Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

(повна назва)

Кафедра програмної інженерії

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський)

Програмна система для аналізу змісту угод на основі великих мовних моделей

(тема)

Виконав:

здобувач 4 року навчання

групи ПЗПІ-21-5

Дмитро ДУЛЬСЬКИЙ

(Власне ім’я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного

забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник доцент кафедри ПІ Олексій ТУРУТА

(посада, Власне ім’я, ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_\_ Кирило СМЕЛЯКОВ

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_\_\_\_\_

(підпис)

« » 2025 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дульському Дмитру Андрійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для аналізу змісту угод на основі великих мовних моделей

Затверджена наказом по університету від 19.05.2025

2. Термін подання студентом роботи до екзаменаційної комісії 01.06.2022 \_

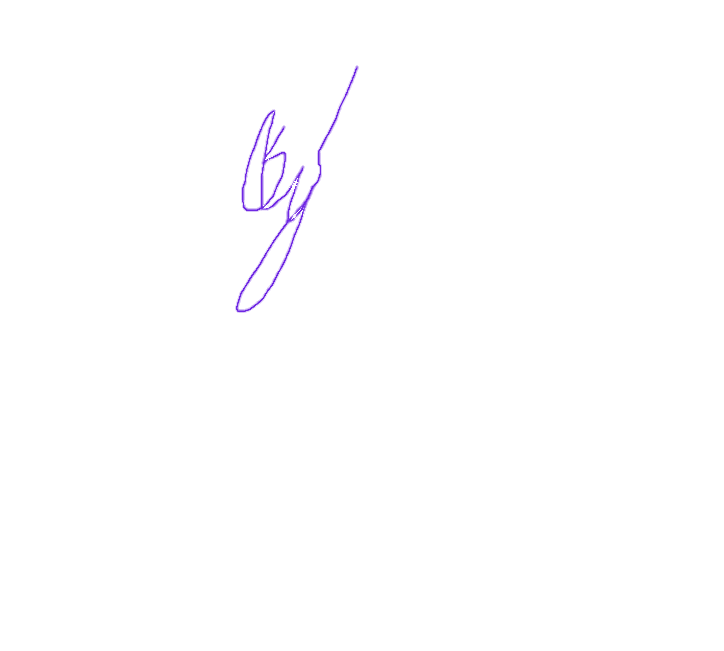
3. Вихідні дані до роботи Розробити десктопний застосунок використовуючи C# та WPF для аналізу юридичних угод із використанням великих мовних моделей. Застосунок повинен мати чат-інтерфейс, підтримку голосового введення, завантаження документів (PDF, DOCX, TXT) та управління чатами.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 13.05.2025 | *виконано* |
| 2 | Створення специфікації ПЗ | 13.05.2025 | *виконано* |
| 3 | Проектування ПЗ | 14.05.2025 | *виконано* |
| 4 | Розробка ПЗ | 21.05.2025 | *виконано* |
| 5 | Тестування ПЗ | 27.05.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 30.05.2025 | *виконано* |
| 7 | Підготовка презентації та доповіді | 31.05.2025 | *виконано* |
| 8 | Попередній захист | 05.06.2025 | *виконано* |
| 9 | Нормоконтроль, рецензування | 05.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 07.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 08.06.2025 | *виконано* |

****

Дата видачі завдання «7» «квітня» 2025р.

Здобувач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи доцент кафедри ПІ Олексій ТУРУТА

(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

**РЕФЕРАТ**

Пояснювальна записка до кваліфікаційної роботи бакалавра, 69 стор., 25 рис., 5 табл., 29 джерел.

АНАЛІЗ УГОД, ВЕЛИКА МОВНА МОДЕЛЬ, ГОЛОСОВЕ ВВЕДЕННЯ, ЗВІТИ, ПРОГРАМНА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ.

Метою роботи є розробка настільного застосунку для платформи Windows, який забезпечує інтерактивну взаємодію користувача зі штучним інтелектом у форматі діалогового вікна для аналізу змісту угод. Програма дозволяє завантажувати юридичні документи у форматах PDF, DOCX та TXT, надсилати текстові та голосові запити щодо їхнього змісту, отримувати від AI роз’яснення, виявлені ризики та ключові положення, а також зберігати історію діалогів для подальшого використання.

Клієнтська частина реалізована з використанням мови програмування C# та технології WPF (Windows Presentation Foundation) [1]. Для взаємодії зі штучним інтелектом інтегровано OpenAI Assistants API, який забезпечує контекстну обробку запитів користувача [2]. Для голосового вводу реалізовано запис аудіо з мікрофона та обробку за допомогою моделі Whisper API, що надає функцію автоматичного розпізнавання мовлення [3].

У застосунку використано такі ключові моделі даних:

* ChatMessage — повідомлення у чаті з вказанням ролі, часу та вмісту;
* MicrophoneRecorder — реалізує функціональність запису голосу з мікрофона з подальшим збереженням у форматі MP3;
* RoleToAlignmentConverter та RoleToBackgroundConverter — конвертери для стилізації повідомлень у чаті відповідно до ролі;
* AudioTranscriptionService — модуль для взаємодії з API розпізнавання мовлення;
* DocumentHandler — обробляє вміст PDF, DOCX та TXT-файлів та передає його у запит до AI.

У розробці використано такі бібліотеки та технології:

* OpenAI .NET SDK — для інтеграції з OpenAI Assistants та Whisper API;
* NAudio — для запису аудіо з мікрофона у форматі WAV;
* Microsoft.Xaml.Behaviors.Wpf — для спрощення обробки подій у XAML;
* DocumentFormat.OpenXml та PdfSharp — для читання DOCX та PDF документів відповідно.

Інтерфейс користувача побудовано у стилі сучасного чат-додатку: повідомлення користувача та асистента відображаються в окремих блоках з відповідним форматуванням, передбачено індикатор "асистент думає...", реалізовано перетягування файлів (drag & drop) для зручного завантаження, а також індикацію процесу транскрипції голосу.

У результаті виконаної роботи:

* реалізовано повноцінний клієнтський застосунок з функціональністю AI-асистента;
* інтегровано підтримку голосових команд та аналізу текстових документів;
* забезпечено зручний графічний інтерфейс для ефективної взаємодії користувача із системою.

**ABSTRACT**

ANALYSIS OF AGREEMENTS, LARGE LANGUAGE MODEL, VOICE INPUT, REPORTS, SOFTWARE SYSTEM, ARTIFICIAL INTELLIGENCE.

The aim of this work is to develop a desktop application for the Windows platform that enables interactive user interaction with artificial intelligence in a chat-style interface for contract content analysis. The application allows users to upload legal documents in PDF, DOCX, and TXT formats, send text and voice queries regarding their content, receive AI-generated explanations, identified risks, and key provisions, as well as save the dialogue history for future reference.

The client side is implemented using C# and the WPF (Windows Presentation Foundation) framework [1]. Integration with OpenAI Assistants API enables contextual processing of user queries using a large language model [2]. For voice input, audio recording from the microphone is implemented and processed using the Whisper API, providing automatic speech recognition functionality [3].

The application uses the following key data models:

* ChatMessage — represents a single chat message including the role (user or assistant), timestamp, and content;
* MicrophoneRecorder — provides functionality for recording audio from the microphone and saving it in MP3 format;
* RoleToAlignmentConverter and RoleToBackgroundConverter — converters for styling chat messages according to the sender’s role;
* AudioTranscriptionService — a module for using speech-to-text API;
* DocumentHandler — processes the contents of PDF, DOCX, and TXT files and submits them as input to the AI.

The following libraries and technologies are used in the development:

* OpenAI .NET SDK — for integration with OpenAI Assistants and Whisper APIs;
* NAudio — for recording audio from the microphone in WAV format;
* Microsoft.Xaml.Behaviors.Wpf — to simplify event handling in XAML;
* DocumentFormat.OpenXml and PdfSharp — for reading DOCX and PDF documents, respectively.

The user interface is designed in the style of a modern chat application: messages from the user and assistant are displayed in distinct blocks with appropriate formatting. A "thinking" indicator shows when the assistant is generating a response. The application also supports drag-and-drop file upload for convenience, and visual feedback for the transcription process.

As a result of the completed work:

* a fully functional desktop client application with AI assistant capabilities was developed;
* support for voice commands and text document analysis was integrated;
* a user-friendly graphical interface was created to enable efficient interaction with the system.

**ЗМІСТ**

[Перелік скорочень 9](#_Toc200533890)

[Вступ 10](#_Toc200533891)

[1 Аналіз предметної галузі 11](#_Toc200533892)

[1.1 Огляд аналогів 11](#_Toc200533893)

[1.2 Виявлення та вирішення проблем 14](#_Toc200533894)

[1.3 Постановка задачі 18](#_Toc200533895)

[2 Формування вимог до програмної системи 20](#_Toc200533896)

[2.1 Функціональні вимоги 20](#_Toc200533897)

[2.2 Нефункціональні вимоги 20](#_Toc200533898)

[3 Архітектура та проєктування програмного забезпечення 22](#_Toc200533899)

[3.1 UML проєктування ПЗ 22](#_Toc200533900)

[3.2 Проєктування архітектури ПЗ 23](#_Toc200533901)

[3.3 Проєктування структури зберігання даних 24](#_Toc200533902)

[3.4 Приклади найцікавіших алгоритмів та методів 25](#_Toc200533903)

[3.5 Створення UI / UX 26](#_Toc200533904)

[4 Опис прийнятих програмних рішень 30](#_Toc200533905)

[4.1 Опис підключення OpenAI API 30](#_Toc200533906)

[4.2 Обробка та вивід введеної інформації 31](#_Toc200533907)

[4.3 Збереження історії чатів з асистентом 35](#_Toc200533908)

[5 Тестування розробленого програмного забезпечення 41](#_Toc200533909)

[5.1 Тестування back-end частини застосунку 41](#_Toc200533910)

[5.2 Тестування UI/UX застосунку 47](#_Toc200533911)

[Висновки 53](#_Toc200533912)

[Перелік джерел посилання 54](#_Toc200533913)

[Додаток А 57](#_Toc200533914)

[Додаток Б 58](#_Toc200533915)

[Додаток В 67](#_Toc200533916)

**ПЕРЕЛІК СКОРОЧЕНЬ**

LLM – Large Language Model (велика мовна модель)

WPF – Windows Presentation Foundation

API – Application Programming Interface

PDF – Portable Document Format

DOCX – Office Open XML Document

TXT – Text File

MP3 – MPEG Audio Layer III

WAV – Waveform Audio File Format

UI – User Interface

UX – User Experience

UML – Unified Modeling Language

ПЗ – Програмне Забезпечення

XML – Extensible Markup Language

SDK – Software Development Kit

ID – Identifier

GPT – Generative Pre-trained Transformer

# Вступ

У сучасних умовах стрімкого розвитку інформаційних технологій та зростаючого обсягу цифрових даних постає потреба у нових підходах до їх аналізу, обробки та інтерпретації. Особливо актуальним це стає в освітній, науковій, бізнесовій та адміністративній сферах, де користувачі щоденно взаємодіють з великими масивами текстової інформації. Значну роль у цьому процесі починають відігравати системи на основі штучного інтелекту, зокрема великі мовні моделі (LLM), здатні генерувати, узагальнювати та аналізувати текст у зручній для людини формі.

Разом з тим, ефективна взаємодія користувача зі штучним інтелектом потребує зручного інтерфейсу, який забезпечує як текстовий, так і голосовий ввід, підтримує роботу з документами, а також дозволяє зберігати історію діалогу.

З огляду на зазначене, постає актуальна задача створення клієнтського застосунку, що поєднує функціональність великої мовної моделі з можливістю інтерактивної роботи з документами та голосовими командами. Такий застосунок має не лише розширити доступність інтелектуальної допомоги, а й підвищити продуктивність користувача, скоротивши час на виконання рутинних завдань.

Метою цієї роботи є створення програмної системи — десктопного застосунку «AI Assistant» для Windows, який реалізує інтерактивну взаємодію користувача з AI у форматі діалогу. Основна функціональність полягає у можливості надсилати текстові та голосові запити, завантажувати документи у форматах PDF, DOCX, TXT, отримувати аналітичні або інформаційні відповіді від моделі, а також зберігати історію діалогу для подальшого перегляду.

# 1 Аналіз предметної галузі

## 1.1 Огляд аналогів

Юридичні угоди є ключовим елементом ділових, фінансових та міжособистісних взаємин. Вони регламентують права та обов’язки сторін, визначають строки, відповідальність за порушення умов, а також інші критично важливі аспекти співпраці. Однак на практиці значна частина угод містить неточні формулювання, неоднозначні положення або приховані ризики, що можуть спричинити правові спори.

Особливої актуальності набуває питання аналізу змісту договорів у великих обсягах — наприклад, при аудиті, автоматичному моніторингу чи перевірці відповідності стандартам. Традиційний підхід, що базується на ручній перевірці, є трудомістким, повільним і схильним до помилок.

Нейронні методи генерації природної мови є потужним інструментом для створення систем на основі ШІ [4]. Завдяки розвитку великих мовних моделей (Large Language Models, LLM), з’явилась можливість автоматизувати процес аналізу документів, включаючи юридичні. Сучасні LLM здатні не лише розпізнавати структуру документу, а й виявляти юридичні пастки, некоректні формулювання, суперечності, а також витягувати ключові положення, як-от умови штрафів, терміни дії, підстави для розірвання угоди тощо.

Окрім підвищення точності юридичного аналізу, ШІ-асистенти здатні формувати культуру відповідальності, прозорості та етичного використання аналітичних інструментів, зокрема у чутливих сферах, таких як протидія шахрайству чи оцінювання ризиків, спираючись на інформацію зі сторонніх джерел [5].

Під час аналізу існуючих ШІ-асистентів для юридичного аналізу контрактів я детально дослідив провідні рішення на ринку — зокрема Anthropic Claude, Lexion, Kira Systems та LawGeex. Ці інструменти суттєво відрізняються за підходами до обробки юридичних документів, технологічною основою, функціоналом та ціноутворенням.

Anthropic Claude, зокрема модель Claude 3.7 Sonnet, є одним із найбільш точних інструментів для юридичного аналізу: модель досягає F1-метрики понад 0.82 при виявленні критичних умов контрактів, таких як штрафи чи відповідальність за збитки [6]. Ядром моделі є "Constitutional AI" — підхід, який забезпечує безпечне, етичне та пояснюване поводження моделі, а також дозволяє працювати з надвеликим контекстом (до 200 000 токенів). Вартість використання Claude коливається в межах $3 за мільйон вхідних токенів та $15 за мільйон вихідних, що робить його порівняно доступним рішенням у класі високоточних LLM.

Lexion розроблений як інструмент для автоматизації юридичних і бізнес-процесів. Система аналізує угоди, знаходить терміни, зобов’язання, обмеження та автоматично додає їх у внутрішню базу для подальшої роботи. Вона інтегрується з MS Word та Google Docs, і на відміну від універсальних моделей, використовує вузькоспеціалізовані навчальні вибірки та власну нейронну архітектуру [7]. Перевагою Lexion є зручна візуалізація положень контракту в контексті, однак система в основному підходить для обробки англомовних документів.

Kira Systems вирізняється можливістю донавчання під конкретні задачі користувача — наприклад, автоматизоване витягування пунктів з угод M&A чи орендних договорів. В основі системи лежить класична модель машинного навчання з глибокими векторними представленнями, натренована на мільйонах контрактних прикладів. Такий підхід забезпечує високу точність і гнучкість, проте вимагає участі аналітика на етапі налаштування [8].

LawGeex автоматизує перевірку угод шляхом порівняння з шаблонами політик компанії, виявляючи порушення або нетипові формулювання. Він є більш закритою системою, з передналаштованою логікою і без можливості навчання на власних кейсах, але підходить для підприємств, які хочуть мінімізувати юридичні ризики без участі юриста в кожному перегляді.

Порівнюючи ці рішення, можна зробити висновок, що Anthropic Claude має перевагу в гнучкості та універсальності, працює із довгими контрактами й видає якісні пояснення [6]. Lexion — це приклад бізнес-орієнтованої автоматизації з простим інтерфейсом і швидким запуском, а Kira Systems — потужний інструмент для юридичних фірм, де потрібна кастомізація. Натомість LawGeex підійде для компаній зі стандартизованими процесами та потребою в консервативному контролі документів. У цьому контексті програмна система для аналізу змісту угод може суттєво підвищити ефективність юридичного аналізу, автоматизувати попередню експертизу договорів, зменшити людський фактор та пришвидшити ухвалення рішень. Особливо вона буде корисною для юристів, контракт-менеджерів, підприємців і аудиторських служб, яким важливо швидко отримувати юридично релевантну інформацію з великої кількості документів.

Один із найяскравіших прикладів впровадження штучного інтелекту в аналіз юридичних документів демонструє компанія Swiss Prime Site (SPS). У 2023 році, у зв'язку з закриттям універмагу Jelmoli в Цюриху, SPS зіткнулася з необхідністю переглянути понад 1000 чинних контрактів. Замість традиційного підходу із залученням великої юридичної фірми, компанія обрала використання ШІ-рішення від Legartis. Це дозволило завершити проект за два місяці, зекономивши близько 90% витрат порівняно з традиційними методами, без шкоди для якості. ШІ-система автоматично витягувала ключові дані з контрактів, які потім використовувалися для створення листів щодо змін у договорах, забезпечуючи точність і своєчасність повідомлень для всіх сторін [9].

Іншим прикладом є досвід компанії Mondelēz International, яка під час відокремлення свого бізнесу зіткнулася з необхідністю перегляду 40 000 контрактів. Співпрацюючи з компанією Axiom, Mondelēz автоматизувала процес аналізу контрактів, що дозволило завершити збір та оцінку документів менш ніж за 35 днів. Цей підхід забезпечив значну економію коштів і часу порівняно з традиційними юридичними фірмами, дозволяючи компанії дотриматися встановлених термінів і бюджету [10].

Компанія Dell також впровадила автоматизований аналіз контрактів, співпрацюючи з Axiom для обробки 35 000 контрактів на продаж. Це дозволило швидко отримувати доступ до ключових положень контрактів, покращуючи процес прийняття рішень керівництвом і забезпечуючи відповідність зобов'язанням [11].

У сфері юридичних послуг компанія Deloitte впровадила програмне забезпечення Kira Systems для аналізу контрактів, що дозволило скоротити час на перегляд складних документів на 20–90%. Це рішення використовує машинне навчання для швидкого "читання" тисяч документів, витягуючи та структуруючи текстову інформацію для кращого аналізу [12].

Ці приклади демонструють, як впровадження ШІ в аналіз юридичних документів дозволяє компаніям значно підвищити ефективність, зменшити витрати та забезпечити точність у роботі з контрактами.

1.2 Виявлення та вирішення проблем

У сучасних умовах зростаючого обсягу юридичної, фінансової та організаційної документації, традиційні методи ручного аналізу угод демонструють істотне зниження ефективності. Зростання кількості контрактів та складності їх умов вимагають від фахівців все більше часу на опрацювання, що призводить до значних затримок і підвищує ризик пропуску важливих деталей. Через обмеження людських ресурсів та людський фактор помилки стають неминучими, що часто тягне за собою серйозні фінансові та репутаційні втрати для компаній.

Яскравим прикладом наслідків неефективного управління контрактами стала криза британської компанії Carillion. Цей гігант будівельної галузі мав портфель із понад 450 контрактів, серед яких були численні довгострокові та складні угоди. Відсутність систематичного підходу до аналізу ризиків і відсутність автоматизованих інструментів для контролю призвели до накопичення боргів у розмірі 1,5 мільярда фунтів стерлінгів, а також до дефіциту пенсійного фонду на 580 мільйонів. Компанія збанкрутувала у 2018 році, що призвело до втрати роботи понад 3 тисяч співробітників і завдало шкоди понад 30 тисячам постачальників. Важливо зауважити, що у цьому випадку великі збитки були спричинені не лише економічними факторами, але й низькою якістю аналізу та контролю контрактних зобов’язань, що могло бути значно покращено за допомогою систем на основі штучного інтелекту, здатних виявляти ризики та невідповідності в документах [13].

Ще одним важливим прикладом є інцидент у сфері обслуговування житла з компанією Mears LLP. Протягом двох фінансових років працівники компанії систематично змінювали дати виконання ремонтних робіт, створюючи ілюзію своєчасного виконання контрактних зобов’язань. Загалом було виявлено понад 13 тисяч випадків фальсифікації, що склало близько 8% усіх неекстрених ремонтів. Хоча це не принесло безпосередньої фінансової вигоди через бонуси, інцидент підірвав довіру до компанії з боку клієнтів і призвів до затримки укладання нового контракту на суму 1,8 мільярда фунтів стерлінгів. Цей випадок демонструє, наскільки важливо забезпечити контроль якості даних у роботі з договорами, і як автоматизовані системи можуть допомогти у виявленні подібних порушень завдяки аналізу великих обсягів інформації та пошуку аномалій у даних [14].

Загалом, ці приклади чітко показують, що ручний аналіз документів у великих обсягах не лише надто повільний, але й часто неточний. Усе це призводить до підвищеного ризику пропуску критичних моментів у договорах, що тягне за собою значні збитки, затримки в роботі та втрату довіри партнерів і клієнтів. Впровадження штучного інтелекту та асистентів на його основі у процеси аналізу юридичних та фінансових угод могло б суттєво знизити ризики, які призвели до катастрофічних наслідків, подібних до випадків Carillion, будівельних проєктів із помилками у контрактах та ситуації з Mears LLP. Розглянемо, як саме це могло би вплинути на результати й ефективність роботи компаній.

По-перше, автоматизовані системи на базі ШІ здатні обробляти тисячі сторінок документів за лічені хвилини, що в десятки разів швидше за ручний аналіз, де навіть досвідчені юристи витрачають тижні на перевірку складних угод. Це означає, що потенційно понад 90% часу, який раніше витрачався на рутинний аналіз, можна було б звільнити для фокусування на критично важливих аспектах, таких як стратегічне управління ризиками або переговори з контрагентами. Наприклад, у випадку Carillion це могло б скоротити час виявлення проблемних контрактів із місяців до кількох днів, дозволяючи вчасно приймати заходи для мінімізації боргових зобов’язань.

По-друге, ШІ асистенти мають здатність виявляти аномалії, невідповідності та потенційні ризики, які часто залишаються непоміченими при ручному аналізі через людський фактор або перевантаження. Згідно з дослідженнями, впровадження подібних технологій може знизити кількість помилок у документації на 70–80%, що безпосередньо впливає на зменшення фінансових втрат і покращення якості контрактів. У разі будівельної компанії, про яку йшлося раніше, це могло б запобігти включенню невідповідних умов, таких як недооцінка особливостей ґрунту, що могло б зберегти мільйони доларів і уникнути затримок.

Також автоматизація контролю даних і термінів виконання робіт за допомогою ШІ дозволила б уникнути випадків фальсифікації та маніпуляцій із інформацією, подібних до тих, що сталися з Mears LLP. Системи на основі штучного інтелекту можуть аналізувати історичні дані, відстежувати зміни у документах і сигналізувати про підозрілі або незвичайні дії. Це зменшує ймовірність шахрайства та покращує прозорість процесів, що, у свою чергу, позитивно впливає на репутацію компанії і довіру партнерів.

Загалом, можна припустити, що впровадження сучасних ШІ-асистентів у сферу аналізу юридичних документів здатне підвищити загальну ефективність управління контрактами мінімум на 50-70%, одночасно знижуючи ризики фінансових втрат і юридичних проблем.

У сучасному цифровому середовищі голосове введення стає все популярнішим способом взаємодії з пристроями та програмами, зокрема в мобільних застосунках, віртуальних асистентах і десктопних системах. За даними Statista, ще у 2023 році понад 50% інтернет-користувачів у США хоча б раз на місяць користувалися голосовим введенням, а в країнах з розвиненою мобільною інфраструктурою цей показник може сягати 60–70% [15]. Поширення смартфонів, розумних колонок (на кшталт Google Nest чи Amazon Echo), а також поява високоточної технології розпізнавання мовлення суттєво сприяли цій тенденції.

Одним із найважливіших досягнень у цій сфері є розвиток систем розпізнавання мовлення, що базуються на великих мовних моделях (LLM) та штучному інтелекті. Наприклад, OpenAI Whisper — це нейромережева система розпізнавання мовлення, яка навчена на багатомовному корпусі з тисяч годин аудіо. Завдяки цьому вона здатна не лише точно трансформувати мовлення в текст, а й ігнорувати дефекти дикції, акценти, фонетичні викривлення, роблячи технологію доступною навіть для користувачів із вадами мовлення [16].

Це відкриває важливі можливості для інклюзивності: користувачі, яким складно вводити текст вручну або які мають порушення опорно-рухового апарату чи проблеми з читанням, можуть повноцінно взаємодіяти із системою завдяки голосовому введенню. Такі асистенти на основі ШІ, як Google Assistant, Apple Siri чи Microsoft Copilot, активно використовують ці технології, оптимізуючи взаємодію та пришвидшуючи діалог.

У контексті роботи з документами, зокрема юридичними або діловими угодами, голосовий інтерфейс може значно підвищити продуктивність. Наприклад, користувач може швидко сформулювати запит на зразок: «Знайди штрафні санкції в цьому контракті» або «Поясни пункт про дострокове розірвання», не відволікаючись на клавіатуру. Це не лише економить час, а й дозволяє діяти в ситуаціях, коли ручне введення є незручним — наприклад, під час пересування або в офісі під час обговорення з колегами.

Згідно з дослідженням Capgemini, понад 74% користувачів, які застосовують голосові асистенти в повсякденному житті, відзначають, що це економить їм щонайменше 20% часу у порівнянні з ручним введенням [17]. Таким чином, інтеграція розпізнавання мовлення в інтелектуальні системи аналізу документів — не лише зручність, а й суттєвий приріст ефективності.

На основі аналізу предметної області було виявлено такі ключові проблеми, які може вирішити розроблювана програмна система:

* відсутність інструменту для інтелектуального аналізу тексту угоди. Типові редактори не надають змоги ставити запитання до змісту документа та отримувати контекстуально точні відповіді;
* складність орієнтування у великому обсязі даних. Угоди можуть містити десятки сторінок, тому користувачам складно швидко знайти потрібну інформацію (наприклад, умови оплати, терміни, зобов’язання сторін);
* мовний бар’єр або складність юридичної термінології. Багато угод написано офіційно-діловою мовою, яка малозрозуміла широкому колу користувачів;
* відсутність голосового введення запитів. У ситуаціях обмеженого доступу до клавіатури або в мобільних умовах користувачу зручніше задавати питання голосом;
* обмежені можливості аналізу документів в автономному режимі. Існуючі рішення часто потребують постійного підключення до інтернету або не забезпечують інтеграції з мовними моделями.

З метою подолання зазначених недоліків пропонується створити програмну систему, яка дозволяє:

* завантажувати документи у форматі PDF або DOCX;
* автоматично витягувати текст із документів;
* задавати запитання до змісту угоди у текстовій або голосовій формі;
* отримувати точні відповіді від мовної моделі GPT-4, що аналізує зміст документа;
* використовувати розпізнавання мовлення (модель Whisper-1) для зручності голосового введення;
* забезпечити підтримку української, англійської та інших мов інтерфейсу.

Проєкт спрямований на підвищення доступності юридичної інформації, автоматизацію аналізу угод та інтеграцію інструментів штучного інтелекту в повсякденну роботу користувача.

1.3 Постановка задачі

Під час виконання роботи необхідно розробити десктопний застосунок для операційної системи Windows, який забезпечить користувачеві можливість:

* завантажувати текстові документи (угоди, договори тощо);
* ставити запитання щодо їх змісту;
* отримувати інтелектуальні відповіді на основі великої мовної моделі GPT‑4;
* використовувати голосовий ввід запитів за допомогою моделі Whisper-1.

Основні завдання системи:

* інтерфейс користувача на базі WPF з сучасним дизайном і підтримкою української мови;
* завантаження документів у форматах PDF, DOCX або TXT;
* обробка запиту користувача до змісту документа через інтеграцію з GPT‑4;
* виведення відповідей у зручному форматі з можливістю копіювання або збереження результатів;
* голосове введення запитів (Speech-to-Text) за допомогою Whisper-1;
* журнал сеансів, який дозволяє переглядати історію діалогів із системою;
* обробка даних локально або із збереженням приватності переданого вмісту.

В результаті реалізації проєкту користувач отримає зручний засіб для попереднього юридичного аналізу договорів, що дозволить скоротити час на ознайомлення зі змістом документа, швидко виявити суттєві умови або суперечливі пункти, а також ефективно взаємодіяти з системою за допомогою текстових та голосових запитів. Такі засоби як введення голосом, збереження історії спілкування та доступ до неї у будь-який час за наявності мережі Інтернет дозволить зберігати всі розмови, а можливість додавання декількох файлів надасть змогу порівнювати їх між собою

# 2 Формування вимог до програмної системи

## 2.1 Функціональні вимоги

У рамках реалізації програмної системи для аналізу змісту договорів на основі великих мовних моделей були визначені такі функціональні можливості, які забезпечують користувачу повний цикл роботи з документом:

Основний користувач – аналітик або юрист:

* завантаження документа для аналізу у форматах .docx, .pdf або .txt;
* формулювання запиту до вмісту документа у текстовому полі;
* отримання відповіді від GPT-4 з поясненням умов договору, виявленням ризиків, юридичних термінів тощо;
* голосове введення запиту (Speech-to-Text) за допомогою моделі OpenAI Whisper-1;
* перегляд історії сеансів спілкування: питання – відповідь, із фіксацією часу та назви документа;
* збереження результатів аналізу у файл (наприклад, .txt або .pdf) для подальшого використання;
* підтримка української мови в інтерфейсі та обробці запитів.

Адміністратор/розробник:

* налаштування API-ключів для GPT-4 та Whisper.

2.2 Нефункціональні вимоги

Під час роботи було визначено наступні нефункціональні вимоги:

Технології та архітектура:

* мова програмування: C#;
* технологія інтерфейсу: Windows Presentation Foundation (WPF);
* архітектура: Singleton;
* інтеграція з GPT-4: через OpenAI API;
* інтеграція з Whisper-1: або через локальну модель (за умови підтримки), або також через OpenAI API.

Зовнішні бібліотеки:

* + OpenAI API SDK – взаємодія з мовними моделями;
  + System.Speech, NAudio або сторонній binding для обробки аудіофайлів;
  + Microsoft.Office.Interop.Word або DocX, PdfSharp, iText7 – для обробки різних форматів документів.

Інтерфейс:

* дружній та інтуїтивно зрозумілий інтерфейс з підтримкою кнопок, вкладок, вікон діалогу;
* можливість надавати відповіді на всіх мовах світу, в тому числі українською [18];
* підтримка клавіш швидкого доступу для зручності користування (Ctrl+O – відкрити документ, Ctrl+S – зберегти відповідь);
* темна/світла тема (за бажанням можна реалізувати як майбутнє розширення).

Надійність і продуктивність:

* обробка помилок API: повідомлення про втрату інтернет-з’єднання, неправильний запит або відсутній ключ;
* збереження стану сесії: можливість продовжити роботу з того ж документа після перезапуску програми;
* вимоги до продуктивності: час очікування відповіді асистента не перевищує 10 секунд у типових умовах.

Безпека:

* локальна обробка документів без відправлення повного вмісту документа в API (опціонально);
* шифрування API-ключів у конфігураційному файлі;
* обмеження доступу до журналу сесій чи логів – тільки для адміністратора.

# 3 Архітектура та проєктування програмного забезпечення

3.1 UML проєктування ПЗ

За допомогою діаграми прецедентів було визначено основні функції, доступні користувачу у застосунку. Створену діаграму зображено на рисунку 3.1.

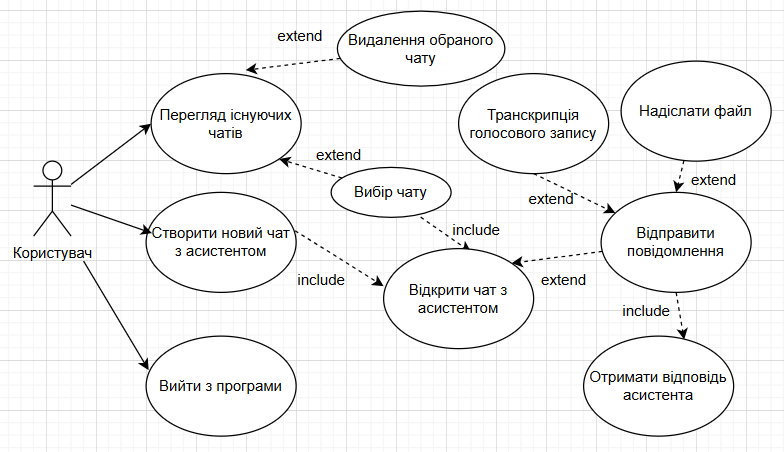


Рисунок 3.1 – Діаграма прецедентів застосунку з аналізу документів (рисунок виконано самостійно)

У застосунку користувач може переглядати наявні чати з асистентами, створювати нові чати або виходити з програми. Перегляд наявних чатів передбачає можливість видалення обраного чату або відкриття потрібного чату для подальшої взаємодії з асистентом. Також відкриття чату можливе одразу після створення нового.

У чаті з асистентом користувач може надіслати повідомлення та отримати відповідь. Перед надсиланням повідомлення можна прикріпити файл або продиктувати текст — тоді він буде автоматично перетворений із голосової форми на текстову.

3.2 Проєктування архітектури ПЗ

Для реалізації програмної системи було обрано синглтон-архітектуру як базову архітектурну модель. На відміну від більш складних підходів, таких як багаторівнева або компонентна архітектура, синглтон-підхід є доцільним для застосунків, які мають лінійний сценарій роботи, обмежену кількість основних функціональних блоків та не потребують масштабування або багатокористувацької взаємодії.

У даному застосунку більшість класів існують у єдиному екземплярі та взаємодіють один з одним напряму, без жорсткої ізоляції по модулях або компонентах. Такий підхід спрощує реалізацію та прискорює розробку, що є критичним у рамках дипломного проєкту. Основний потік користувацького досвіду базується на методах асинхронного програмування [19, c. 145].

Під час створення архітектури чатбота важливо враховувати потребу в маршрутизації запитів користувача до окремих агентів залежно від контексту, спираючись на інформацію зі сторонніх джерел [20].

Ключовим прикладом використання патерна Singleton є клас ThreadStorage, який забезпечує централізоване зберігання та доступ до історії чатів. Цей клас реалізує єдину точку істини для списку створених чатів, що гарантує узгодженість даних протягом усього життєвого циклу застосунку. Хоча він не є синглтоном у класичному обʼєктно-орієнтованому сенсі (без приватного конструктора та статичного екземпляра), по суті, він поводиться як синглтон завдяки повністю статичним членам класу.

Використання такого архітектурного підходу забезпечило:

* простоту взаємодії між компонентами (наприклад, MainWindow, ThreadStorage, assistantClient);
* мінімізацію звʼязаності та залежностей;
* централізований контроль над збереженням та видаленням чатів;
* простіший механізм тестування окремих функцій.

Таким чином, обрана архітектура синглтонів є логічною та виправданою у контексті даного застосунку, з урахуванням обсягів, функціоналу та обмежених часових рамок реалізації.

3.3 Проєктування структури зберігання даних

У розробленій програмній системі було реалізовано просту, але ефективну модель зберігання інформації про активні та історичні сесії спілкування з великою мовною моделлю. Оскільки система працює у вигляді десктопного застосунку та не використовує повноцінну базу даних, було обрано текстовий файл як локальне сховище метаданих чатів. Такий підхід дозволяє забезпечити легкість у реалізації, прозорість зберігання та швидке читання/оновлення інформації.

Для зберігання інформації про кожну створену сесію використовується файл threads.txt, в якому кожен рядок містить метадані чату у вигляді трьох полів: ідентифікатор асистента, ідентифікатор потоку (thread) та ім’я чату. Ці значення розділяються комами. Така структура дозволяє швидко зчитувати всі доступні чати при старті програми або при оновленні списку без необхідності обробки складних структур.

Коли користувач створює новий чат або завершує поточний, відповідна інформація додається до файлу, що дозволяє відновити історію чатів при наступному запуску. Видалення чату також оновлює файл, видаляючи відповідний рядок.

Важливо зазначити, що ця локальна структура слугує лише для збереження ідентифікаторів та назв чатів. Самі ж повідомлення, історія діалогу та контекстні дані зберігаються на стороні серверів OpenAI через API. Тобто, при виборі певного чату з локального списку, застосунок звертається до віддаленого серверу OpenAI за вказаним ідентифікатором потоку (thread ID) та асистента (assistant ID) і динамічно отримує повний вміст діалогу у режимі реального часу. Це дозволяє уникнути дублювання даних на клієнтській стороні та гарантує синхронність історії діалогів з офіційним сервісом.

Таким чином, архітектура зберігання даних у цій системі побудована на принципі розділення відповідальностей: локальна частина відповідає за мінімальний набір даних, необхідний для навігації між чатами, тоді як повноцінний вміст зберігається та обробляється хмарним API.

3.4 Приклади найцікавіших алгоритмів та методів

Алгоритм реалізації голосового введення в застосунку побудований навколо взаємодії користувача з мікрофоном, обробкою аудіо та його перетворенням у текст. Коли користувач натискає кнопку запису, ініціалізується об’єкт MicrophoneRecorder, який починає захоплення звуку з мікрофону. Ще до старту запису формується унікальне ім’я для тимчасового WAV-файлу, щоб уникнути конфліктів із попередніми сесіями. Основи обробки винятків реалізовано відповідно до порад, описаних у [21, c. 278]. Запис триває до моменту, коли користувач повторно натискає кнопку — тоді запис зупиняється, і збережений аудіопотік зберігається у WAV-файл.

Далі цей WAV-файл конвертується у формат MP3 за допомогою нативної бібліотеки libmp3lame [22]. Результуючий MP3-файл передається до API Whisper від OpenAI, де відбувається розпізнавання мовлення. Після успішного розпізнавання отриманий текст одразу вставляється у текстове поле введення повідомлення.

Завершальним етапом є очищення — тимчасові файли WAV і MP3 видаляються, щоб не займати місце на диску. Після цього користувач може знову почати запис, натиснути «Надіслати» текст або вийти з програми. Алгоритм підтримує багаторазове повторення цього процесу без перезапуску застосунку, що робить інтерфейс зручним і гнучким.

Схематичне зображення алгоритму відображене на рисунку 3.2.

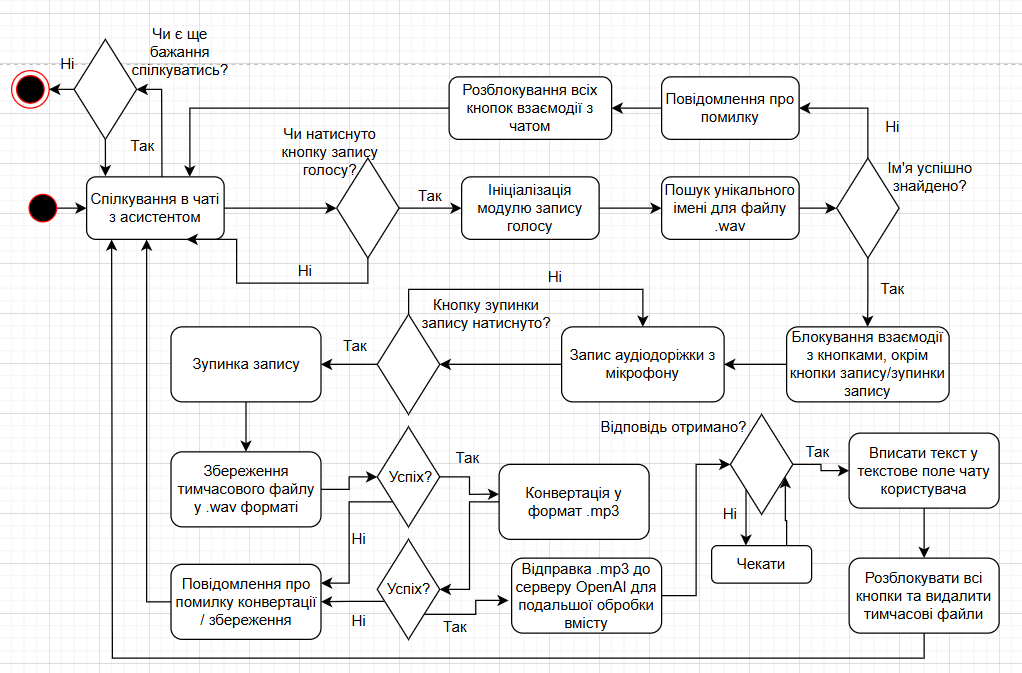


Рисунок 3.2 – Діаграма діяльності запису голосового повідомлення та конвертації його в текст (рисунок виконано самостійно)

Таким чином, за допомогою описаного алгоритму виконується робота моделі whisper-1 з конвертації голосу у текстове повідомлення у розроблюваному програмному забезпеченні.

3.5 Створення UI / UX

Під час розробки дизайну спочатку було створено базову структуру вікна чату. Реалізація візуальних компонентів інтерфейсу була виконана з використанням принципів Material Design [23]. Для цього використовувалася ручка та папір (див. рис. 3.3).

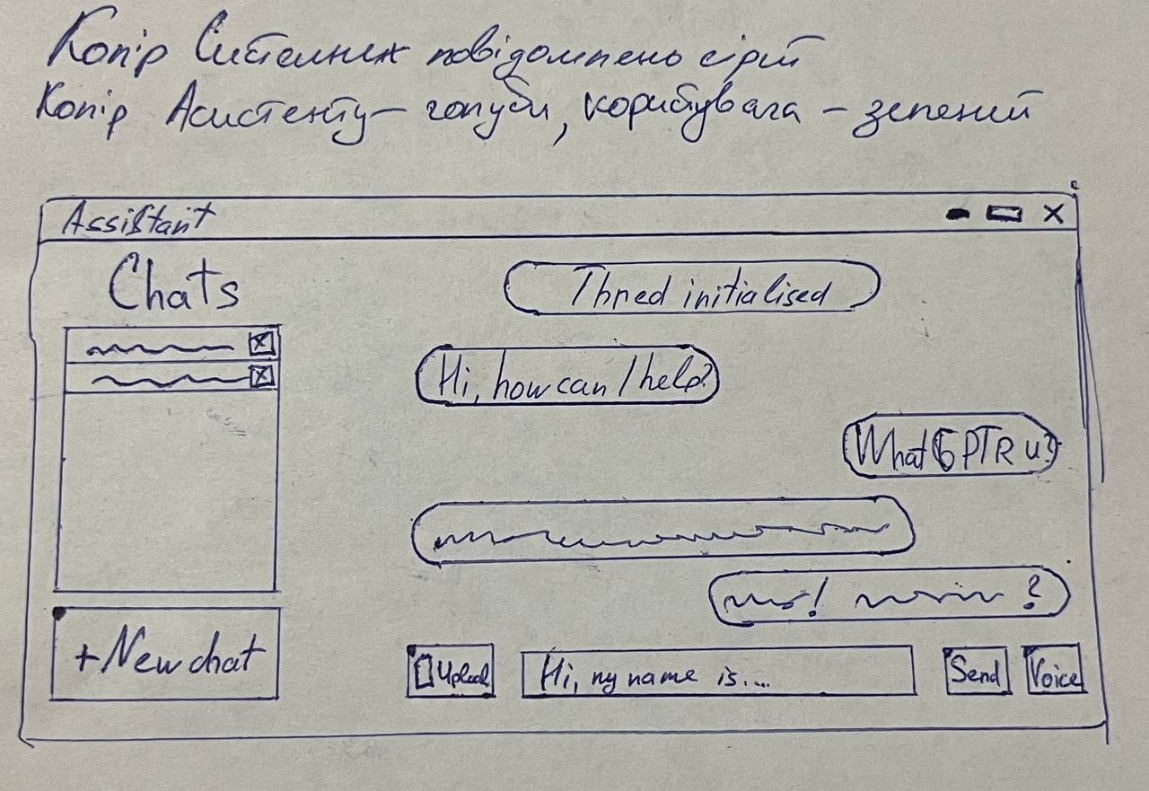


Рисунок 3.3 – Прототип UI застосунку (рисунок виконано самостійно)

Прототип було поділено на кілька ключових зон. Зліва розташована область, що є бічною панеллю чатів, де передбачено місце для відображення списку розмов. На цій панелі присутні назви чатів та кнопки видалення чатів. Більшу центральну частину відведено під основну область чату, де схематично зображено кілька бульбашок з текстом, що імітують обмін повідомленнями між користувачем та асистентом. У нижній частині прототипу розміщено панель введення повідомлень, яка включає поле для набору тексту, а також кнопки "Send" та "Voice", що передбачають можливість відправки як текстових, так і голосових повідомлень. Крім того, на цій панелі наявна кнопка "Upload", яка вказує на функцію завантаження файлів. У нижній лівій частині ескізу розміщено кнопку "New Chat" для ініціювання нової розмови.

На ескізі присутні певні рішення щодо інтерфейсу. Розділення інтерфейсу на функціональні зони, такі як бічна панель чатів, основна область повідомлень та панель введення, є стандартним рішенням для чат-інтерфейсів, що має забезпечити зручну навігацію та використання. Прототип передбачає наявність основних функцій, необхідних для спілкування в чаті: можливість відправки текстових та голосових повідомлень, завантаження файлів та початок нової розмови. Візуалізація системних повідомлень, про що свідчить бульбашка "Thread Initialised", вказує на розуміння важливості інформування користувача про стан чату. Розміщення кнопки "New Chat" у нижній лівій частині є інтуїтивно зрозумілим для користувачів. Розташування поля для введення тексту та кнопок відправки поруч є стандартним і має забезпечити швидке введення та відправку повідомлень. Загалом, ескіз на листку відображає базове розуміння структури та ключових функціональних елементів.

Графічний інтерфейс користувача в застоснку розроблено з використанням технології WPF (Windows Presentation Foundation). Перш ніж реалізовувати інтерфейс користувача, було розглянуто ключові особливості WPF, описані у [24]. Під час розробки застосунку було враховано основні принципи структури проєктів на WPF, описані в технічній літературі [25].

Основна структура побудована навколо двох ключових зон: бокової панелі з переліком чатів та головної області переписки. Всі елементи компонуються через сіткові структури, а стилізація зосереджена на простоті, функціональності та інтуїтивній взаємодії. Приклад готового дизайну відображено на рисунку 3.4.

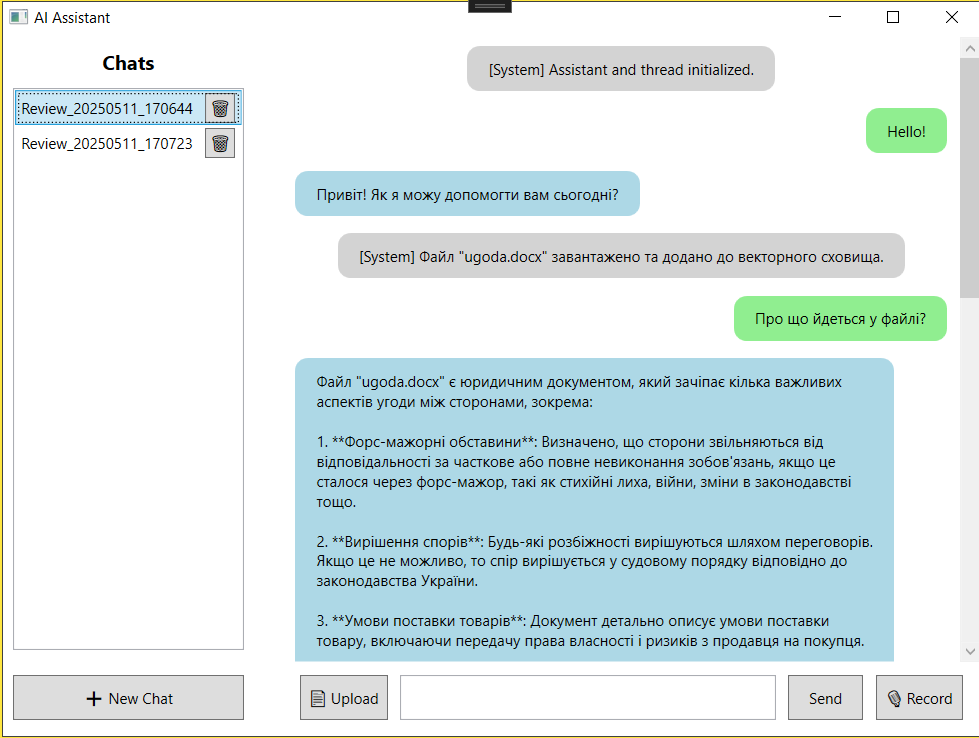


Рисунок 3.4 – UI застосунку (скріншот з ПЗ)

Для покращення взаємодії реалізовано візуальні підказки та блокування елементів під час запитів. Наприклад, при транскрипції голосу або надсиланні повідомлення весь інтерфейс частково затемнюється, з’являється індикатор завантаження, а активні кнопки стають недоступними. Це запобігає випадковим діям та інформує користувача про поточну обробку. Видалення чатів із підтвердженням реалізовано через модальне вікно, як запропоновано в [26].

Також застосунок реагує на зміну даних автоматично — використання двостороннього прив’язування забезпечує оновлення вмісту без потреби в ручному втручанні. Завдяки такому підходу користувацький досвід залишається плавним і передбачуваним, а взаємодія з програмою — ефективною та комфортною. Елементи мають високу швидкість оновлення інформації, що не створюватиме незручностей у використанні застосунку з метою економії часу.

# 4 Опис прийнятих програмних рішень

## 4.1 Опис підключення OpenAI API

У розробленому мною десктопному асистенті підключення до OpenAI API реалізовано в основному класі MainWindow, який наслідується від Window та представляє головне вікно WPF-додатку. Для роботи з різними сервісами OpenAI використано кілька спеціалізованих клієнтів, кожен з яких відповідає за конкретну функціональність системи.

Підключення до OpenAI здійснюється через набор приватних полів класу, які ініціалізуються при запуску додатку. Система включає OpenAIClient як основний клієнт, OpenAIFileClient для роботи з файлами, AssistantClient для управління асистентами, AudioClient для транскрипції аудіо та VectorStoreClient для роботи з векторними сховищами документів. Актуальні дослідження демонструють ефективність використання вбудованих латентних представлень для класифікації запитів у мультимовному середовищі, спираючись на інформацію зі сторонніх джерел [27].

private void InitializeAssistant()

{

InputBox.IsEnabled = false;

SendButton.IsEnabled = false;

UploadButton.IsEnabled = false;

RecordButton.IsEnabled = false;

var apiKey = Environment.GetEnvironmentVariable("OPENAI\_API\_KEY");

if (string.IsNullOrWhiteSpace(apiKey))

{

MessageBox.Show("API key not found in environment variables.",

"Missing API Key", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

whisperClient = new AudioClient("whisper-1", apiKey);

client = new OpenAIClient(apiKey);

fileClient = client.GetOpenAIFileClient();

vectorStoreClient = client.GetVectorStoreClient();

assistantClient = client.GetAssistantClient();

ThreadListBox.ItemsSource = ThreadStorage.LoadThreads();

}

Для забезпечення безпеки API ключ зберігається в змінних середовища системи, що дозволяє уникнути хардкодингу конфіденційної інформації в коді та забезпечує гнучкість конфігурації для різних середовищ розгортання. AudioClient ініціалізується з конкретною моделлю "whisper-1" для транскрипції аудіо записів користувача в текст з підтримкою української мови.

OpenAIFileClient використовується для завантаження документів користувача на сервери OpenAI, після чого VectorStoreClient додає ці файли до векторних сховищ для забезпечення пошуку по їх змісту. AssistantClient керує створенням та налаштуванням віртуальних асистентів з спеціалізованими інструкціями та інструментами пошуку файлів.

var assistantOptions = new AssistantCreationOptions

{

Name = "Resume Extractor",

Instructions = "Ти Асистент, який читає PDF файл та надає відповіді на запитання на мові, з якою користувач робив запит.",

Tools = { new FileSearchToolDefinition() },

ToolResources = new()

{

FileSearch = new()

{

VectorStoreIds = { vectorStore.Id }

}

}

};

var assistant = await assistantClient.CreateAssistantAsync("gpt-4o", assistantOptions);

Система включає перевірку наявності API ключа та відповідне повідомлення користувачу в разі його відсутності. При відсутності ключа всі елементи інтерфейсу залишаються неактивними, що запобігає спробам виконання операцій без автентифікації. Така архітектура підключення забезпечує модульність, безпеку та ефективне використання різних сервісів OpenAI API в рамках єдиного додатку.

4.2 Обробка та вивід введеної інформації

У розробленому мною додатку реалізовано комплексну систему обробки різних типів вхідної інформації, включаючи текстові повідомлення, файли документів та аудіо записи. Центральним елементом цієї системи є метод ProcessInput, який забезпечує уніфікований підхід до обробки всіх типів користувацького вводу та координує взаємодію з OpenAI API.

Текстова інформація надходить через текстове поле InputBox, де користувач може вводити запити безпосередньо або через обробку завантажених файлів. Система автоматично визначає тип вводу - чи це прямий текстовий запит, чи шлях до файлу, який потребує попередньої обробки. При натисканні кнопки відправки або клавіші Enter активується процес обробки, який спочатку блокує всі елементи інтерфейсу для запобігання множинних запитів та додає повідомлення користувача до чату.

private async Task ProcessInput(string input)

{

if (string.IsNullOrWhiteSpace(input)) return;

InputBox.IsEnabled = false;

SendButton.IsEnabled = false;

UploadButton.IsEnabled = false;

RecordButton.IsEnabled = false;

ThreadListBox.IsEnabled = false;

NewChat.IsEnabled = false;

AddChatMessage("User", input);

var thinkingMessage = new ChatMessage

{

Role = "Assistant",

Text = "[Assistant is thinking...]"

};

ChatList.Items.Add(thinkingMessage);

ChatList.ScrollIntoView(ChatList.Items[ChatList.Items.Count - 1]);

string prompt = File.Exists(input) ? ReadFileContent(input) : input;

var msgs = new List<MessageContent> { MessageContent.FromText(prompt) };

await assistantClient.CreateMessageAsync(threadId, MessageRole.User, msgs);

var run = await assistantClient.CreateRunAsync(threadId, assistantId);

}

Обробка файлів різних форматів здійснюється через спеціалізований метод ReadFileContent, який автоматично визначає тип файлу за розширенням та застосовує відповідний алгоритм вилучення тексту. Для текстових файлів використовується прямий читання вмісту, для документів Word застосовується OpenXML API, а для PDF файлів - бібліотека iText. Це забезпечує широку сумісність з різними типами документів, які користувачі можуть завантажувати для аналізу.

private string ReadFileContent(string filePath)

{

var ext = Path.GetExtension(filePath).ToLower();

if (ext == ".txt")

{

return File.ReadAllText(filePath);

}

else if (ext == ".docx" || ext == ".doc")

{

using var wordDoc = WordprocessingDocument.Open(filePath, false);

return wordDoc.MainDocumentPart.Document.Body.InnerText;

}

else if (ext == ".pdf")

{

using var reader = new PdfReader(filePath);

using var pdfDoc = new PdfDocument(reader);

string text = "";

for (int i = 1; i <= pdfDoc.GetNumberOfPages(); i++)

{

text += PdfTextExtractor.GetTextFromPage(pdfDoc.GetPage(i)) + "\n";

}

return text;

}

else throw new Exception("Unsupported file type.");

}

Аудіо записи обробляються через інтегровану систему запису та транскрипції, яка використовує бібліотеку NAudio для захоплення звуку з мікрофона та його конвертації у формат MP3. Процес запису контролюється через кнопку запису, яка змінює свій стан та вигляд залежно від поточного режиму. Під час запису аудіо дані зберігаються в пам'яті у форматі WAV з частотою дискретизації 44100 Гц та моно каналом, що забезпечує оптимальну якість для подальшої обробки.

Після завершення запису аудіо файл автоматично конвертується у MP3 формат за допомогою LAME енкодера та зберігається тимчасово на робочому столі з унікальним ім'ям. Потім файл передається до Whisper API для транскрипції з налаштуваними параметрами, включаючи вказівку мови як української та інструкцію щодо збереження оригінальної мови користувача. Після успішної транскрипції тимчасовий аудіо файл видаляється, а отриманий текст автоматично вставляється в поле вводу для подальшої обробки.

Завантаження файлів через інтерфейс здійснюється за допомогою стандартного діалогового вікна Windows, після чого файли передаються до OpenAI для зберігання та індексації у векторному сховищі. Цей процес дозволяє асистенту використовувати завантажені документи як джерела знань для відповіді на запити користувача. Система автоматично повідомляє користувача про успішне завантаження файлу та його додавання до векторного сховища через системне повідомлення в чаті.

Взаємодія з асистентом OpenAI відбувається через створення повідомлень у відповідному потоці (thread) та запуск виконання (run). Система постійно моніторить статус виконання запиту, очікуючи завершення обробки, після чого отримує відповідь асистента та відображає її в інтерфейсі чату. Під час обробки запиту користувач бачить індикатор "Assistant is thinking...", який замінюється на фактичну відповідь після завершення обробки.

private async void SaveRecordingAsMp3()

{

memoryStream.Position = 0;

string outputPath = GetUniqueRecordingPath();

try

{

using (var reader = new WaveFileReader(memoryStream))

using (var mp3Writer = new LameMP3FileWriter(outputPath, reader.WaveFormat, LAMEPreset.STANDARD))

{

reader.CopyTo(mp3Writer);

}

RecordButton.IsEnabled = false;

AudioTranscriptionOptions options = new()

{

ResponseFormat = AudioTranscriptionFormat.Simple,

Language = "uk",

Temperature = 0.0f,

Prompt = "Не перекладай мову користувача"

};

AudioTranscription transcription = await Task.Run(() =>

{

return whisperClient.TranscribeAudio(outputPath, options);

});

File.Delete(outputPath);

InputBox.Text = transcription.Text;

}

catch (Exception ex)

{

MessageBox.Show($"Failed to save MP3 or transcribe:\n{ex.Message}", "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

Відповіді асистента обробляються з урахуванням можливих анотацій, які видаляються з тексту для забезпечення чистого відображення. Система автоматично прокручує чат до останнього повідомлення та відновлює активність всіх елементів інтерфейсу після завершення обробки запиту. Така архітектура забезпечує плавну взаємодію користувача з системою та надійну обробку різних типів вхідних даних.

4.3 Збереження історії чатів з асистентом

У розробленому мною додатку реалізовано комплексну систему збереження та управління історією чатів з асистентом OpenAI, яка забезпечує персистентність даних між сесіями роботи програми. Система базується на локальному зберіганні метаданих чатів та взаємодії з OpenAI API для отримання повного вмісту конверсацій.

Основою системи збереження є клас ThreadInfo, який інкапсулює необхідну інформацію про кожен чат. Цей клас містить три ключові властивості: AssistantId для ідентифікації конкретного асистента OpenAI, ThreadId для ідентифікації потоку повідомлень та ChatName для відображення користувачу зрозумілої назви чату. Такий підхід дозволяє ефективно організувати зв'язок між локальними даними та ресурсами на серверах OpenAI.

Локальне збереження метаданих чатів здійснюється через статичний клас ThreadStorage, який управляє файлом threads.txt у директорії додатку. Файл використовує простий CSV-подібний формат, де кожен рядок представляє один чат у форматі "AssistantId,ThreadId,ChatName". Цей підхід забезпечує простоту реалізації та надійність збереження даних без необхідності використання складних баз даних.

public static void SaveThread(string assistantId, string threadId, string chatName)

{

string entry = $"{assistantId},{threadId},{chatName}";

if (!File.Exists(storagePath))

{

File.WriteAllText(storagePath, entry + Environment.NewLine);

}

else

{

var existing = new HashSet<string>(File.ReadAllLines(storagePath));

if (!existing.Contains(entry))

File.AppendAllText(storagePath, entry + Environment.NewLine);

}

}

Створення нового чату відбувається через метод NewChatButton\_Click, який координує створення векторного сховища, асистента та потоку повідомлень в OpenAI. Процес починається з блокування інтерфейсу та відображення оверлея для інформування користувача про виконання операції. Спочатку створюється векторне сховище для зберігання документів, потім налаштовується асистент з спеціалізованими інструкціями та інструментами пошуку файлів, і нарешті створюється новий потік для ведення конверсації.

private async void NewChatButton\_Click(object sender, RoutedEventArgs e)

{

OverlayGrid.Visibility = Visibility.Visible;

MainGrid.IsEnabled = false;

try

{

WelcomePanel.Visibility = Visibility.Collapsed;

var createOp = vectorStoreClient.CreateVectorStore(true);

vectorStore = createOp.Value;

var assistantOptions = new AssistantCreationOptions

{

Name = "Resume Extractor",

Instructions = "Ти Асистент, який читає PDF файл та надає відповіді на запитання на мові, з якою користувач робив запит.",

Tools = { new FileSearchToolDefinition() },

ToolResources = new()

{

FileSearch = new()

{

VectorStoreIds = { vectorStore.Id }

}

}

};

var assistant = await assistantClient.CreateAssistantAsync("gpt-4o", assistantOptions);

assistantId = assistant.Value.Id;

var thread = await assistantClient.CreateThreadAsync();

threadId = thread.Value.Id;

string chatName = "Review\_" + DateTime.Now.ToString("yyyyMMdd\_HHmmss");

ThreadStorage.SaveThread(assistantId, threadId, chatName);

ThreadListBox.ItemsSource = ThreadStorage.LoadThreads();

ThreadListBox.Items.Refresh();

ThreadListBox.SelectedIndex = ThreadListBox.Items.Count - 1;

}

finally

{

OverlayGrid.Visibility = Visibility.Collapsed;

MainGrid.IsEnabled = true;

}

}

Відображення списку чатів здійснюється через компонент ThreadListBox, який прив'язується до даних, завантажених з локального сховища. При ініціалізації додатку викликається метод LoadThreads, який зчитує файл метаданих та перетворює кожен рядок у об'єкт ThreadInfo. Система обробляє можливість наявності ком у назвах чатів, правильно об'єднуючи частини назви після розділення рядка.

public static List<ThreadInfo> LoadThreads()

{

var result = new List<ThreadInfo>();

if (!File.Exists(storagePath))

return result;

foreach (var line in File.ReadAllLines(storagePath))

{

var parts = line.Split(',');

if (parts.Length >= 3)

{

var chatName = string.Join(",", parts, 2, parts.Length - 2);

result.Add(new ThreadInfo

{

AssistantId = parts[0],

ThreadId = parts[1],

ChatName = chatName

});

}

}

return result;

}

Вибір існуючого чату активує метод ThreadListBox\_SelectionChanged, який завантажує повну історію повідомлень з OpenAI API. Процес включає очищення поточного відображення чату, встановлення ідентифікаторів асистента та потоку, та асинхронне отримання всіх повідомлень через GetMessagesAsync. Система обробляє як повідомлення користувача, так і відповіді асистента, видаляючи технічні анотації для чистого відображення тексту.

private async void ThreadListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ThreadListBox.SelectedItem is not ThreadInfo selectedThread)

{

WelcomePanel.Visibility = Visibility.Visible;

return;

}

WelcomePanel.Visibility = Visibility.Collapsed;

threadId = selectedThread.ThreadId;

assistantId = selectedThread.AssistantId;

ChatList.Items.Clear();

List<ChatMessage> allMessages = new List<ChatMessage>();

OverlayGrid.Visibility = Visibility.Visible;

MainGrid.IsEnabled = false;

try

{

await foreach (var message in assistantClient.GetMessagesAsync(threadId))

{

if (message.Role == MessageRole.Assistant || message.Role == MessageRole.User)

{

foreach (var content in message.Content)

{

string text = content.Text;

if (content.TextAnnotations != null)

{

foreach (var annotation in content.TextAnnotations)

text = text.Replace(annotation.TextToReplace, "");

}

allMessages.Insert(0, new ChatMessage

{

Role = text.StartsWith("[System]") ? "system" : message.Role.ToString(),

Text = text

});

}

}

}

foreach (var chatMessage in allMessages)

{

AddChatMessage(chatMessage.Role, chatMessage.Text);

}

}

finally

{

OverlayGrid.Visibility = Visibility.Collapsed;

MainGrid.IsEnabled = true;

}

}

Видалення чату реалізовано через контекстне меню з підтвердженням операції. Процес включає видалення асистента та потоку з OpenAI серверів, оновлення локального файлу метаданих через метод DeleteThread та оновлення інтерфейсу. Якщо видаляється поточно активний чат, система скидає стан до початкового екрану привітання та деактивує елементи управління. Метод DeleteThread використовує фільтрацію для видалення відповідного рядка з файлу, перезаписуючи його без видаленого запису.

Система забезпечує надійність через обробку помилок та використання блоків try-finally для гарантованого відновлення стану інтерфейсу. Така архітектура забезпечує надійну роботу з історією чатів та надійне збереження даних між сесіями використання додатку.

У результаті розробки програмної системи було створено десктопний застосунок, що дозволяє здійснювати аналіз змісту угод з використанням великих мовних моделей (LLM). Реалізований функціонал забезпечує завантаження документів у різних форматах, інтеграцію голосового вводу за допомогою Whisper, підтримку багатокрокового діалогу з GPT-моделлю, а також автоматичне збереження результатів у базу даних.

Система підтримує функціонал запитань до документа, виявлення ключових понять, виділення зобов’язань сторін, а також виявлення потенційних ризиків або суперечностей у змісті. Запити можуть бути введені вручну або за допомогою мікрофона, а відповіді відображаються у зручному форматі.

На рисунку 4.1 наведено приклад реального спілкування з асистентом:



Рисунок 4.1 – Чат з асистентом (скріншот з ПЗ)

Підсумовуючи, створена система є сучасним інструментом для підтримки прийняття рішень у сфері юридичного аналізу. Вона демонструє, як поєднання можливостей великих мовних моделей, голосового вводу та зручного інтерфейсу може підвищити ефективність взаємодії з документами та зменшити ризик людських помилок при їх тлумаченні.

# 5 Тестування розробленого програмного забезпечення

5.1 Тестування back-end частини застосунку

Для забезпечення стабільної та ефективної роботи програмної системи, призначеної для аналізу змісту юридичних угод із використанням штучного інтелекту, було проведено навантажувальне тестування основних функціональних компонентів. Особливістю обраного підходу стало використання вбудованих інструментів платформи .NET, зокрема механізму Stopwatch для вимірювання точного часу виконання окремих операцій у мілісекундах та секундах до сотих часток.

На відміну від класичних рішень із використанням сторонніх засобів (таких як Apache JMeter), цей підхід дозволив інтегрувати процес вимірювання безпосередньо у вихідний код і в реальному часі відслідковувати тривалість обробки запитів користувача. Таке тестування було особливо цінним на етапі розробки, коли необхідно було швидко оцінити вплив змін у коді на продуктивність системи.

У межах тестування моделювалися типові сценарії взаємодії з користувачем, зокрема:

* завантаження юридичного документа;
* надсилання текстового запиту;
* розпізнавання голосового вводу;
* формування відповіді від AI.

Для кожного сценарію запускалося тестування з таймером, після чого система повідомляла час виконання у чаті, що дозволяло оперативно фіксувати показники.

Фрагмент коду, який реалізує подібне вимірювання:

using System.Diagnostics;

namespace DiplomaWork

{

public class PerformanceTester

{

private Stopwatch \_stopwatch;

private string \_timerName;

public PerformanceTester()

{

\_timerName = String.Empty;

\_stopwatch = new Stopwatch();

}

public void StartTimer(string txt)

{

\_timerName = txt;

\_stopwatch.Reset();

\_stopwatch.Start();

Console.WriteLine($"[{\_timerName}]: Таймер запущено...");

}

public void StopTimer()

{

\_stopwatch.Stop();

double seconds = \_stopwatch.Elapsed.TotalSeconds;

Console.WriteLine($"[{\_timerName}]: Операція завершена. Час виконання: {seconds:F2} секунд.");

}

}

}

Нижче будуть наведені результати тестування при кабельному підключенні до інтернету зі швидкістю 90-100 мбіт/сек. Отримані дані використаємо для аналізу швидкодії застосунку у звичних умовах.

Середній час виконання операції видалення одного чату становить приблизно **1,41 секунди**: дані для аналізу середнього часу виконання наведені нижче.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,95 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,29 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,28 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,38 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,31 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,50 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,20 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,25 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,16 секунд.

[Deleting chat thread]: Таймер запущено...

[Deleting chat thread]: Операція завершена. Час виконання: 1,95 секунд.

Середній час створення нового чату становить 2,83 секунди, середній час завантаження чату у вікно програми — 0,45 секунди.

Загальний середній час на створення та відображення одного чату дорівнює приблизно 3,28 секунди.

Дані для аналізу середнього часу виконання наведені нижче.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 1,64 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,28 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 1,55 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,28 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 4,44 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,27 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 5,17 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,87 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 2,07 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,36 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 1,75 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,23 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 2,01 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,38 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 2,21 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,60 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 2,82 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,24 секунд.

[Creating new chat]: Таймер запущено...

[Creating new chat]: Операція завершена. Час виконання: 4,60 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,94 секунд.

Середній час завантаження одного чату становить 0,32 секунди.

Дані для аналізу середнього часу виконання наведені нижче.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,36 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,25 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,32 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,29 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,25 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,25 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,29 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,32 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,28 секунд.

[Loading selected chat thread]: Таймер запущено...

[Loading selected chat thread]: Операція завершена. Час виконання: 0,42 секунд.

Середній час завантаження одного файлу до векторного сховища становить 4,18 секунди.

Дані для аналізу середнього часу виконання наведені нижче.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,04 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,22 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,53 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,61 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 3,82 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 5,07 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,02 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 4,04 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 3,62 секунд.

[Uploading file to vector store]: Таймер запущено...

[Uploading file to vector store]: Операція завершена. Час виконання: 3,80 секунд.

Середній час розпізнавання одного голосового повідомлення змістом на одне-два речення становить 1,09 секунди. Повідомлення були складені українською мовою, а замір часу відбувався від завершення запису з мікрофону до отримання розпізнаного тексту.

Дані для аналізу середнього часу виконання наведені нижче.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,45 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,03 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 0,67 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 0,82 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,07 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 0,56 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,38 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,64 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,04 секунд.

[Processing recording]: Таймер запущено...

[Processing recording]: Операція завершена. Час виконання: 1,64 секунд.

Середній час генерації відповіді на один запит з одного-двох речень становить **10,30 секунди.**

Дані для аналізу середнього часу виконання наведені нижче.

[Processing input text]: Операція завершена. Час виконання: 11,59 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 10,38 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 11,30 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 8,56 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 9,38 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 11,97 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 8,81 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 6,95 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 14,22 секунд.

[Processing input text]: Таймер запущено...

[Processing input text]: Операція завершена. Час виконання: 13,78 секунд.

Цей підхід дозволив:

* оцінити середній час обробки одного запиту (близько 10.3 с для локальної моделі);
* виявити найбільш навантажені ділянки коду;
* провести ітеративну оптимізацію логіки взаємодії з API.

Отримані результати свідчать про задовільну продуктивність системи при типовому навантаженні. Водночас дана методика закладає основу для масштабованого автоматизованого тестування в майбутньому, зокрема в умовах паралельного доступу від кількох користувачів. Наступним кроком варто провести тестування користувацького інтерфейсу програми.

Також, можна додатково ознайомитись з порівнянням швидкодії ядра нашого програного застосунку (OpenAI API, gpt-4 model) з відомими аналагоми у Додатку Д.

5.2 Тестування UI/UX застосунку

Наведемо опис процесу оцінки інтерфейсу користувача (UI) програмної системи, зосередженого на інтуїтивності, зрозумілості та загальній зручності використання (UX) у чат-орієнтованому застосунку.

Тестування інтуїтивності та зручності інтерфейсу було проведено із залученням 10 студентів з Харківського Національного Університету Радіоелектроніки віком від 20 до 21 років, які не мали попереднього досвіду взаємодії з цією системою. Мета тестування полягала у виявленні того, наскільки інтерфейс дозволяє користувачам без сторонньої допомоги виконати базові дії: створити новий чат, ввести повідомлення, надіслати його, ініціювати голосове введення або завантажити документ. Початковим вікном для тестування було обрано початкове вікно програми з уже існуючими чатами. (див. рис. 5.1).

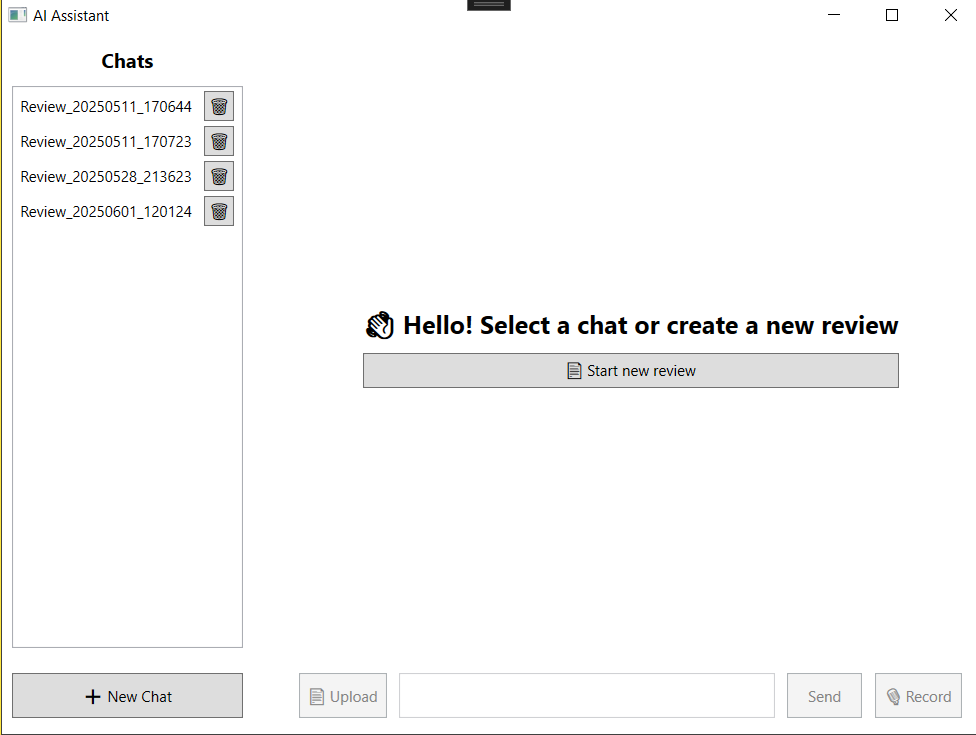


Рисунок 5.1 – Вікно, з якого починалось тестування (скріншот з ПЗ)

Користувачі отримали наступне завдання: самостійно розпочати діалог із системою за допомогою будь-якого методу введення (текст або мікрофон) та, за бажання, завантажити документ. Спостереження здійснювалось у формі "thinking aloud", тобто учасники коментували свої дії під час взаємодії.

Результати показали, що:

* 10 з 10 респондентів без труднощів ідентифікували поле для введення тексту завдяки його розміщенню в нижній частині інтерфейсу та стандартному вигляду;
* 9 з 10 без труднощів змогли створити новий чат.
* 9 з 10 одразу знайшли кнопку «Send» праворуч від поля введення та зрозуміли її призначення;
* 8 з 10 інтерпретували іконку мікрофона як засіб голосового введення;
* 8 з 10 змогли одразу зрозуміти, як видаляються чати, та успішно скористатися цим. Також вони підтвердили доцільність віка підтвердження видалення чату [26];
* лише 7 з 10 правильно визначили призначення кнопки з іконкою документу, а інші вагалися щодо її істинного призначення або плутали її з кнопкою для прикріплення файлів (як у месенджерах).

Середня оцінка зрозумілості елементів інтерфейсу за п’ятибальною шкалою (1 – незрозуміло, 5 – інтуїтивно):

* поле введення тексту: 4,9;
* кнопка надсилання: 4,8;
* мікрофон: 4,2;
* завантаження документа: 4,0.

Зауваження, які виникли під час тестування:

* деякі користувачі (3 з 10) очікували, що кнопка завантаження матиме підпис або розгорнуту підказку при наведенні курсору;
* назви існуючих чатів більше виглядають як ідентифікатори, і зрозуміти, що в них записана саме дата створення чату – важко;
* один учасник зазначив, що при першому запуску застосунку йому не було очевидно, що потрібно натиснути «New Chat», оскільки відсутнє автозапрошення чи візуальний фокус на цій кнопці;
* інший респондент зазначив, що кнопка голосового введення не мала зворотного зв'язку у вигляді зміни іконки або кольору, що змушувало сумніватися, чи функція активувалася;
* з восьми людей, що змогли одразу протестувати взаємодію з мікрофоном, двоє висловило думку, що варто кнопку стоп, яка з’являється під час запису, зробити червоного кольору;
* була виявлена проблема, яка полягала у неможливості вибору мікрофону для запису голосу. Двоє людей саме через цю проблему не змогли скористатися мікрофоном, адже за замовчуванням у їхній операційній системі був увімкнений віртуальний мікрофон, і саме він використовувався програмою для запису голосу;
* було зафіксовано непослідовність у стилях деяких елементів (наприклад, відсутність однакових відступів між кнопками), що хоч і не перешкоджало взаємодії, але впливало на загальне візуальне сприйняття.

Загалом 90% користувачів змогли виконати всі необхідні дії без сторонньої допомоги, що дозволяє вважати UI достатньо інтуїтивним для базового використання. Разом з тим, деякі аспекти — зокрема, додаткові підказки, візуальні ефекти натискання та покращення послідовності елементів — рекомендовано доопрацювати відповідно до сучасних принципів проєктування інтерфейсів чат-застосунків, описаних у матеріалах Google Material Design [28].

Під час тестування UI/UX програмної системи також було перевірено поведінку інтерфейсу користувача в ситуаціях, коли виконуються асинхронні операції, зокрема завантаження файлів і створення нових чатів. Під час завантаження, користувач мав отримувати екран блокування введення, що наведений на рисунку 5.2.

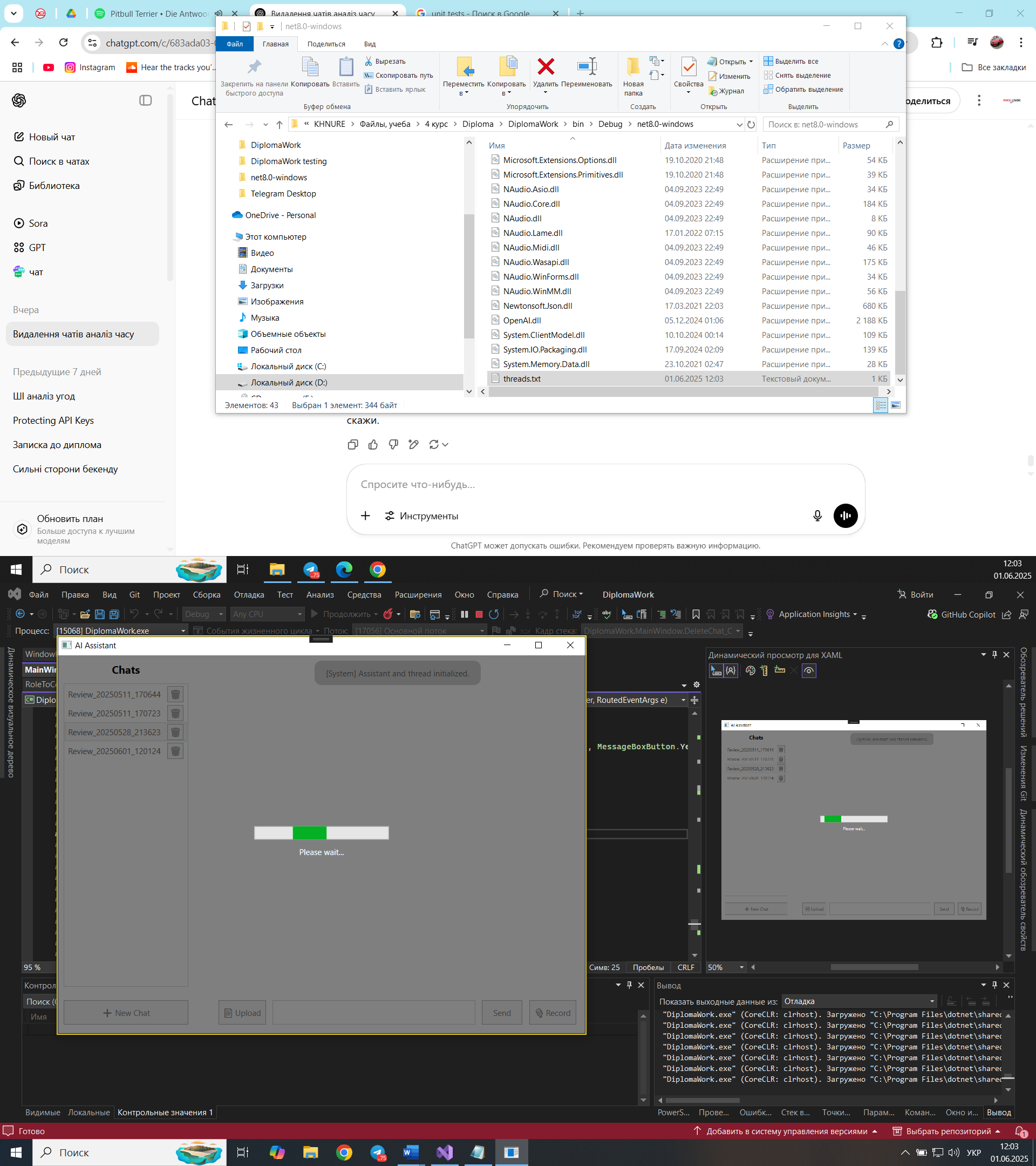


Рисунок 5.2 – UI застосунку при завантаженні файлу чи створенні нового чату (скріншот з ПЗ)

Основною метою було оцінити, чи можливо користувачеві взаємодіяти з елементами інтерфейсу в момент, коли це не передбачено логікою програми.

У випадку завантаження документа було реалізовано повне блокування елементів керування за допомогою виведення «OverlayGird» – UI-елементу, що перешкоджав взаємодії з програмою поки та не обробить дію. Проведено 30 симульованих спроб натиснення на кнопки («Send», «Record», «Upload», вибір чату) та введення тексту в «InputBox» під час завантаження — у 100% випадків усі елементи залишалися заблокованими, взаємодія була неможливою. Це свідчить про правильну реалізацію механізму блокування.

Натомість у ситуації створення нового чату виявлено незначну вразливість. Було проведено 20 спроб створення нового чату з одночасним натисканням кнопки «Send» під час короткого періоду активації «OverlayGrid». У 17 з 20 випадків не вдавалось взаємодіяти з програмою, однак у 3 спробах (15%) користувачу вдалося натиснути кнопку саме в момент перемикання стану — що свідчить про недопрацювання, пов’язане з багатопоточністю і несинхронною обробкою зміни станів інтерфейсу.

Один із тест кейсів:

Сценарій: користувач створює новий чат і одразу натискає «Send».

Передумови: у методі створення чату активується «OverlayGrid» і асинхронно запускається завантаження даних.

Кроки:

* користувач натискає «New Chat»;
* одразу (в перші 100–300 мс) натискає кнопку «Send»;
* повторює дію 20 разів.

Результати:

* у 17 з 20 випадків кнопка була заблокована до моменту натискання;
* у 3 випадках натискання відбулося під час переходу, що призвело до неочікуваної реакції (відправлення порожнього повідомлення або помилки).

Таким чином, система демонструє повністю надійне блокування при завантаженні файлів (100% стабільності) і майже надійне при створенні чату (85%). З урахуванням типу проблеми (некритична, виникає лише за швидких натискань у вузькому таймфреймі), результат вважається загалом задовільним, однак рекомендовано доопрацювати логіку блокування з урахуванням потокобезпеки, наприклад, шляхом централізованого керування станом UI через «Dispatcher.Invoke» або використання механізму «SemaphoreSlim».

Візуальні сигнали (затемнення інтерфейсу, індикатор очікування) повністю відповідають очікуванням користувача та не викликають спроб взаємодії при активному процесі.

Отримані результати свідчать про високу стабільність інтерфейсу з одиничними винятками, які можливо усунути незначними змінами в логіці керування станами.

# Висновки

У результаті виконання кваліфікаційної роботи було розроблено десктопний застосунок «AI Assistant» для аналізу змісту юридичних угод з використанням можливостей великих мовних моделей. Програма призначена для персонального комп’ютера та реалізована у вигляді інтерактивного чат-інтерфейсу з підтримкою як текстової, так і голосової взаємодії.

Було проведено дослідження предметної галузі та проаналізовано існуючі рішення у сфері автоматизованого аналізу документів. На основі цього сформовано вимоги до функціоналу, серед яких — зручне управління чатами, перегляд і обробка документів, розпізнавання голосу та підтримка багатопоточності. Визначено ключові сценарії взаємодії користувача та сформовано архітектуру застосунку, орієнтовану на простоту, лінійність і стабільність роботи з API OpenAI.

У процесі розробки було реалізовано збереження чатів через локальну систему зберігання, розроблено асинхронну взаємодію з серверами OpenAI, а також вбудовано механізм запису аудіо, конвертації його в MP3 та подальшої транскрипції тексту через модель Whisper. Всі запити до серверу супроводжуються візуальними індикаторами очікування та блокуванням інтерфейсу задля покращення UX.

Для реалізації програмної системи було використано платформу .NET із застосуванням Windows Presentation Foundation (WPF), мову програмування C#, бібліотеку libmp3lame для аудіообробки та офіційні клієнти OpenAI для інтеграції мовних моделей.

Розроблений застосунок має потенціал до подальшого вдосконалення, зокрема — додавання розширеного аналізу юридичних термінів, підсвічування порушень у договорах та підтримки багатомовності.

# Перелік джерел посилання

1. Microsoft Learn. Binding in WPF [Електронний ресурс].  
URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/data-binding-overview> (дата звернення: 19.04.2024).

2. OpenAI. Assistants API documentation [Електронний ресурс].  
URL: <https://platform.openai.com/docs/assistants/overview> (дата звернення: 28.04.2024).

3. GitHub – openai/whisper [Електронний ресурс].  
URL: <https://github.com/openai/whisper> (дата звернення: 06.05.2024).

4.Erkut Erdem, Anette Frank, Letitia Parcalabescu та ін., "Neural Natural Language Generation: A Survey on Multilinguality, Multimodality, Controllability and Learning", *Journal of Artificial Intelligence Research*, 73 (2022): <https://doi.org/10.1613/jair.1.12918>.

**5.**Irina van der Vet та Leena Malkki, "Copilot in service: Exploring the potential of the large language model-based chatbots for fostering evaluation culture in preventing and countering violent extremism", Open Research Europe, 3:176 (2025): https://doi.org/10.12688/openreseurope.176.v2.

6. Anthropic. Claude 3 Model Card [Електронний ресурс]. URL: https://www.anthropic.com/news/claude-3-model-family (дата звернення: 30.04.2025).

7. Lexion Overview [Електронний ресурс]. URL: https://www.lexion.ai/product (дата звернення: 05.05.2025).

8. Kira Systems Features [Електронний ресурс]. URL: https://kirasystems.com/features/ (дата звернення: 09.05.2025).

9. Legartis. Swiss Prime Site: AI to Analyze Over 1’000 Contractual Relationships [Електронний ресурс]. URL: https://www.legartis.ai/blog/case-study-ai-contract-analysis (дата звернення: 25.04.2025).

10. Best Practice AI. AI Case Study | Mondelēz International saved on expenses by automating client contract analysis ensuring corporate spinoff delivered on-time and budget [Електронний ресурс]. URL: https://www.bestpractice.ai/ai-case-study-best-practice/mondel%C4%93z\_international\_saved\_on\_expenses\_by\_automating\_client\_contract\_analysis\_ensuring\_corporate\_spinoff\_delivered\_on-time\_and\_budget (дата звернення: 28.04.2025).

11. Best Practice AI. AI Case Study | Dell reduces costs while ensuring contractual obligations are met with contract analysis automation [Електронний ресурс]. URL: https://www.bestpractice.ai/ai-case-study-best-practice/dell\_reduces\_costs\_while\_ensuring\_contractual\_obligations\_are\_met\_with\_contract\_analysis\_automation (дата звернення: 30.04.2025).

12. Best Practice AI. AI Case Study | Deloitte saves between 20-90% time in reviewing complex documents such as contracts using machine learning [Електронний ресурс]. URL: https://www.bestpractice.ai/ai-case-study-best-practice/deloitte\_saves\_between\_20-90%25\_time\_in\_reviewing\_complex\_documents\_such\_as\_contracts\_using\_machine\_learning (дата звернення: 02.05.2025).

13. Carillion. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Carillion (дата звернення: 28.04.2025).

14. The Times. Firm falsified data before bidding for £1.8bn council contract [Електронний ресурс]. URL: https://www.thetimes.co.uk/article/firm-falsified-performance-before-bidding-for-18bn-council-contract-z30j8ftdd (дата звернення: 02.05.2025).

15. Statista. “Frequency of voice assistant usage in the U.S. 2023.” URL: https://www.statista.com/statistics/973815/us-voice-assistant-usage-frequency/ (дата звернення: 05.05.2025).

16. OpenAI. Whisper Speech Recognition Model [Електронний ресурс]. URL: https://github.com/openai/whisper (дата звернення: 04.05.2025).

17. Capgemini Research Institute. “Smart Talk: How organizations and consumers are embracing voice and chat assistants.” URL: https://www.capgemini.com/research/smart-talk/ (дата звернення: 28.04.2025).

18. Nataliia Saichyshyna, Daniil Maksymenko, Oleksii Turuta та ін., "Extension Multi30K: Multimodal Dataset for Integrated Vision and Language Research in Ukrainian", UNLP 2023: <https://huggingface.co/datasets/turuta/Multi30k-uk>.

19. Richter J. CLR via C#. Microsoft Press, 2012.

20. Daniil Maksymenko та Oleksii Turuta, "Interpretable Conversation Routing via the Latent Embeddings Approach", Computation, 12, № 237 (1 грудня 2024): <https://doi.org/10.3390/computation12120237>.

21. Albahari J., Albahari B. C# 10 in a Nutshell. O’Reilly Media, 2022.

22. LAME – High quality MPEG Audio Layer III (MP3) encoder [Електронний ресурс]. URL: <https://lame.sourceforge.io/> (дата звернення: 04.05.2024).

23. Material UI: React components that implement Material Design [Електронний ресурс]. URL: <https://mui.com/material-ui/> (дата звернення: 20.04.2024).

24. Baeldung. Introduction to WPF [Електронний ресурс].  
URL: <https://www.baeldung.com/cs/wpf-introduction> (дата звернення: 03.05.2024).

25. Petzold C. Programming Windows Presentation Foundation. Microsoft Press, 2006.

26. Stack Overflow – How to delete items in WPF list with confirmation [Електронний ресурс].  
URL: <https://stackoverflow.com/questions/31657228/delete-listboxitem-with-confirmation-in-wpf> (дата звернення: 07.05.2024).

27. Daniil Maksymenko, Danylo Kryvoshein, Olena Turuta та ін., "Benchmarking Conversation Routing in Chatbot Systems Based on Large Language Models", ProfIT AI 2024: <http://ceur-ws.org>.

28. Google Material Design. Communication Patterns: Chat [Електронний ресурс]. URL: <https://m3.material.io/patterns/communication/chat> (дата звернення: 27.05.2024).

29. Посилання на GitHub проєкту [Електронний ресурс]. URL: https://github.com/NureDulskyiDmytro/2025\_B\_PI\_PZPI-21-5\_Dulskyi\_D\_A.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

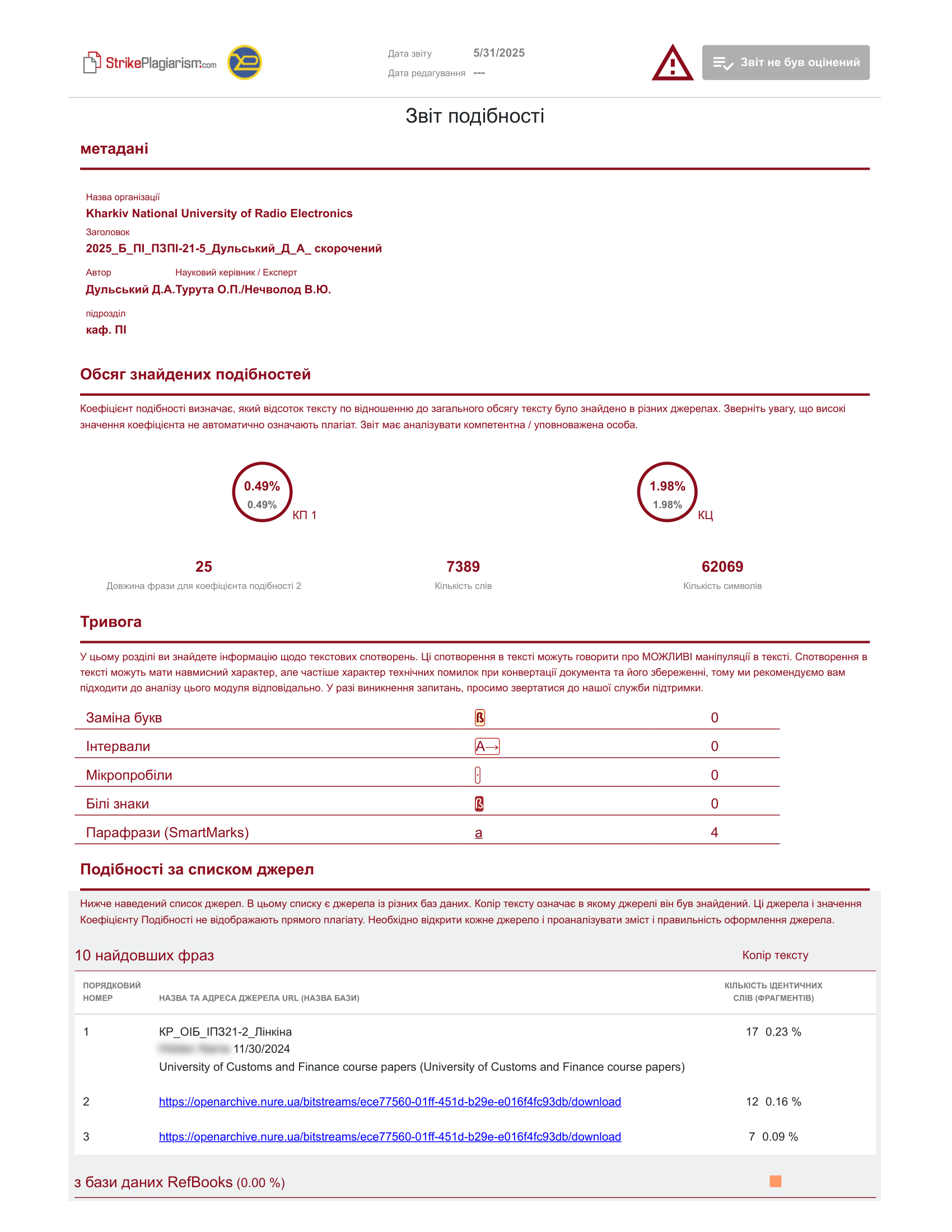


Рисунок А.1 – Звіт результатів перевірки на плагіат

ДОДАТОК Б

Слайди презентації

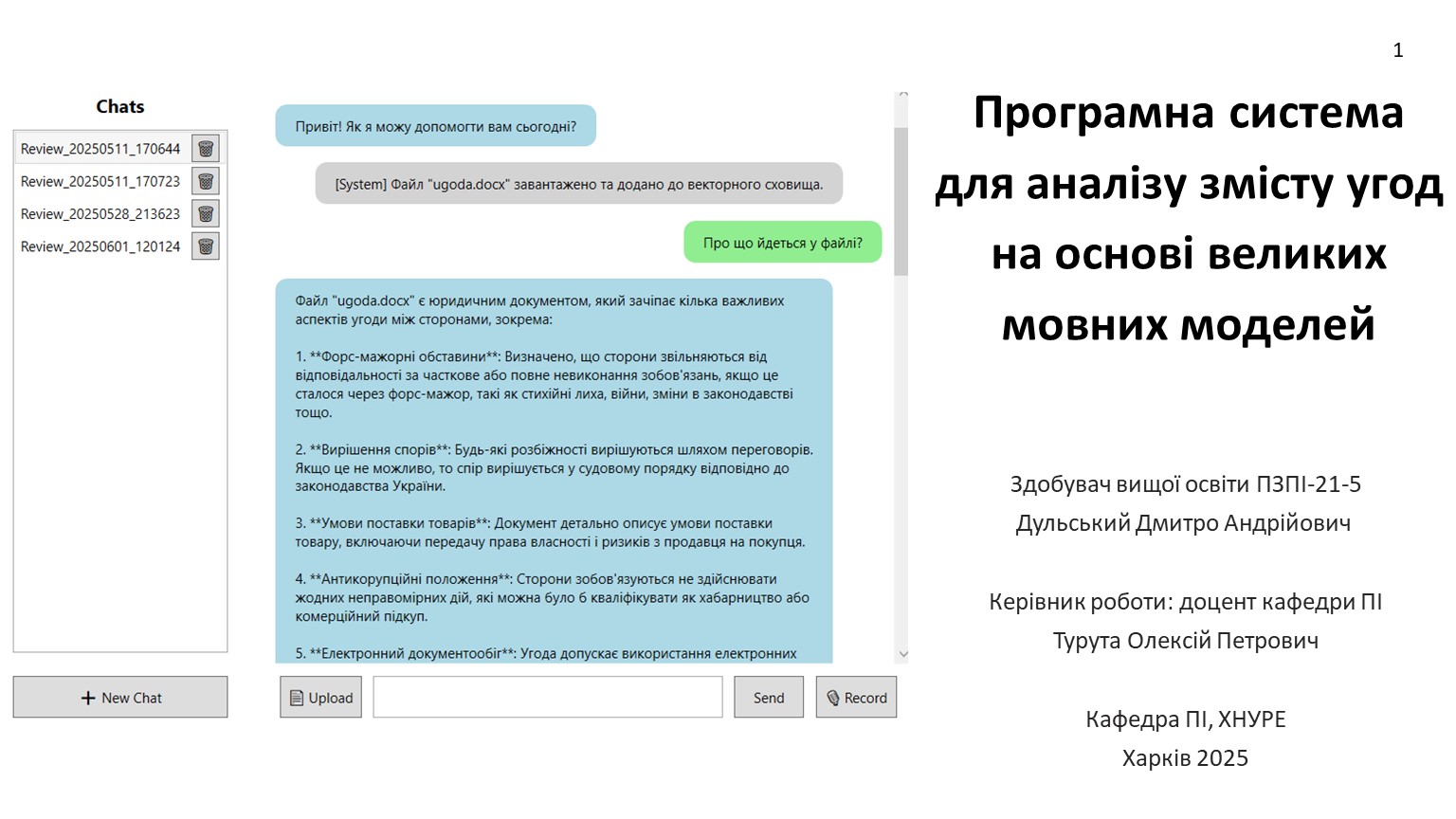


Рисунок Б.1 – Слайд 1

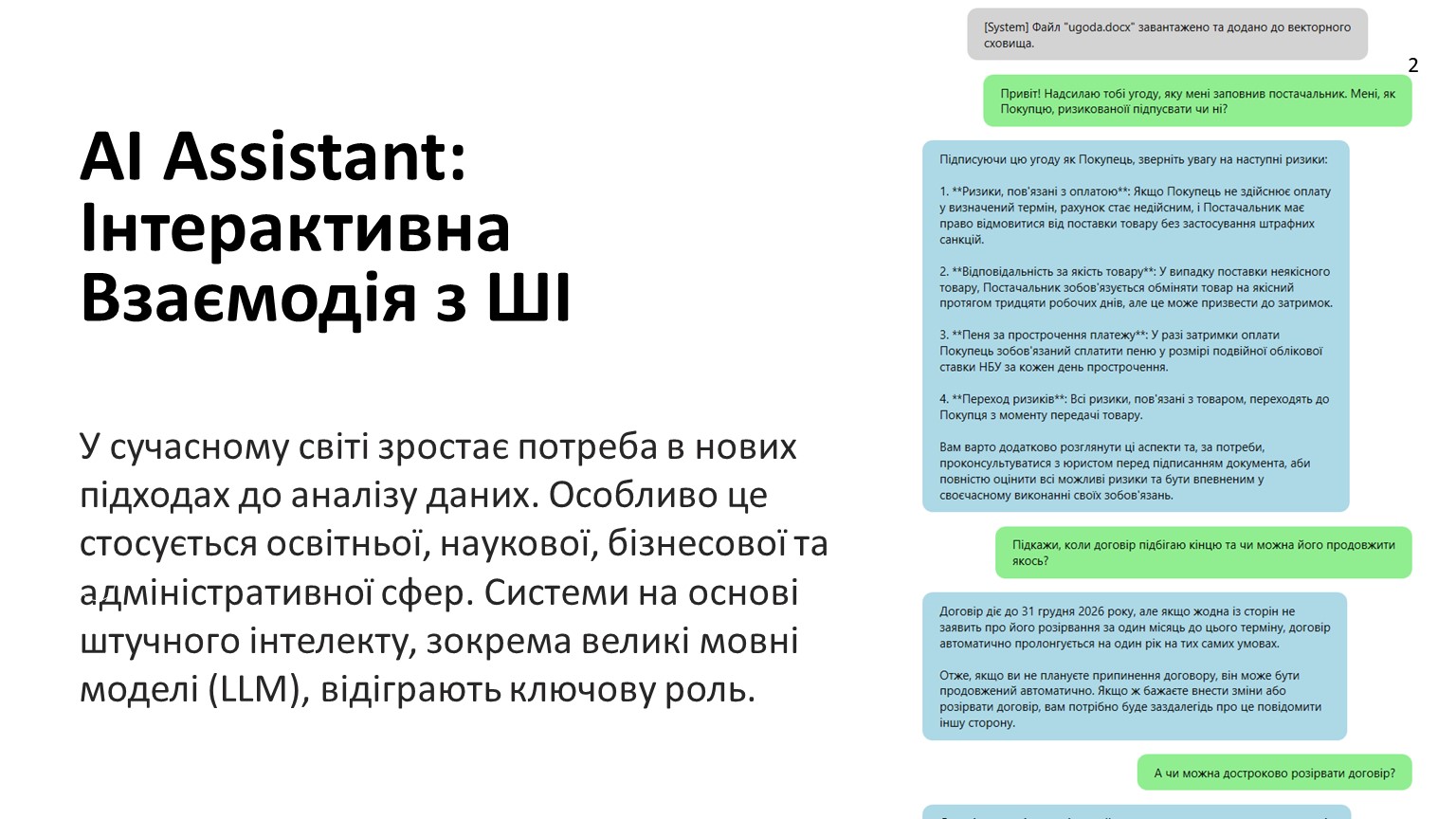


Рисунок Б.2 – Слайд 2

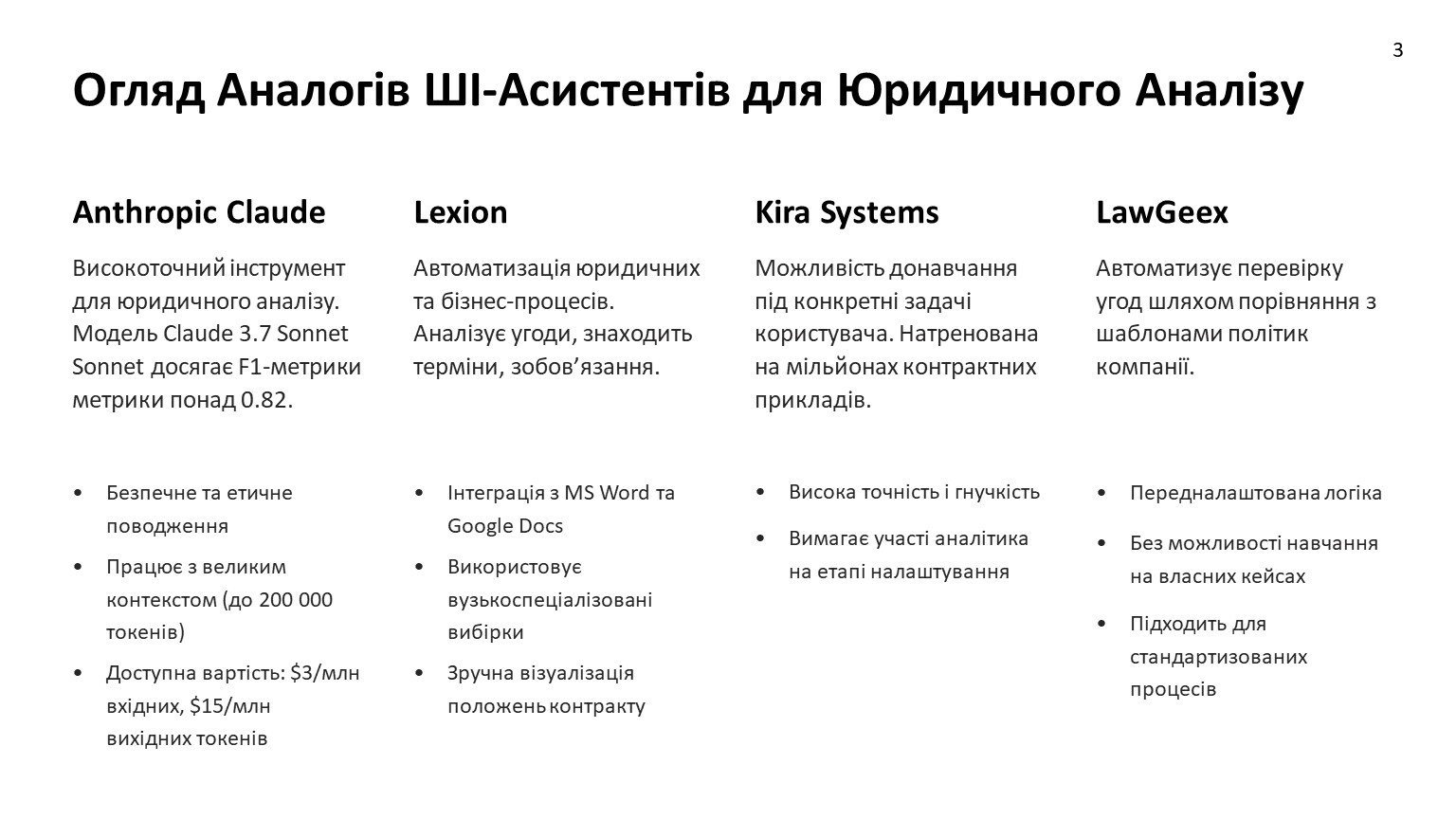


Рисунок Б.3 – Слайд 3

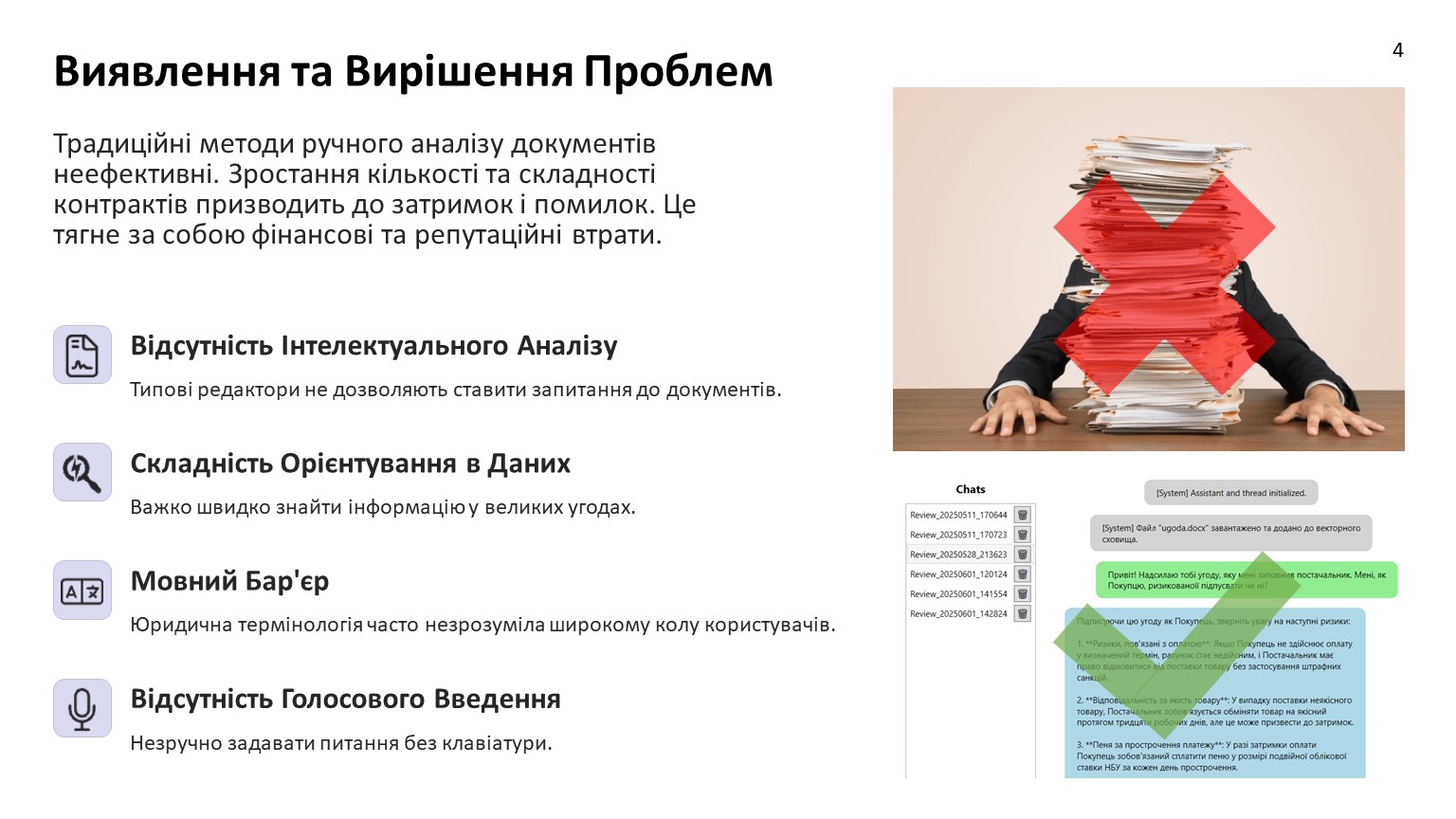


Рисунок Б.4 – Слайд 4

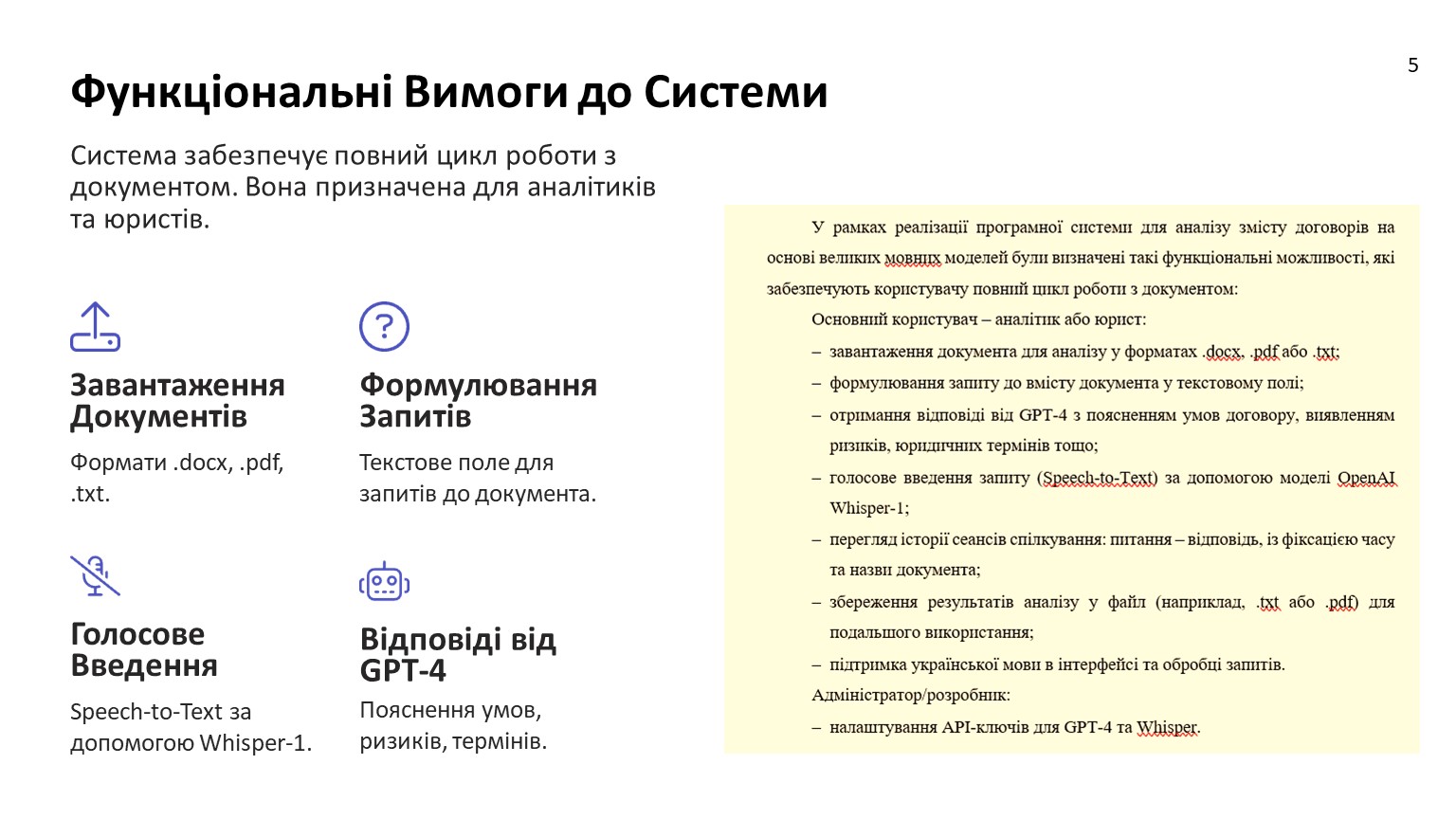


Рисунок Б.5 – Слайд 5



Рисунок Б.6 – Слайд 6



Рисунок Б.7 – Слайд 7

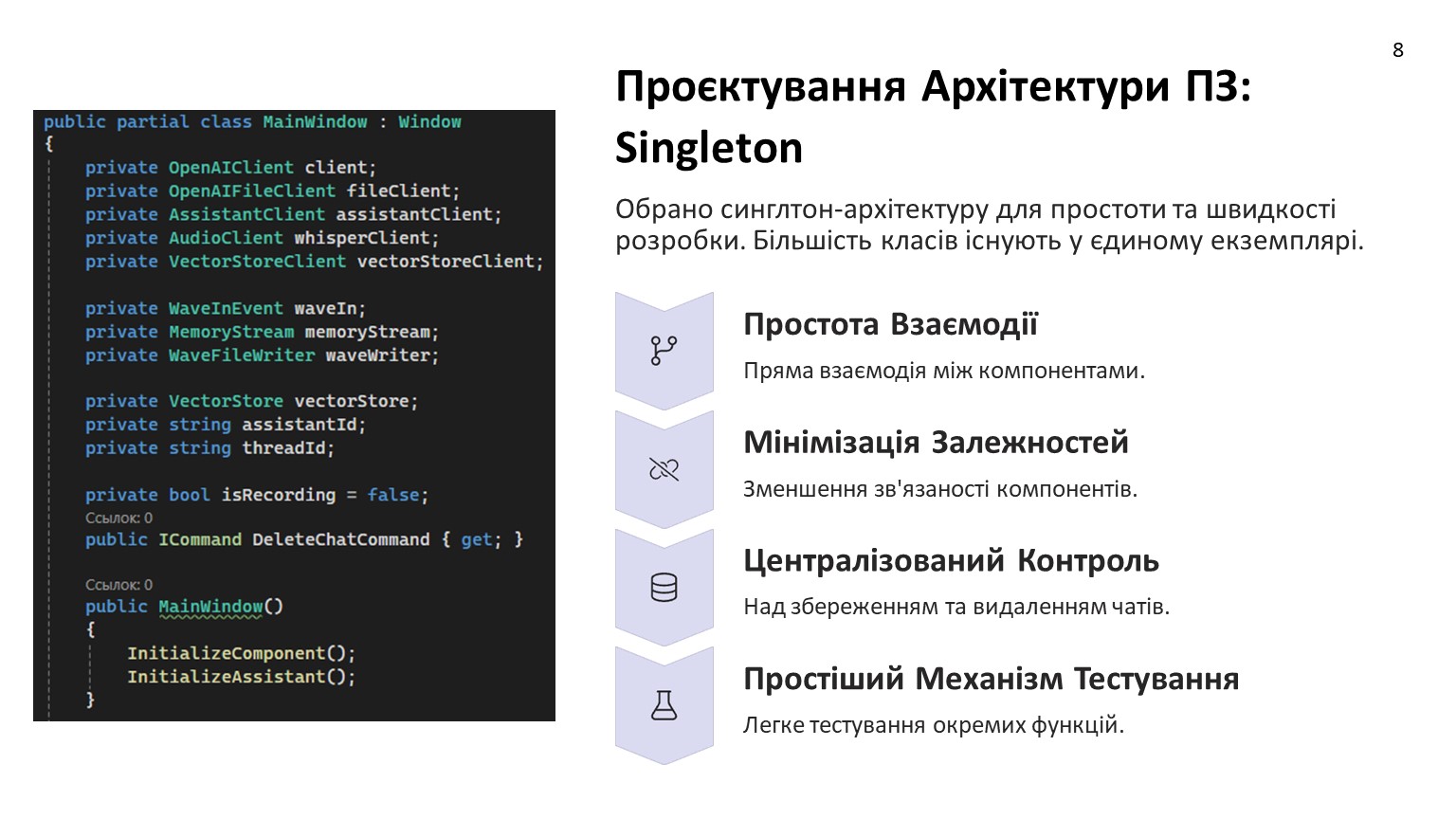


Рисунок Б.8 – Слайд 8

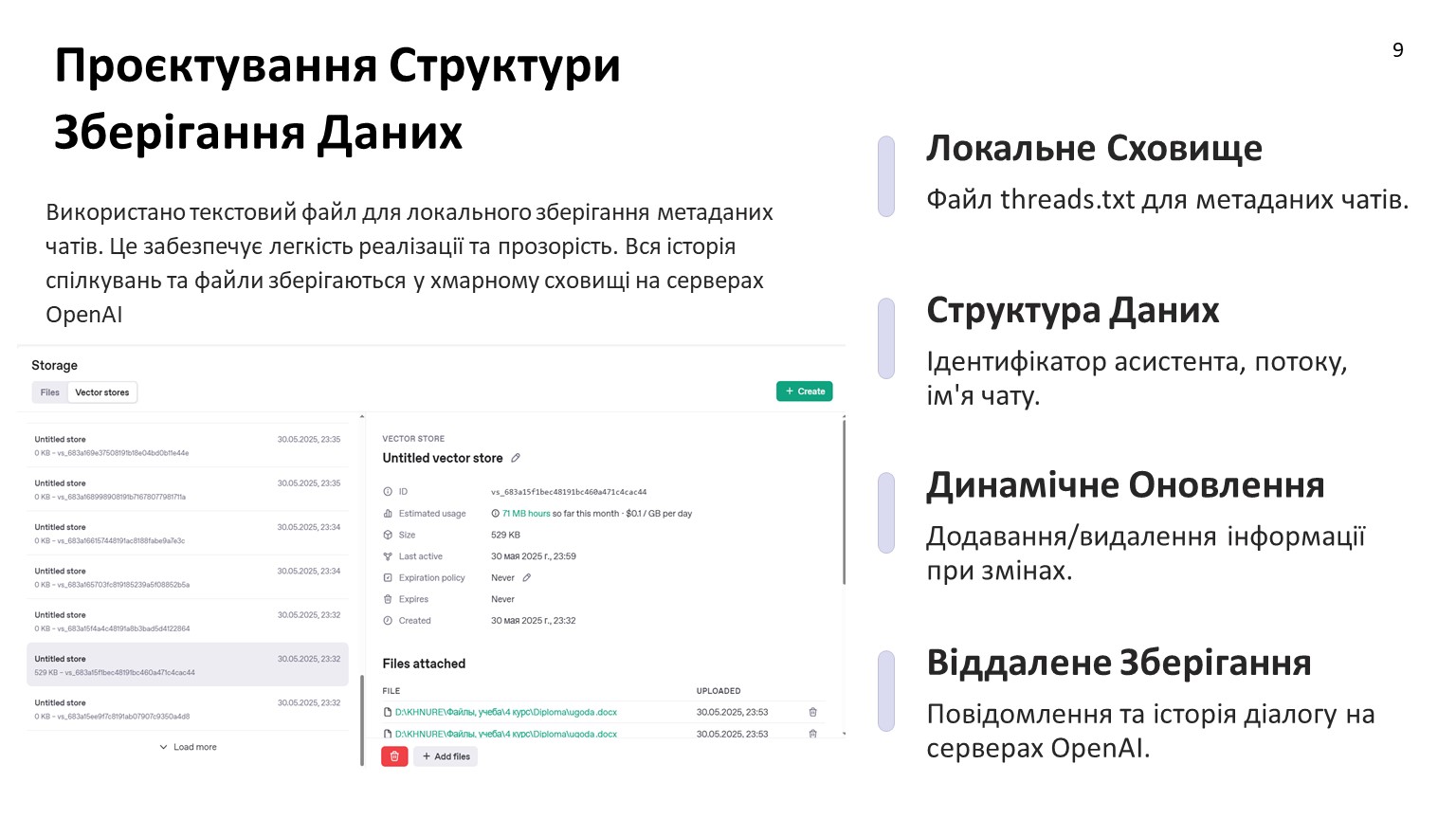


Рисунок Б.9 – Слайд 9



Рисунок Б.10 – Слайд 10

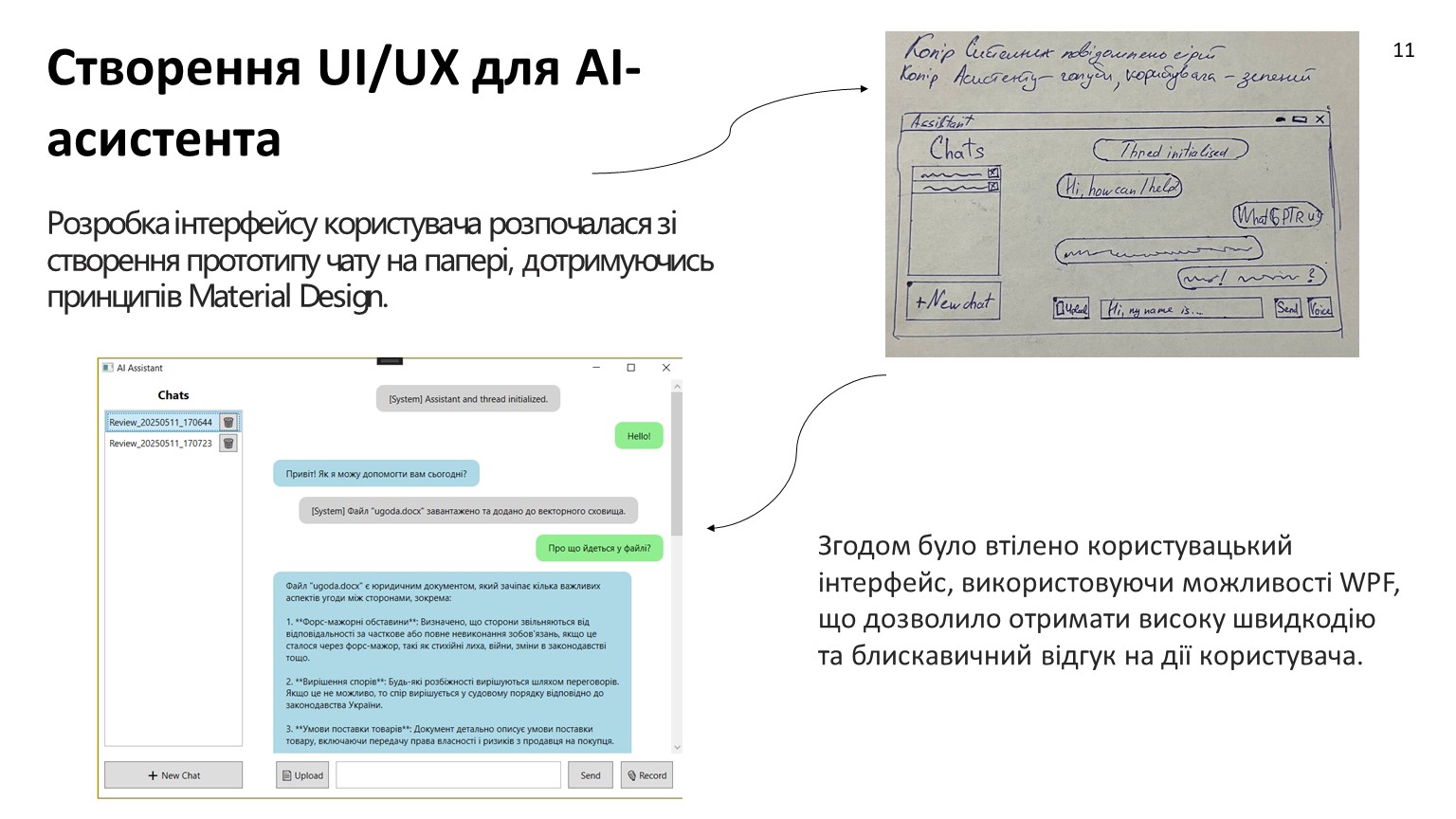


Рисунок Б.11 –Слайд 11

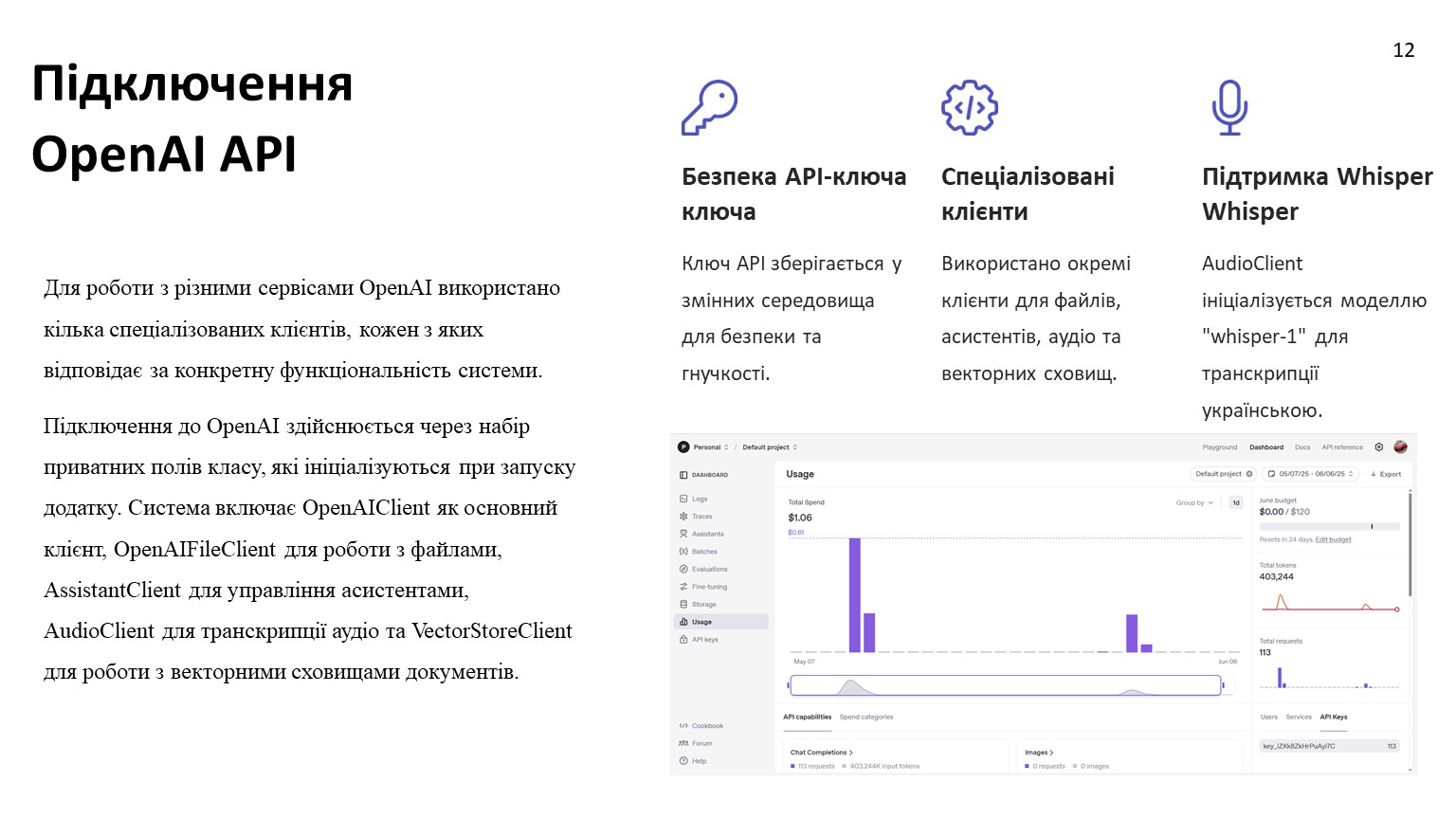


Рисунок Б.12 – Слайд 12

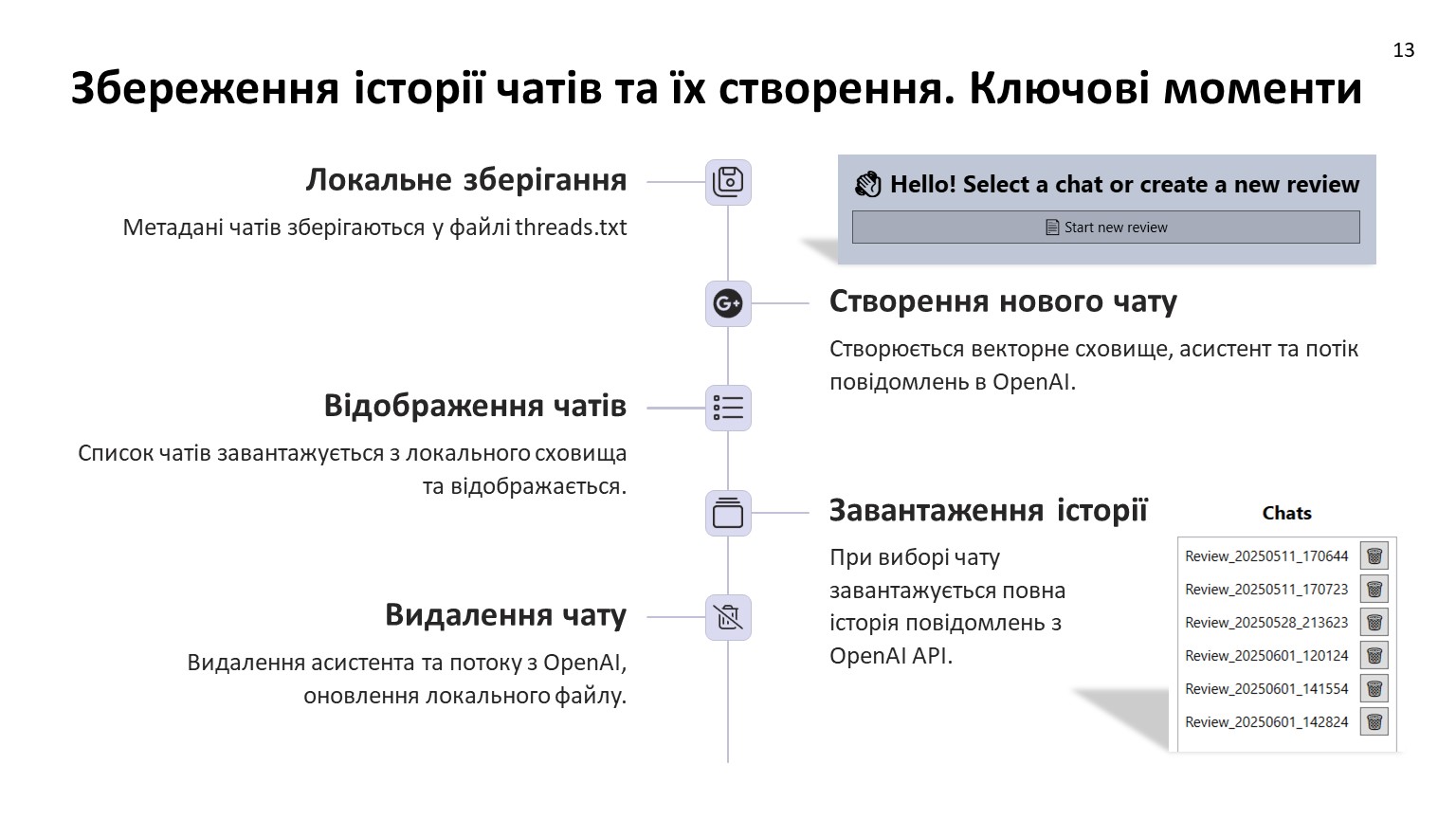


Рисунок Б.13 – Слайд 13

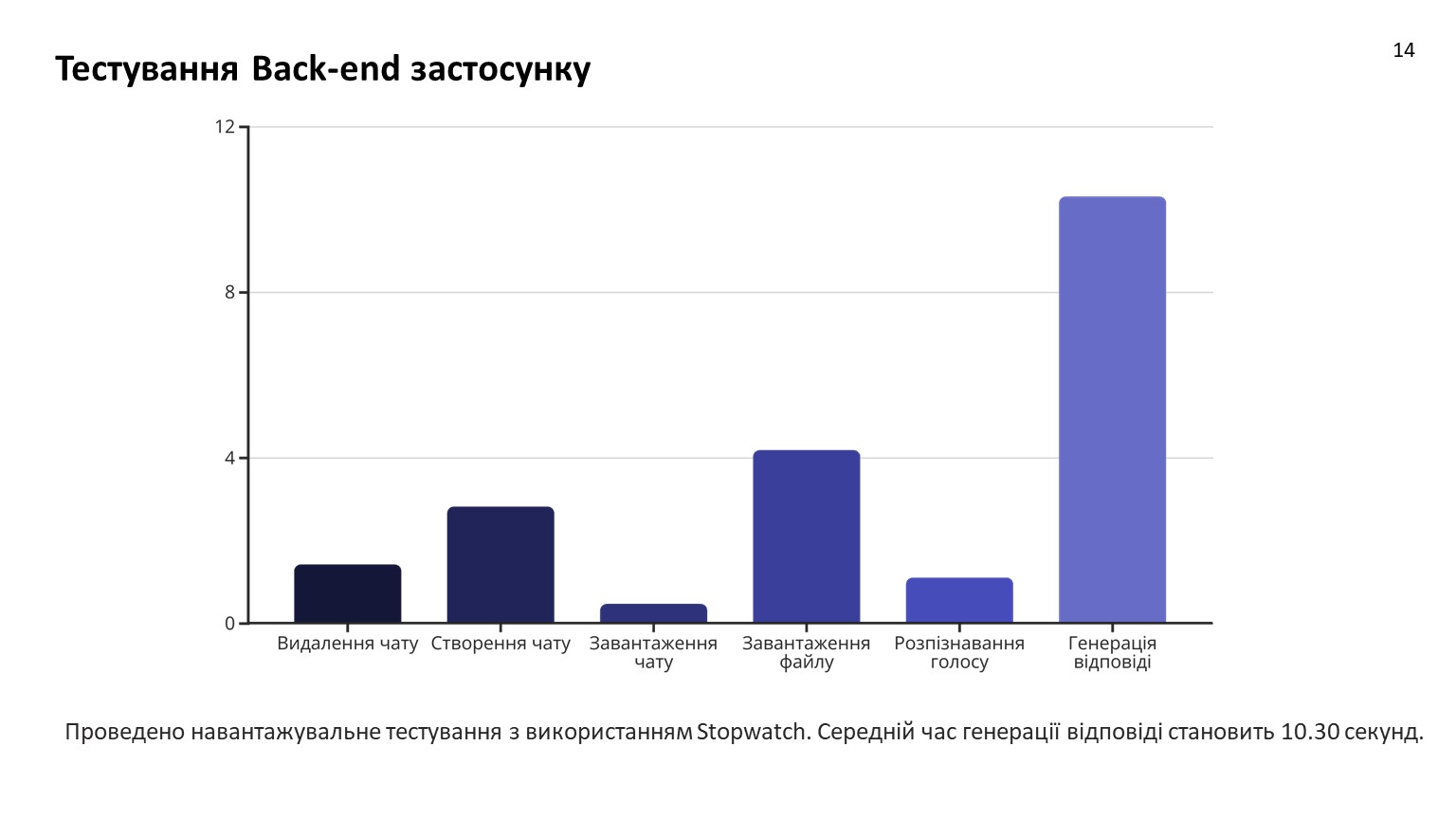


Рисунок Б.14 – Слайд 14

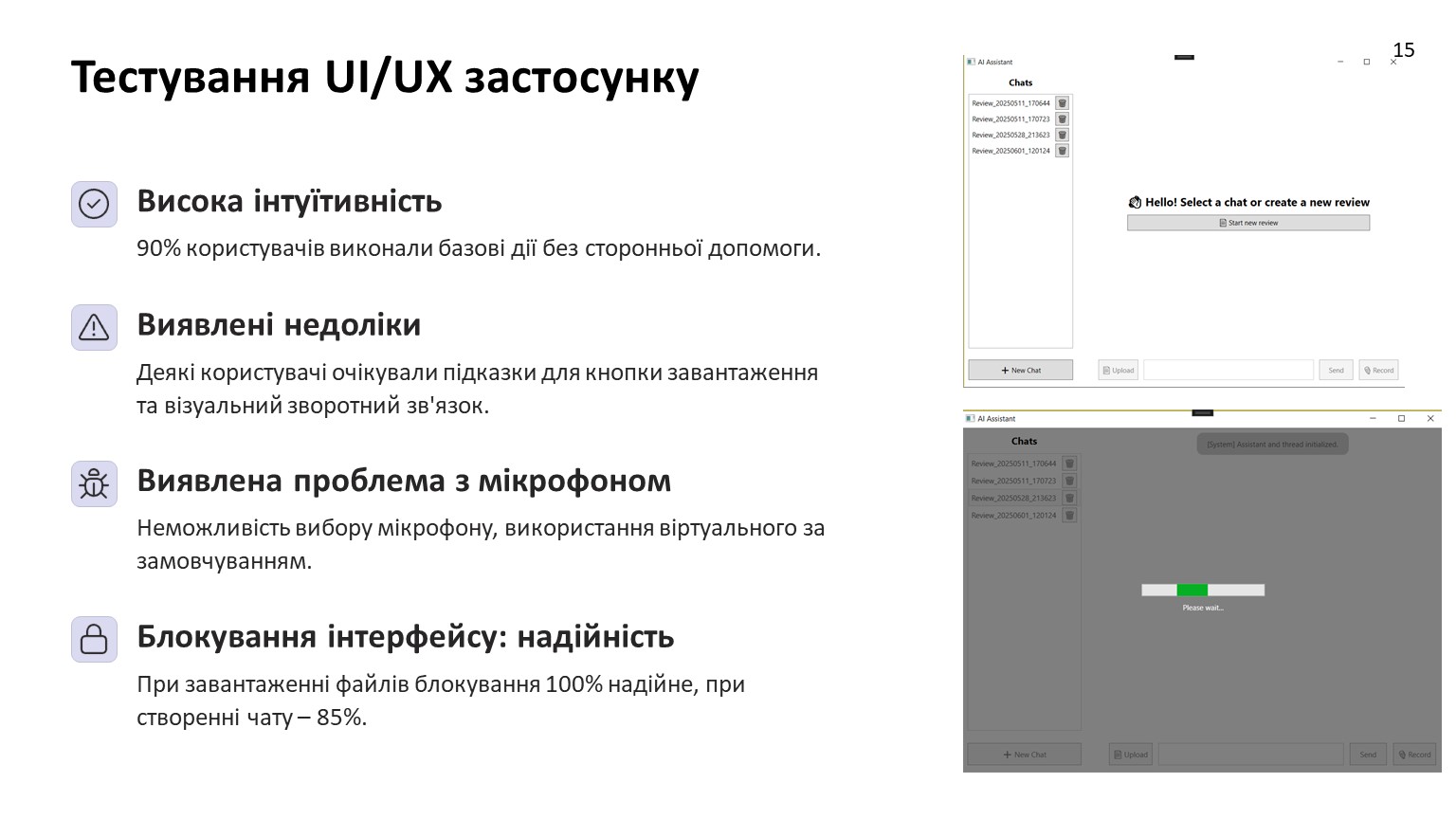


Рисунок Б.15 – Слайд 15



Рисунок Б.16 – Слайд 16

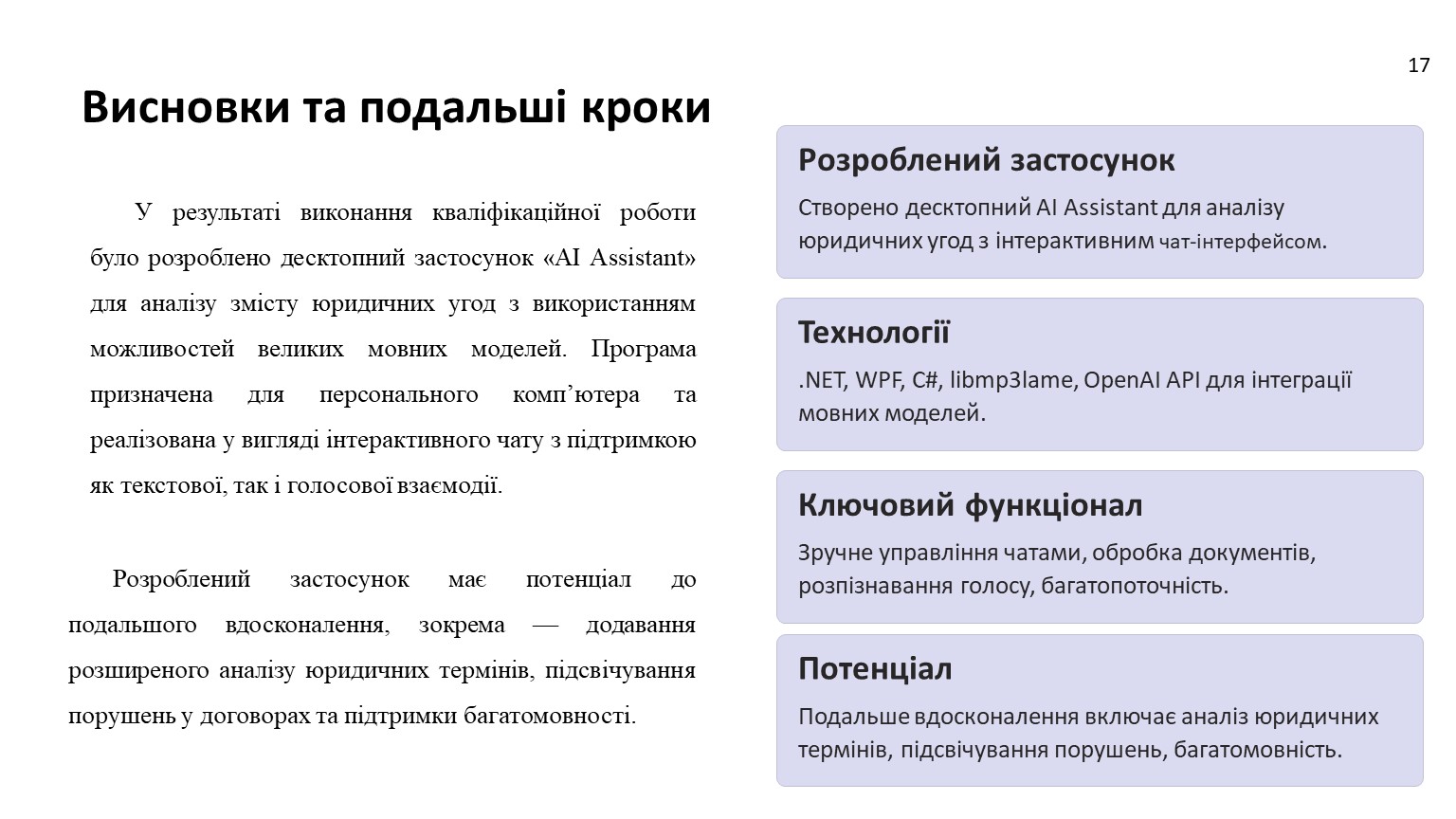


Рисунок Б.17 – Слайд 17

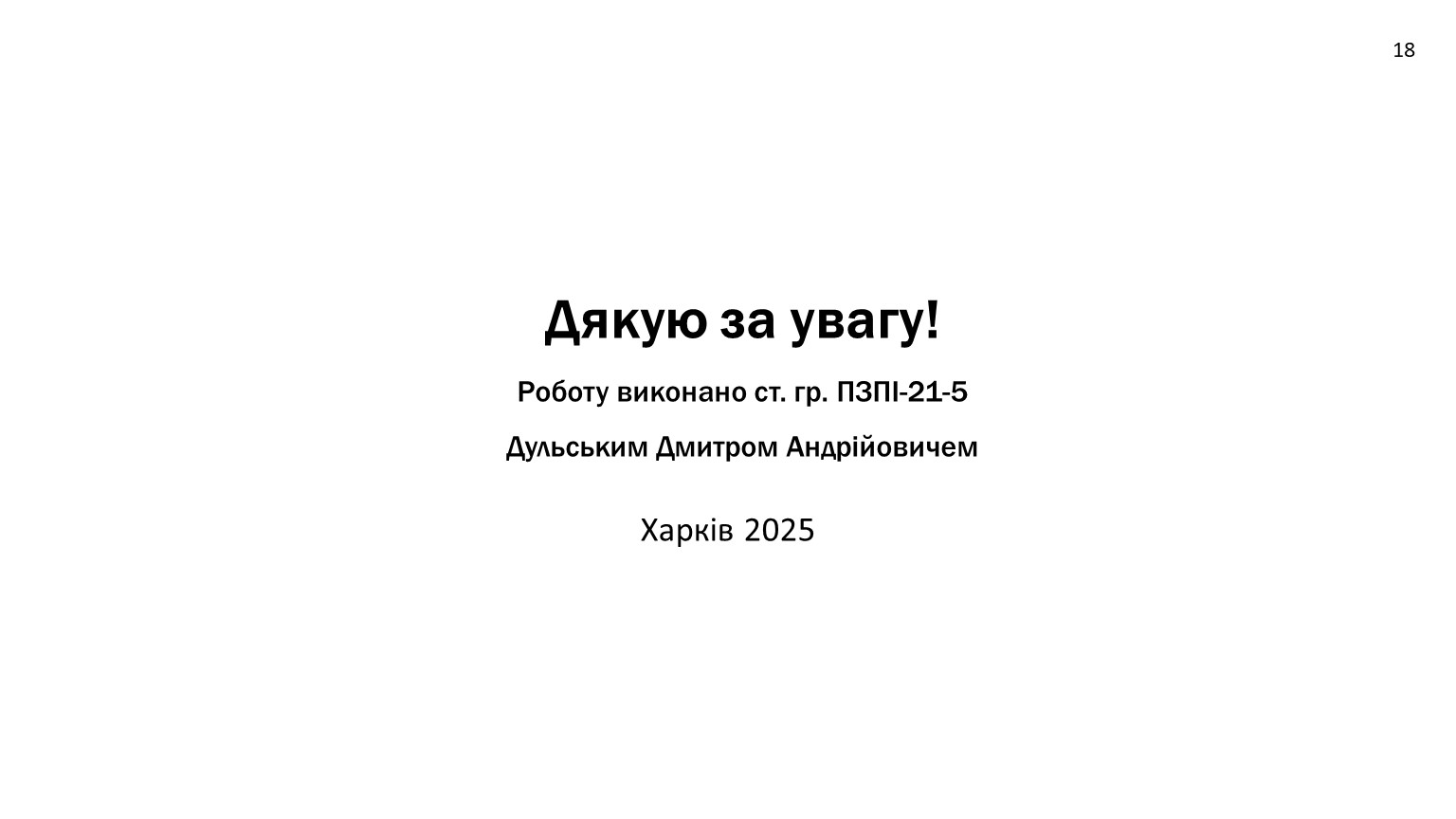


Рисунок Б.18 – Слайд 18

ДОДАТОК В

Порівняльний аналіз показників швидкодії відомих великих мовних моделей та розробленого програмного застосунку

Таблиця В.1 – Порівняльна таблиця швидкодії для Anthropic AI: Claude 3.7 Sonnet (WEB):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва великої мовної моделі | Anthropic AI: Claude 3.7 Sonnet (WEB) | | | |
| Номер Тесту | 1 | 2 | 3 | 4 |
| Час завантаження | 2,0 | 1,8 | 1,7 | 1,6 |
| Час відповіді | 21,0 | 26,7 | 23,8 | 21,3 |
| Загальний час | 23,0 | 28,5 | 25,5 | 22,9 |
| Кількість слів | 183,0 | 193,0 | 199,0 | 199,0 |
| Кількість символів | 1313,0 | 1411,0 | 1407,0 | 1393,0 |
| Слів/сек | 8,0 | 6,8 | 7,8 | 8,7 |
| Символів/сек | 57,1 | 49,5 | 55,2 | 60,8 |

Таблиця В.2 – Порівняльна таблиця швидкодії для DeepSeek-V3 (WEB):

|  |  |  |  |
| --- | --- | --- | --- |
| Назва великої мовної моделі | DeepSeek-V3 (WEB) | | |
| Номер Тесту | 1 | 2 | 3 |
| Час завантаження | 17,0 | 15,6 | 22,0 |
| Час відповіді | 34,0 | 26,6 | 31,8 |
| Загальний час | 51,0 | 42,2 | 53,8 |
| Кількість слів | 225,0 | 190,0 | 230,0 |
| Кількість символів | 1719,0 | 1401,0 | 1747,0 |

Таблиця В.3 – Порівняльна таблиця швидкодії для OpenAI: GPT-4-turbo (WEB):

|  |  |  |  |
| --- | --- | --- | --- |
| Назва великої мовної моделі | OpenAI: GPT-4-turbo (WEB) | | |
| Номер Тесту | 1 | 2 | 3 |
| Час завантаження | 5,6 | 5,1 | 5,3 |
| Час відповіді | 25,0 | 19,5 | 24,0 |
| Загальний час | 30,6 | 24,6 | 29,3 |
| Кількість слів | 190,0 | 192,0 | 197,0 |
| Кількість символів | 1438,0 | 1424,0 | 1489,0 |

Таблиця В.4 – Порівняльна таблиця швидкодії для мого проєкту (OpenAI gpt-4o, Desktop application):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Назва великої мовної моделі | Мій проєкт (OpenAI gpt-4o, Desktop application) | | | | | | | | | |
| Номер Тесту | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Час завантаження | 1,2 | 1,5 | 2,0 | 1,6 | 1,0 | 1,6 | 1,3 | 1,7 | 1,4 | 1,3 |
| Час відповіді | 15,3 | 10,5 | 25,7 | 12,9 | 17,8 | 10,5 | 11,3 | 12,9 | 22,2 | 19,9 |
| Загальний час | 16,5 | 11,9 | 27,7 | 14,5 | 18,8 | 12,0 | 12,5 | 14,6 | 23,6 | 21,2 |
| Кількість слів | 143 | 161 | 165 | 150 | 139 | 145 | 140 | 126 | 119 | 127 |
| Кількість символів | 1056 | 1122 | 1217 | 1093 | 1033 | 1057 | 967 | 927 | 862 | 912 |

Таблиця В.5 – Таблиця з підрахунком швидкодії кожного з аналізаторів за даними з таблиць В.1, В.2, В.3, В.4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва моделі | Anthropic AI: Claude 3.7 Sonnet (WEB) | DeepSeek-V3 (WEB) | OpenAI: GPT-4-turbo (WEB) | Мій проєкт (OpenAI gpt-4o, Desktop application) |
| Середній час завантаження документу | 1,8 | 18,2 | 5,3 | 1,5 |
| Середній час надання відповіді | 23,2 | 30,8 | 22,8 | 15,9 |
| Середній час відповіді з урахуванняя завантаження документу | 25,0 | 49,0 | 28,2 | 17,3 |
| Середня кількість слів на відповідь | 193,5 | 215,0 | 193,0 | 141,5 |
| Середня кількість символів на відповідь | 1381,0 | 1622,3 | 1450,3 | 1024,6 |
| Середня кількість слів за секунду | 7,7 | 4,4 | 6,9 | 8,2 |
| Середня кількість символів за секунду | 55,3 | 33,1 | 51,5 | 59,1 |

Дані, що наведені у таблицях В.1, В.2, В.3 та В.4, вимірювались за допомогою секундоміру. Початком відліку для заміру часу на завантаження документу слугувало натискання кнопки «Відкрити» діалогового вікна операційної системи Windows 10, а кінцем – передача користувачу права на відправку повідомлень. Початком відліку для заміру часу на відповідь асистенту слугувало надсилання запиту великій мовній моделі, а кінцем – отримання повної відповіді та передача користувачу права на відправку повідомлень. Заміри виконувались 23-го травня 2025 р. при кабельному підключенні до інтернету зі швидкістю 90-100 мбіт/сек.

Порівняльний аналіз показників із таблиці В.5 свідчить про виражену перевагу створеного мною десктопного застосунку на базі моделі GPT-4o над іншими сучасними рішеннями, що працюють через веб-інтерфейс. Середній час завантаження документу в моєму проєкті становить лише 1.5 секунди, що на 16.7% менше порівняно з Claude 3.7 (1.8 с), на 71.7% менше, ніж у GPT-4-turbo (5.3 с), і вражаючі 91.8% менше, ніж у DeepSeek-V3, де цей показник сягає 18.2 секунди.

Щодо часу генерації відповіді, мій застосунок також є лідером: середній показник становить 15.9 секунди. Це на 30.3% швидше за GPT-4-turbo (22.8 с), на 31.5% швидше, ніж у Claude 3.7 (23.2 с), і на 48.4% менше, ніж у DeepSeek-V3 (30.8 с). Якщо враховувати повний час взаємодії з документом (завантаження + відповідь), то мій проєкт справляється в середньому за 17.3 секунди, що на 38.7% менше за GPT-4-turbo (28.2 с), на 30.8% швидше за Claude 3.7 (25.0 с), і на 64.7% швидше, ніж DeepSeek-V3 (49.0 с).

Окремо варто виділити ефективність мовного генератора. Мій застосунок видає в середньому 8.2 слів на секунду, що на 6.5% більше, ніж у Claude 3.7 (7.7 сл/сек), на 18.8% більше, ніж у GPT-4-turbo (6.9 сл/сек) та на 86.4% швидше за DeepSeek-V3 (4.4 сл/сек). За аналогічним принципом, генерація символів у моєму проєкті відбувається зі швидкістю 59.1 символів за секунду, що на 6.9% швидше за GPT-4-turbo (51.5 симв/сек), на 6.9% швидше за Claude 3.7 (55.3 симв/сек), і на 78.5% перевищує показник DeepSeek-V3 (33.1 симв/сек).

Попри дещо менший обсяг відповідей (141.5 слова на відповідь проти 193–215 у конкурентів), проєкт забезпечує оптимальне співвідношення інформативності та швидкодії. У сукупності всі показники свідчать про високий ступінь оптимізації та доцільність впровадження мого проєкту в практичних сценаріях.