separate sections for each page or chapter.
        Do not use markdown, bullet points, asterisks, or headings.
        Do not copy sentences word for word.
        Do not invent information.

        Write one coherent paragraph that includes:
        1. the main topic;
        2. the main ideas;
        3. the purpose of the described system;
        4. the mentioned functions or approaches;
        5. the general conclusion.

        Length: about {n + 3} sentences.

        Document:
        {text}
        """
```

```
            result = ollama_generate(prompt, model_name=model_name, num_predict=280)

            root.after(0, lambda: set_progress(100, "Обробка завершена"))

            if not result:
                return "❌ AI не повернув результат."

            return result

        # ---------- ВЕЛИКІ ДОКУМЕНТИ ----------
        chunks = split_text_into_chunks(text, chunk_size=10000)

        # Обмежуємо кількість частин, щоб не обробляти книгу годинами
        MAX_CHUNKS = 8

        if len(chunks) > MAX_CHUNKS:
            step = max(1, len(chunks) // MAX_CHUNKS)
            chunks = chunks[::step][:MAX_CHUNKS]

        partial_summaries = []
        total_chunks = len(chunks)

        for index, chunk in enumerate(chunks, start=1):
            progress = int((index / total_chunks) * 80)

            root.after(
                0,
                lambda i=index, total=total_chunks, p=progress:
                set_progress(p, f"AI обробляє частину {i} з {total}")
            )

            if lang == "ua":
                prompt = f"""
Стисло передай зміст цього фрагмента українською мовою.

Не копіюй текст дослівно.
Не використовуй markdown.
Не вигадуй інформацію.
Передай тільки ключові події, ідеї або факти.

Фрагмент:
{chunk}
"""
            else:
                prompt = f"""
Briefly summarize this fragment in English.

Do not copy the text word for word.
Do not use markdown.
Do not invent information.
Include only key events, ideas, or facts.
```

```
Fragment:
{chunk}
"""

            part_result = ollama_generate(prompt, model_name=model_name, num_predict=320)

            if part_result:
                partial_summaries.append(part_result)

        if not partial_summaries:
            return "❌ AI не зміг сформувати проміжні резюме."

        combined_text = "\n\n".join(partial_summaries)

        root.after(0, lambda: set_progress(90, "Формування фінального резюме..."))

        if lang == "ua":
            final_prompt = f"""
На основі резюме частин створи короткий виклад усього документа українською мовою.

Передай:
- загальну тему;
- основні ідеї;
- ключові моменти;
- підсумковий висновок.

Не використовуй markdown.
Не вигадуй інформацію.
Не пиши занадто довго.

Приблизно {n + 3} речень.

Резюме частин:
{combined_text}
"""
        else:
            final_prompt = f"""
Based on the part summaries, create a short summary of the whole document in English.

Include:
- general topic;
- main ideas;
- key points;
- final conclusion.

Do not use markdown.
Do not invent information.
Do not write too much.
```

```
About {n + 3} sentences.


Part summaries:
{combined_text}
"""


        result = ollama_generate(final_prompt, model_name=model_name, num_predict=220)


        root.after(0, lambda: set_progress(100, "Обробка завершена"))


        if not result:
            return " ❌ AI не повернув фінальний результат."


        return result


    except requests.exceptions.ConnectionError:
        return " ❌ Ollama не запущена або недоступна."


    except Exception as e:
        return f" ❌ Помилка AI: {e}"
# ---------- ПЕРЕКЛАД ----------
def translate(text, lang):
    try:
        if lang == "en":
            return GoogleTranslator(source='auto', target='en').translate(text)
        return text
    except:
        return text


# ---------- DOCX, PDF ----------
def read_docx(path):
    doc = Document(path)
    return "\n".join(p.text for p in doc.paragraphs)


def read_pdf(path, max_pages=150):
    reader = PdfReader(path)
    text = ""


    total_pages = len(reader.pages)


    if total_pages > max_pages:
        answer = messagebox.askyesno(
            "Великий PDF",
            f"PDF має {total_pages} сторінок.\n\n"
            f"Для стабільної роботи буде зчитано перші {max_pages} сторінок.\n\n"
            "Продовжити?"
        )


        if not answer:
            return ""
```

```
    pages_to_read = min(total_pages, max_pages)

    for i in range(pages_to_read):
        page_text = reader.pages[i].extract_text()
        if page_text:
            text += page_text + "\n"

    return text

# ---------- ОБРОБКА ----------

def set_progress(value, text=""):
    progress_bar["value"] = value
    progress_label.config(text=f"{text} {int(value)}%")
    root.update_idletasks()

def animate_progress(value=1):
        if is_processing and value < 95:
            set_progress(value, "Обробка документа...")

def is_large_document(text):
    chars = len(text)
    words = len(re.findall(r'\w+', text))

    return chars > 150000 or words > 20000
def get_text_stats(text):
    chars = len(text)
    words = len(re.findall(r'\w+', text))
    sentences = len([s for s in re.split(r'[.!?]+', text) if s.strip()])

    return chars, words, sentences
def process(text):
    global last_summary, is_processing

    is_processing = True
    root.after(0, lambda: animate_progress(1))



    n = int(length_var.get())
    lang = lang_var.get()
    print("LANG =", lang)
    mode = mode_var.get()
    chars, words, sentences = get_text_stats(text)

    stats_text = (
        f"Файл: {current_file}\n"
        f"Символів: {chars}\n"
        f"Слів: {words}\n"
        f"Речень: {sentences}\n\n"
    )
    root.after(0, lambda: set_progress(35, "Аналіз тексту..."))
```

```
    if mode == "ai":
        if not check_ollama_requirements():
            is_processing = False
            root.after(0, lambda: set_progress(0, "AI-режим недоступний"))
            return


        result = ollama_summary(text, n, lang)
        status = ("🤖 AI (Ollama)", "#22c55e")
    else:
        result = translate(local_summary(text, n), lang)
        status = ("🧠 Локально", "#f59e0b")


    last_summary = stats_text + result


    is_processing = False
    root.after(0, lambda: set_progress(100, "Обробка завершена"))


    cursor.execute(
        "INSERT INTO history (filename, summary, mode, lang, length, created_at) VALUES (?,?,?,?,?,?)",
(current_file, last_summary, mode, lang, n, datetime.now().strftime("%d.%m.%Y %H:%M")))
    conn.commit()
    root.after(0, lambda: set_progress(100, "Обробка завершена"))
    root.after(0, lambda: update_ui(last_summary, status))
def update_ui(result, status):
    label_status.config(text=status[0], fg=status[1])
    output.delete(1.0, tk.END)
    output.insert(tk.END, result)


# ---------- ВІДКРИТИ ----------
def open_file():
    global current_file


    path = filedialog.askopenfilename(filetypes=[("Files", "*.txt *.docx *.pdf")])
    if not path:
        return




    current_file = path.split("/")[-1]


    if path.endswith(".docx"):
        text = read_docx(path)
    elif path.endswith(".pdf"):
        text = read_pdf(path)
    else:
        text = open(path, encoding="utf-8").read()


    output.delete(1.0, tk.END)
    output.insert(tk.END, "⏳ Обробка...")
    set_progress(0, "Файл завантажено")
```

```
    threading.Thread(target=process, args=(text,), daemon=True).start()


# ---------- ІСТОРІЯ ----------
def show_history():
    win = tk.Toplevel(root)
    win.title("Історія обробки документів")
    win.geometry("850x500")
    win.configure(bg="#0f172a")

    tk.Label(
        win,
        text="Історія обробки документів",
        font=("Segoe UI", 18, "bold"),
        bg="#0f172a",
        fg="white"
    ).pack(pady=10)

    listbox = tk.Listbox(
        win,
        font=("Segoe UI", 11),
        bg="#020617",
        fg="white",
        selectbackground="#3b82f6"
    )
    listbox.pack(fill="both", expand=True, padx=15, pady=10)

    cursor.execute("""
    SELECT filename, summary, mode, lang, length, created_at
    FROM history
    ORDER BY id DESC
    """)
    data = cursor.fetchall()

    for row in data:
        filename = row[0]
        mode = "AI" if row[2] == "ai" else "Локально"
        lang = row[3].upper()
        length = row[4]
        created = row[5] if row[5] else "без дати"

        item = f"{created} | {filename} | {mode} | {lang} | {length} реч."
        listbox.insert(tk.END, item)

    def select(e):
        i = listbox.curselection()
        if not i:
            return

        row = data[i[0]]

        detail = tk.Toplevel(win)
```

```
        detail.title("Деталі запису")
        detail.geometry("800x600")
        detail.configure(bg="#0f172a")

        info = (
            f"Файл: {row[0]}\n"
            f"Дата: {row[5] if row[5] else 'без дати'}\n"
            f"Режим: {'AI (Ollama)' if row[2] == 'ai' else 'Локальний'}\n"
            f"Мова: {row[3].upper()}\n"
            f"Довжина: {row[4]} реч.\n\n"
            f"Результат:\n"
        )

        text_box = tk.Text(
            detail,
            font=("Segoe UI", 12),
            bg="#020617",
            fg="white",
            wrap="word"
        )
        text_box.pack(fill="both", expand=True, padx=15, pady=15)

        text_box.insert(tk.END, info + row[1])

    listbox.bind("<<ListboxSelect>>", select)

# ---------- ЗБЕРЕЖЕННЯ ----------
def save_txt():
    if last_summary:
        f = filedialog.asksaveasfilename(defaultextension=".txt")
        if f:
            open(f,"w",encoding="utf-8").write(last_summary)

def save_docx():
    if last_summary:
        f = filedialog.asksaveasfilename(defaultextension=".docx")
        if f:
            doc = Document()
            doc.add_paragraph(last_summary)
            doc.save(f)

def save_pdf():
    if last_summary:
        f = filedialog.asksaveasfilename(defaultextension=".pdf")

        if f:
            pdfmetrics.registerFont(
                TTFont('Arial', 'C:/Windows/Fonts/arial.ttf')
            )

            doc = SimpleDocTemplate(
                f,
```

```
            rightMargin=35,
            leftMargin=35,
            topMargin=35,
            bottomMargin=35
        )

        styles = getSampleStyleSheet()
        style = styles["Normal"]
        style.fontName = "Arial"
        style.fontSize = 11
        style.leading = 14
        style.spaceAfter = 6

        elements = []

        clean_text = last_summary.replace("**", "")

        for line in clean_text.split("\n"):
            line = line.strip()
            if line:
                elements.append(Paragraph(line, style))

        doc.build(elements)

def clear():
    output.delete(1.0, tk.END)

# ---------- UI ----------
def make_group(frame, options, var):
    buttons = []
    def select(v):
        var.set(v)
        for b,val in buttons:
            b.config(bg="#3b82f6" if val==v else "#334155")

    for text,val in options:
        b = tk.Button(frame, text=text,
                      bg="#334155", fg="white",
                      padx=12, pady=6, relief="flat",
                      command=lambda v=val:select(v))
        b.pack(side="left", padx=5)
        buttons.append((b,val))

    select(var.get())

# ---------- GUI ----------
root = tk.Tk()
root.title("STYSLO AI")
root.geometry("1000x700")
root.configure(bg="#0f172a")

tk.Label(root, text="STYSLO AI",
```

```
        font=("Segoe UI", 28, "bold"),
        bg="#0f172a", fg="white").pack(pady=10)

# ДОДАНИЙ НАПИС
tk.Label(root, text="ПЗ для автоматичного створення текстових документів",
        font=("Segoe UI", 14),
        bg="#0f172a", fg="#94a3b8").pack()

label_status = tk.Label(root, text="", font=("Segoe UI", 12), bg="#0f172a")
label_status.pack()

progress_label = tk.Label(
    root,
    text="",
    font=("Segoe UI", 10),
    bg="#0f172a",
    fg="#94a3b8"
)
progress_label.pack(pady=3)

progress_bar = ttk.Progressbar(
    root,
    orient="horizontal",
    length=500,
    mode="determinate"
)

progress_bar.pack(pady=5)

panel = tk.Frame(root, bg="#1e293b")
panel.pack(pady=15)

mode_var = tk.StringVar(value="local")
lang_var = tk.StringVar(value="ua")
length_var = tk.StringVar(value="3")

f1 = tk.Frame(panel, bg="#1e293b"); f1.pack(pady=5)
make_group(f1, [("🧠 Локально","local"),("🤖 AI","ai")], mode_var)

f2 = tk.Frame(panel, bg="#1e293b"); f2.pack(pady=5)
make_group(f2, [("UA UA","ua"),("GB EN","en")], lang_var)

f3 = tk.Frame(panel, bg="#1e293b"); f3.pack(pady=5)
make_group(f3, [("Коротке","2"),("Середнє","3"),("Довге","5")], length_var)

btn_frame = tk.Frame(root, bg="#0f172a")
btn_frame.pack(pady=10)

def btn(text, color, cmd, col):
    tk.Button(btn_frame, text=text, bg=color, fg="white",
              padx=14, pady=6, relief="flat",
```

```
            command=cmd).grid(row=0,column=col,padx=5)


btn("🗁 Відкрити файл", "#3b82f6", open_file, 0)
btn("🖫 TXT", "#10b981", save_txt, 1)
btn("🗎 DOCX", "#6366f1", save_docx, 2)
btn("🖷 PDF", "#f59e0b", save_pdf, 3)
btn("🗄 Історія", "#8b5cf6", show_history, 4)
btn("🧹 Очистити", "#ef4444", clear, 5)

output = tk.Text(root, font=("Segoe UI", 13),
                 bg="#020617", fg="white")
output.pack(expand=True, fill="both", padx=20, pady=15)

root.mainloop()
```

ДОДАТОК **Б**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Постумент Денис Андрійович здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення для автоматичного створення текстових документів» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

o [ ] інструменти генеративного ШІ не використовувалися.

o [X] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, ____ChatGPT, DeepL_____інші (вказати назву)) були використані для:

♣ [ ] перекладу іншомовних джерел;

♣ [X] стилістичного редагування та перевірки граматики;

♣ [X] допомоги у написанні/відлагодженні програмного коду;

♣ [ ] інше: _____.

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

« » _____2026 р.	_____	**Денис ПОСТУМЕНТ**