МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА З ІНТЕГРОВАНИМ ШІ-АГЕНТОМ НА ОСНОВІ MISTRAL

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи ЦДБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Технології цифрового дизайну»



(підпис)

Кімлача Вячеслава Віталійовича
(прізвище, ім'я, по батькові)

Науковий керівник



(підпис)

Омельчук Антон Анатолійович
к. т. н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми

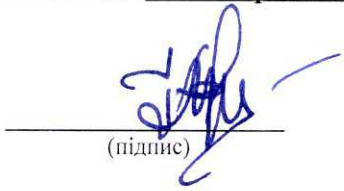


(підпис)

Ратушняк Тетяна Володимирівна, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри



(підпис)

Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 65 26.05.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 26.05.2025 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 10.06.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії: _____ М.М. Філоненко
(підпис)

_____ Т.В. Ратушняк
(підпис)

_____ В.О. Ніжегородцев
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	20		Омельчук А.А. Франчук В.М. Філоненко М.М. Ратушняк Т.В. Ніжегородцев В.О. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	7		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	3		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	35		

Оцінювання кваліфікаційної роботи науковим керівником – 35 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
70	C	4 (добре)	10.06.2025	Франчук В.М. Філоненко М.М. Ратушняк Т.В. Ніжегородцев В.О. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Технології цифрового дизайну»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«14» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Кімлачу Вячеславу Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА З ІНТЕГРОВАНИМ ШІ-АГЕНТОМ НА ОСНОВІ MISTRAL.
   Затверджена наказом ректора від «13» листопада 2024 № 1717
   Строк здачі закінченої роботи «12 » травня 2025

2. Мета роботи: розробка інформаційного вебінтерфейсу з чат-ботом на основі Mistral для обробки історичних запитів.
3. Орієнтовна база дослідження: Google Sites для вебінтерфейсу, Google Apps Script для серверної логіки, Mistral Large 2.1 API як мовна модель, Lighthouse для тестування, Chrome DevTools для відладки.
4. Вихідні дані до роботи: документація Mistral API, приклади з Google Apps Script, вимоги до інтеграції API в Google Sites, результати тестування через Lighthouse.
5. Перелік графічного матеріалу: скріншоти вебінтерфейсу, схема архітектури інтеграції, відповіді Mistral на запити, результати тестування Lighthouse, QR-код доступу.
6. Зміст кваліфікаційної роботи: ВСТУП — Актуальність теми, мета, завдання, методи дослідження, практична значущість. РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. 1.1. Використання ШІ в освітніх вебінтерфейсах 1.2. Огляд сучасних LLM: GPT, Gemini, Mistral, DeepSeek. 1.3. Обґрунтування вибору моделі Mistral Large 2.1. РОЗДІЛ 2. ІНТЕГРАЦІЯ МОДЕЛІ MISTRAL У ВЕБІНТЕРФЕЙС. 2.1. Технологічна платформа: Google Sites та Apps Script 2.2. Архітектура рішення та логіка обробки запитів. 2.3. Взаємодія з API Mistral через UrlFetchApp. РОЗДІЛ 3. ПРОЄКТНО-ПРИКЛАДНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ. 3.1. Опис інтерфейсу, реалізація HTML/JS 3.2. Приклади діалогів, аналіз відповідей моделі. 3.3. Тестування стабільності, продуктивності та безпеки. ВИСНОВКИ. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ. ДОДАТКИ — Повний код, фрагменти реалізації, скріншоти.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2024	До 30.10.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	25.12.2024	25.12.2024
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	15.02.2025	15.02.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	14.04.2025	14.04.2025
5	**Висновки**	01.05.2025	01.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	09.05.2025	09.05.2025
7	Отримання відгуку керівника	03.06.2025	03.06.2025
8	Отримання зовнішньої рецензії	05.06.2025	05.06.2025

8. Дата видачі завдання «14» листопада 2024

9. Керівник кваліфікаційної роботи _____ (підпис) Омельчук А.А (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Ратушняк Т.В. (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Кімлач В.В. (прізвище, ініціали)

АНОТАЦІЯ

Розробка вебінтерфейсу з інтегрованим ШІ-агентом на основі моделі Mistral.

Мета роботи – створення вебінтерфейсу з чат-ботом, що у форматі діалогу відповідає на історичні запитання, використовуючи можливості мовної моделі Mistral. Реалізація виконана на Google Sites із серверною частиною на Google Apps Script без окремого хостингу.

Завдання:

• аналіз технологій впровадження ШІ у вебінтерфейси;

• порівняння моделей нового покоління (GPT, Gemini, Mistral, DeepSeek);

• обґрунтування вибору Mistral для освітнього чат-бота;

• реалізація функціонального вебінтерфейсу;

• тестування системи на реальних запитаннях.

Предмет дослідження – інтеграція мовної моделі у вебінтерфейс.

Об'єкт – технології реалізації клієнт-серверної взаємодії з AI у вебсередовищі.

База – створений сайт із віджетом чат-бота.

Результатом є методика інтеграції LLM у Google Sites з оцінкою її ефективності для освіти.

Розробка може застосовуватись для освітніх вебресурсів і консультаційних сайтів.

Робота містить 54 сторінки тексту, 11 рисунків, 1 таблицю, 42 джерела, 3 додатки.

Ключові слова: Mistral, чат-бот, AI, Google Sites, Apps Script, API, вебінтерфейс, історія, діалог, LLM

Рік виконання: 2025

Рік захисту: 2025

ABSTRACT

Development of a Web Interface with an Integrated AI Agent Based on the Mistral Model

The aim of the thesis is to develop a web interface with an embedded chatbot capable of answering historical questions in a dialogue format using the Mistral large language model. The solution is implemented on Google Sites with server-side logic powered by Google Apps Script, eliminating the need for a dedicated backend.

The objectives include:

• analyzing technologies for integrating AI into web interfaces;

• comparing next-generation language models (GPT, Gemini, Mistral, DeepSeek);

• justifying the choice of Mistral for educational chatbot use;

• implementing a functional web interface with an AI agent;

• testing the system with real user queries and analyzing response accuracy.

The subject of the study is the integration of a language model into a web interface.

The object of the study is modern technologies for implementing client-server interaction with AI in a web environment.

The research is based on a developed website with an embedded chatbot widget.

The result is a methodology for integrating an open-source LLM into Google Sites and an assessment of its effectiveness for educational use.

The developed solution can be applied to interactive educational platforms, internal information systems of educational institutions, or consultation-based websites.

The qualification thesis contains 54 pages of main text, 11 figures, 1 table, 42 references, and 3 appendices.

Keywords: Mistral, chatbot, AI, Google Sites, Apps Script, API, web interface, history, dialogue, LLM

Year of completion: 2025

Year of defense: 2025

Зміст

ВСТУП 3
Розділ 1. Огляд технологій штучного інтелекту та платформи розробки вебінтерфейсу 6
1.1 Вебінтерфейси з підтримкою штучного інтелекту: сучасні тенденції 6
1.2 Мовні моделі нового покоління: GPT, Gemini, Mistral, DeepSeek 7
1.3 Вибір платформи для реалізації: Google Sites та Google Apps Script..... 12
Розділ 2. Розробка інтерфейсу користувача з інтегрованим ШІ-агентом на основі Mistral 14
2.1. Архітектура проєкту та загальна логіка роботи ШІ-системи 14
2.2. Інтеграція HTML і JavaScript у Google Sites 17
2.3. Реалізація функціоналу Google Apps Script 22
2.4. Логіка обробки запитів і формування відповіді 29
2.5. Забезпечення безпеки та обмеження доступу 33
Розділ 3. Тестування та можливості створенного сайту та вебінтерфейсу 39
3.1. Функціональні можливості сайту 39
3.2. Приклади взаємодії користувача з ботом 43
3.3. Оцінка ефективності та перспективи вдосконалення..... 46
3.4. Тестування та оптимізація сайту..... 49
ВИСНОВКИ..... 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 61
ДОДАТКИ..... 67

ВСТУП

Сучасні тенденції розвитку інформаційних технологій свідчать про стрімке впровадження систем штучного інтелекту (ШІ) в освітній процес та у вебінтерфейси різного призначення. Генеративні мовні моделі (Large Language Models, LLM), такі як GPT-4 та інші, продемонстрували здатність проводити змістовні діалоги і надавати корисні відомості у відповідь на запити користувачів. Протягом 2023–2024 років спостерігається безпрецедентне зростання використання ШІ-інструментів у навчанні: за даними звітів, понад половину викладачів та переважна більшість студентів тепер регулярно використовують штучний інтелект у навчальному процесі.[1] Це підтверджує актуальність розвитку інтерактивних освітніх ресурсів, що використовують ШІ для підвищення залученості і персоналізації навчання. В історичній освіті з’являються інноваційні рішення, коли учні можуть «спілкуватися» з історичними постатями через чат-боти, що оживляє минуле та стимулює інтерес до вивчення історії.[2] Таким чином, створення вебінтерфейсу з інтегрованим ШІ-агентом, здатним у форматі діалогу відповідати на питання з історії міжвоєнного періоду та Другої світової війни, є своєчасним і суспільно значущим завданням.

Метою роботи є розробка інформаційного вебінтерфейсу із чат-ботом на основі мовної моделі Mistral Large 2.1, який здатен відповідати на історичні питання міжвоєнного періоду та Другої світової війни у форматі діалогу. Для досягнення поставленої мети передбачено вирішення таких завдань:

- проаналізувати сучасні тенденції впровадження штучного інтелекту у вебінтерфейси та освітні онлайн-системи;
- дослідити новітні великі мовні моделі (GPT, Gemini, Mistral, DeepSeek) та порівняти їх характеристики, обґрунтувавши вибір моделі Mistral для реалізації чат-бота;

- обрати платформу для створення вебінтерфейсу (розглянути можливості Google Sites та Google Apps Script) та обґрунтувати доцільність її використання;
- розробити прототип вебінтерфейсу з інтегрованим ШІ-агентом на базі вибраної мовної моделі і платформи, реалізувавши функціонал діалогової взаємодії з користувачем;
- протестувати роботу створеного чат-бота на запитаннях з історії міжвоєнного періоду і Другої світової війни, оцінити якість відповідей та відповідність очікуванням цільової аудиторії;
- узагальнити результати та визначити напрями подальшого удосконалення розробленого рішення.

Об’єктом дослідження є інтерактивні інформаційні вебінтерфейси з підтримкою штучного інтелекту в освітній сфері. Предметом дослідження є методи та засоби розробки вебінтерфейсу з інтегрованим мовним ШІ-агентом (на прикладі моделі Mistral) для надання відповідей на історичні запитання у діалоговому режимі.

Для досягнення мети використано комплекс загальнонаукових і спеціальних методів. Зокрема, було здійснено аналіз науково-технічної літератури і огляд сучасних вебрішень (для виявлення тенденцій та існуючих аналогів), застосовано порівняльний аналіз характеристик мовних моделей (для обґрунтування вибору моделі Mistral), методи проєктування програмних систем (для розробки архітектури інтерфейсу та інтеграції чат-бота), а також експериментальний метод (тестування працездатності та точності відповідей чат-бота на спеціально підібраних історичних запитаннях).

Практична значущість отриманих результатів. Розроблений у межах роботи вебінтерфейс з ШІ-агентом має практичну цінність як прототип інтерактивного історичного ресурсу, що може бути впроваджений у діяльність освітніх закладів. Він дозволяє учням та викладачам у зручній формі

отримувати відповіді на питання з історії, поглиблюючи знання про міжвоєнний період та період Другої світової війни. Такий чат-бот може виступати в ролі «віртуального консультанта» або помічника в навчальному процесі, підсилюючи самостійну пізнавальну діяльність учнів. Результати роботи можуть бути використані як основа для створення інших тематичних інтерактивних освітніх ресурсів, а також як демонстрація можливостей інтеграції відкритих мовних моделей (таких як Mistral) у вебзастосунки з мінімальними витратами.

Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт, предмет, методи дослідження та висвітлено практичну значущість роботи. У першому розділі проведено аналіз сучасних тенденцій розвитку вебінтерфейсів з підтримкою ШІ, розглянуто новітні мовні моделі (GPT, Gemini, Mistral, DeepSeek) та обґрунтовано вибір моделі Mistral для реалізації проєкту, а також виконано огляд платформ розробки вебінтерфейсів (Google Sites, Google Apps Script) з обґрунтуванням вибору платформи. Другий розділ присвячено проєктуванню та реалізації вебінтерфейсу з інтегрованим ШІ-агентом: описано архітектуру системи, інтеграцію мовної моделі Mistral через прикладний інтерфейс (API), а також наведено ключові фрагменти програмної реалізації. У третьому розділі описано тестування і результати експериментальної експлуатації створеного чат-бота: проаналізовано приклади діалогів з користувачами, оцінено точність і доречність відповідей ШІ-агента на історичні питання, а також розглянуто питання безпеки та етичності використання такого інструменту. У висновках узагальнено результати дослідження, сформульовано основні висновки та рекомендації щодо подальшого розвитку системи.

Розділ 1. Огляд технологій штучного інтелекту та платформи розробки вебінтерфейсу

1.1 Вебінтерфейси з підтримкою штучного інтелекту: сучасні тенденції

У сучасному вебпросторі дедалі більшого поширення набувають інтерактивні системи, що інтегрують штучний інтелект безпосередньо у вебінтерфейс. Це означає, що користувачі на вебсайтах можуть ставити запитання або вести діалог із вбудованим AI-асистентом і одразу отримувати згенеровані відповіді, замість того щоб шукати інформацію традиційним способом. Такий підхід уже активно використовується в сфері онлайн-сервісів підтримки клієнтів і консультативних систем: прогнозується, що інтерактивні AI-чатботи стануть новою нормою у вебсервісах завдяки покращенню архітектури та дизайну чатботів.[2] На рис. 1.1 наведено динаміку глобальних інвестицій у технології штучного інтелекту в період з 2019 по 2024 рік, що підкреслює значний зріст фінансових вкладень у розвиток цієї галузі та демонструє її перспективність. З точки зору зручності для користувача, вбудований ШІ-агент на сайті забезпечує миттєвий доступ до необхідних відомостей у форматі розмови, що підвищує залученість і швидкість отримання інформації.

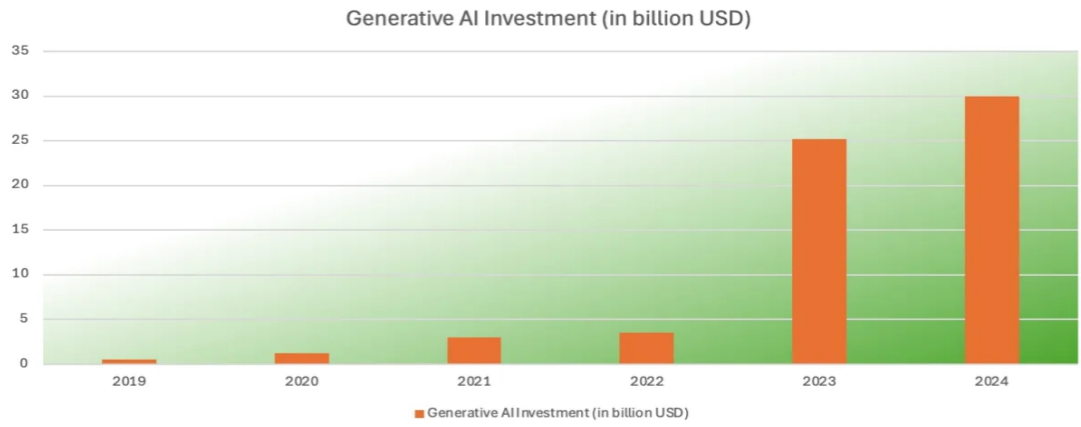


Рис. 1.1 – Динаміка глобальних інвестицій у технології штучного інтелекту за 2018–2024 рр.

Особливо перспективним є використання таких технологій в освітніх та інформаційно-просвітницьких ресурсах. На зміну статичним текстам і традиційним пошуковим системам приходять діалогові системи навчання, які дозволяють ставити запитання природною мовою і отримувати розгорнуті пояснення або довідкові матеріали. Прикладом є новітні платформи, що дають можливість спілкуватися з історичними постатями через AI-чат: зокрема, англомовний сервіс YourStory Education дозволяє учням вести діалоги від першої особи з видатними діячами минулого, перетворюючи вивчення історії на динамічний інтерактивний процес.[2] За допомогою подібних рішень навчання стає більш захопливим – учні можуть, наприклад, запитати Бенджаміна Франкліна про його відкриття чи Генрі Форда про винахід конвеєра, і отримати відповіді від імені цих персонажів, засновані на реальних історичних фактах.[2] Освітні експерти відзначають, що такі підходи сприяють персоналізації навчального досвіду та підвищують мотивацію учнів до вивчення матеріалу.[2]

Відтак, інтеграція ШІ-агентів у вебінтерфейси є актуальною тенденцією, що змінює підходи до подачі інформації та навчання. Для успішної реалізації подібних систем необхідно обрати потужну мовну модель, здатну генерувати точні й зрозумілі відповіді на запити з відповідної тематичної галузі (в даному випадку – історія 1918–1945 років), а також забезпечити зручну платформу для доставки цієї функціональності кінцевим користувачам через веб.

1.2 Мовні моделі нового покоління: GPT, Gemini, Mistral, DeepSeek

Для реалізації функцій інтелектуального чат-бота основним технологічним компонентом є велика мовна модель (LLM – Large Language Model), яка виконує розуміння запитання та генерацію відповіді. За останні роки з’явилося декілька поколінь таких моделей, і наразі доступні як комерційні закриті рішення, так і відкриті моделі від нових дослідницьких

команд. Розглянемо ключових представників нового покоління LLM, акцентуючи увагу на їх можливостях та придатності для нашого проєкту.

GPT-4 (OpenAI) — одна з найпотужніших генеративних моделей, здатна виконувати творчі та технічні завдання. Вона є пропрієтарною, доступною лише через платний API або інтегровані продукти. Висока вартість навчання ($100 млн) та обмежена гнучкість у розгортанні[4].

Gemini (Google DeepMind) — мультимодальна модель, що працює з текстом, зображеннями, аудіо та кодом. Інтегрована в сервіси Google і доступна через Vertex AI, але закрита для зовнішнього розгортання. Відзначається логічними та стратегічними можливостями[5][6][7].

Mistral Large 2.1 (Mistral AI) — відкрита модель із 123 млрд параметрів, доступна для дослідників під спеціальною ліцензією. Відрізняється високою продуктивністю, порівнянною з GPT-4 та Claude 3, з підтримкою багатьох мов і можливістю розгортання на власній інфраструктурі[9][10].

DeepSeek (DeepSeek AI) — відкрита модель з ліцензією MIT, оптимізована для високої продуктивності при відносно низьких витратах на навчання ($6 млн). Демонструє результати, не гірші за GPT-4 у медичній діагностиці та інших завданнях[4][11].

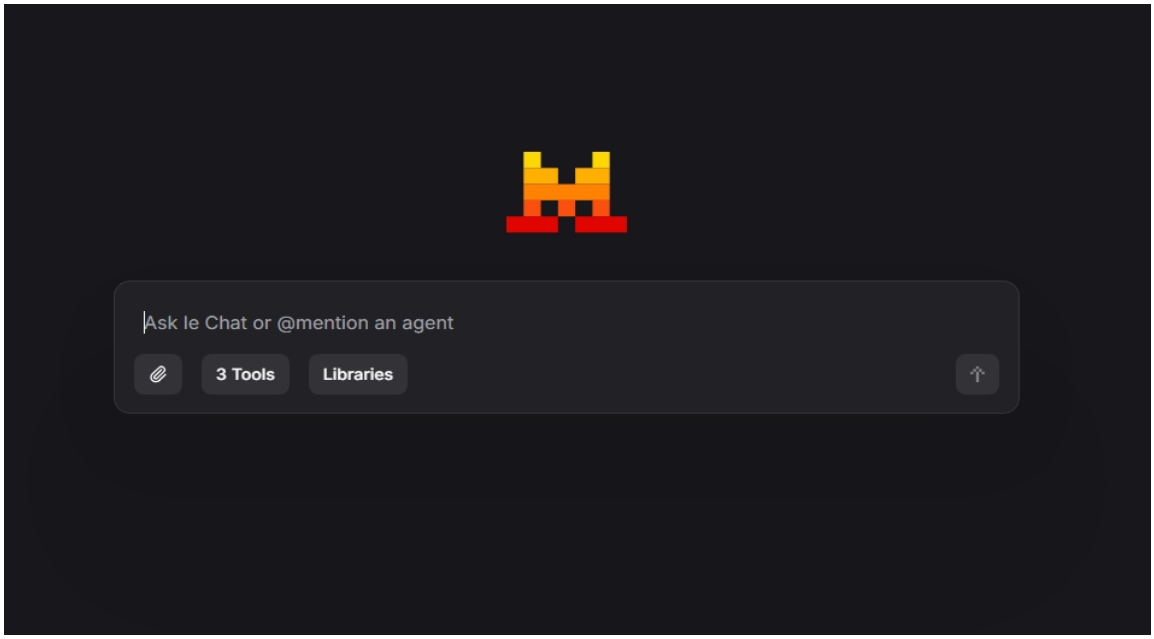


Рис. 1.2 – Поле для запитів в Mistal AI.

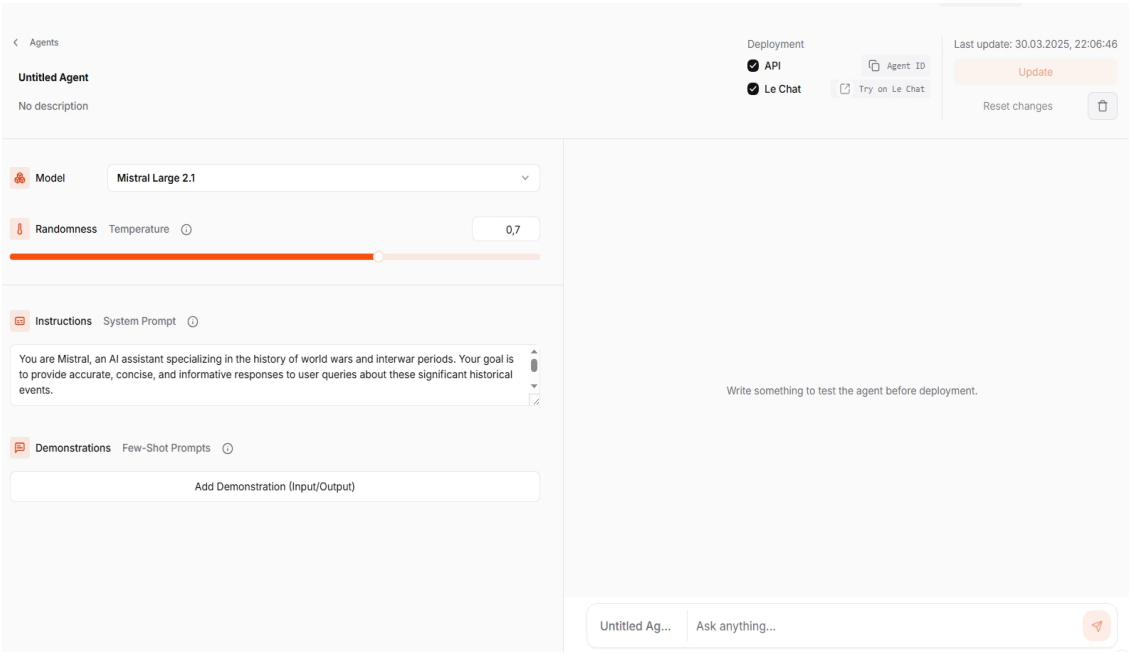


Рис. 1.3 – Налаштування версії та промптів для коректної роботи AI.

DeepSeek (DeepSeek AI). DeepSeek – ще один представник відкритих LLM, створений дослідницькою командою з Китаю. Компанія DeepSeek презентувала серію моделей з різною кількістю параметрів (7B, 67B тощо), які позиціюються як “open-weight” – тобто відкриті для розгортання, хоча і з певними умовами використання[4]. Модель DeepSeek-R1, випущена на

початку 2025 року, отримала ліцензію MIT і за повідомленнями забезпечує якість відповідей на рівні GPT-4[4]. При цьому її розробники досягли такої продуктивності значно меншими ресурсами: тренування DeepSeek-V3 обійшлося приблизно у $6 млн, що на порядок менше за витрати на GPT-4[4]. У галузевих бенчмарках відкриті моделі DeepSeek продемонстрували результати, не гірші, а подекуди кращі за закриті аналоги (наприклад, в завданнях медичної діагностики вони зрівнялися з GPT-4 і Gemini 2.0)[11]. Цей успіх підтверджує перспективність відкритого підходу, де спільнота може перевершувати великі корпорації завдяки ефективності та доступності технологій. Для наочності наведемо порівняння розглянутих моделей у таблиці 1.

Таблиця 1.1 – Порівняльна характеристика мовних моделей нового покоління

Модель	**Розробник**	**Обсяг моделі**	**Доступність**	**Особливості**
GPT-4 (OpenAI)	OpenAI, США	*закрито* (≈170B+)	Закрита (API, платний)	Висока якість, сильне логічне мислення; комерційна модель з обмеженим доступом.
Gemini (Google)	Google DeepMind, США	*невідомо* (мультимодальна)	Закрита (Google Cloud)	Підтримка тексту та зображень; інтеграція з сервісами Google; найсучасніші алгоритми DeepMind.
Mistral Large 2	Mistral AI, ЄС	123 млрд параметрів	Відкрита (API, дослідницька ліцензія)	128k контекст; багатомовність; продуктивність на рівні GPT-4, можливість власного розгортання.[2]
DeepSeek R1	DeepSeek, Китай	67 млрд параметрів	Відкрита (MIT ліцензія)	Відкриті ваги моделі; відповіді якістю як GPT-4 при менших ресурсах навчання;

Модель	Розробник	Обсяг моделі	Доступність	Особливості
				можна локально донавчати.[11]

Як видно з таблиці 1.1, відкриті моделі (Mistral, DeepSeek) є привабливими для академічних та некомерційних проєктів, оскільки забезпечують високу якість роботи, але не накладають жорстких обмежень на використання. У рамках даного дослідження обрано модель Mistral Large 2.1. Основними аргументами на її користь є: по-перше, відкритість і можливість безкоштовного використання для дослідницьких цілей; по-друге, сильна підтримка різних мов (що важливо з огляду на потенційну двомовність інтерфейсу – українська та англійська); по-третє, розширені можливості моделі в плані великого контексту і зменшеної схильності до фактичних помилок (галюцинацій) завдяки спеціальному налаштуванню[9]. Mistral виступає компромісом між продуктивністю та доступністю: вона наближається за якістю до провідних закритих моделей, але дозволяє інтеграцію в власне програмне рішення без значних фінансових витрат. Це робить її оптимальним вибором для створення історичного чат-бота в умовах обмежених ресурсів студентського проєкту.

1.3 Вибір платформи для реалізації: Google Sites та Google Apps Script

Для розробки вебінтерфейсу з інтегрованим ШІ-агентом обрано зв’язку Google Sites та Google Apps Script, яка забезпечує простоту розгортання і інтеграції чат-бота без потреби в окремому сервері. Google Sites дозволяє швидко створювати вебсторінки через зручний інтерфейс без програмування HTML/CSS, а Google Apps Script забезпечує серверну логіку, виконуючи код у хмарі та здійснюючи запити до зовнішніх API, таких як Mistral. Це рішення мінімізує витрати на розробку, спрощує масштабування і є безкоштовним у межах лімітів Google Workspace.

Apps Script надає можливість розгортання вебзастосунку з унікальною URL-адресою, який можна вбудувати у Google Sites через iframe. Вебінтерфейс взаємодіє з користувачем, надсилаючи запити до мовної моделі Mistral та отримуючи відповіді, які відображаються на сторінці в режимі реального часу. Це дозволяє інтегрувати чат-бота у сторінки сайту без потреби налаштовувати складну серверну інфраструктуру.

Платформа Google Sites забезпечує швидке прототипування та легке оновлення контенту, що є важливим для навчальних проєктів. Масштабованість під аудиторію закладів освіти, відсутність потреби у хостингу та інтеграція з екосистемою Google роблять цю зв’язку ідеальним рішенням для реалізації історичного чат-бота з мінімальними витратами та високою доступністю.

Розділ 2. Розробка інтерфейсу користувача з інтегрованим ШІ-агентом на основі Mistral

У цьому розділі представлено технічну реалізацію вебінтерфейсу чат-бота з інтегрованим штучним інтелектом (ШІ) на базі моделі Mistral Large 2.1. На відміну від теоретичних аспектів, розглянутих у попередньому розділі, тут детально описано архітектуру системи, компоненти інтерфейсу користувача і серверної логіки, механізм обробки діалогових запитів користувача та заходи забезпечення безпеки. Рішення реалізовано засобами хмарних сервісів Google (Google Sites і Google Apps Script) без потреби у виділеному сервері – вся логіка знаходиться у хмарі. Код програми (фрагменти HTML/JS інтерфейсу та скрипти Apps Script) винесено до Додатків, а в тексті розділу розглядаються ключові фрагменти та їх призначення.

2.1. Архітектура проєкту та загальна логіка роботи ШІ-системи

Проєкт має багаторівневу архітектуру, що складається з клієнтської частини (вебінтерфейс Google Sites), серверної частини (скрипт Google Apps Script) та зовнішнього API з моделлю Mistral. Клієнтська частина відповідає за відображення інтерфейсу діалогу та відправлення запитів, серверна – за прийом запитів і виклик моделі ШІ, а служба Mistral API – за генерацію відповіді на основі запиту. Взаємодія між цими компонентами відбувається послідовно у форматі запит–відповідь. Схематично обмін даними між компонентами зображено на рис. 2.1.

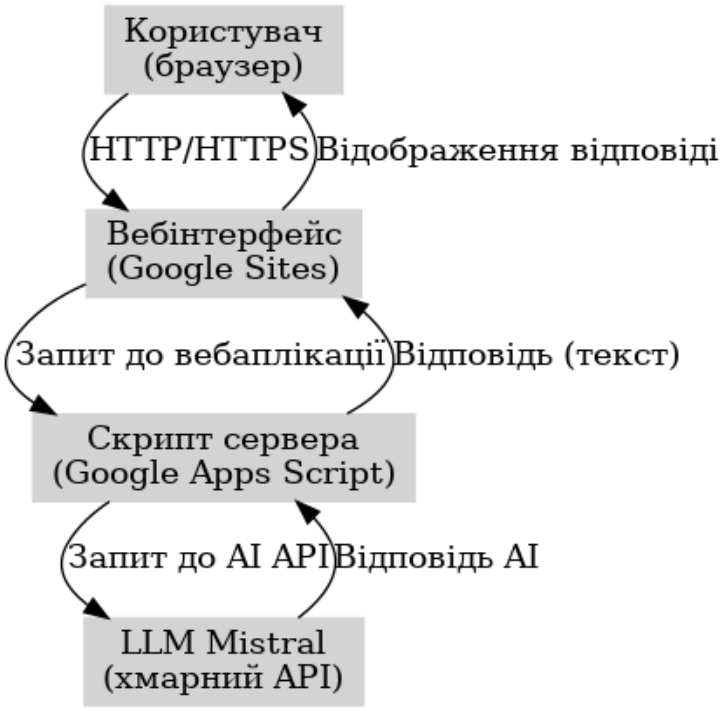


Рис. 2.1. Схема взаємодії між користувачем, вебінтерфейсом Google Sites, серверним скриптом Google Apps Script та зовнішнім API Mistral.

Як показано на рис. 2.1, цикл взаємодії починається з того, що користувач у браузері вводить питання історичної тематики у поле інтерфейсу (крок 1). Після натискання кнопки запит відправляється з клієнтського скрипта (JavaScript на сторінці Google Sites) до вебслужби Google Apps Script (крок 2). Google Apps Script виступає посередником: отримавши HTTP-запит, він викликає зовнішній Mistral API з переданим текстом питання (крок 3). Mistral Large 2.1 – це велика мовна модель нового покоління від компанії Mistral AI, що відзначається підвищеними можливостями в генерації тексту, коду, логічному висновку і підтримує багато мов[15][16]. Mistral API аналізує запитання і генерує відповідь у діалоговому форматі (імітує відповідь «асистента» на запит «користувача»). Згенерована моделью відповідь повертається з хмарного сервісу Mistral назад до скрипту (крок 4). Далі Apps Script обробляє отримані дані, формує власну HTTP-відповідь і надсилає її клієнтському інтерфейсу (крок 5). Нарешті, браузер отримує цю відповідь і відображає її користувачу на вебсторінці у вигляді продовження діалогу (крок

6). Таким чином, забезпечується діалоговий режим роботи: користувач задає питання, а система відповідає в інтерактивному форматі.

Важливою особливістю реалізованої системи є те, що вона побудована повністю на хмарних платформах Google і не потребує окремого серверного хостингу. Вебсторінка інтерфейсу створена за допомогою Google Sites, що надає простий спосіб розміщення статичного контенту і дозволяє вбудовувати користувацький HTML/JavaScript. У свою чергу, Google Apps Script виконує роль серверного середовища для написання коду на JavaScript, який виконується на стороні сервера (в інфраструктурі Google). Apps Script дозволяє опублікувати скрипт як вебзастосунок, тобто зробити його доступним через URL для обробки HTTP-запитів[17]. Для цього скрипт містить функцію doGet(e), яка автоматично викликається при надходженні GET-запиту на відповідну URL-адресу і може повертати відповідь у вигляді HTML-сторінки або текстових даних[17]. У даному проєкті Apps Script-вебслужба працює у режимі API: вона отримує від клієнта текст запиту та повертає згенеровану відповідь у форматі даних (JSON). Mistral Large 2.1 доступний через власний веб API, що передбачає надсилання HTTP POST запитів з параметрами моделі і текстом діалогу, аутентифікацію за API-ключем та отримання відповіді у форматі JSON. Тому Google Apps Script виступає “посередником” між фронтендом і ШІ-сервісом: він формує запит до Mistral API у потрібному форматі, додає ключ доступу та отримує результат для передачі користувачу.

Отже, загальна логіка роботи системи виглядає так: вебінтерфейс збирає дані від користувача і відправляє на серверний скрипт, скрипт перенаправляє запит до моделі Mistral та отримує від неї згенеровану відповідь, після чого інтерфейс відображає цю відповідь користувачеві. Така архітектура забезпечує розподіл обчислень: ресурсоємне генерування відповіді виконується на стороні моделі Mistral (в хмарі), тоді як клієнтська частина залишається легкою і швидкою. Крім того, використання Google Sites і Apps

Script означає, що вся інфраструктура розгорнута у хмарі Google – розробнику не потрібно піднімати власний сервер або хостинг, достатньо опублікувати скрипт і сайт відповідно. Це спрощує підтримку і масштабування рішення, адже Google забезпечує виконання скрипта та мережевий доступ до нього.

2.2. Інтеграція HTML і JavaScript у Google Sites

Інтерфейс користувача чат-бота реалізовано у середовищі Google Sites шляхом вбудовування спеціального HTML-коду з елементами форми та скриптами (див. Рис. 2.2). Новий Google Sites не передбачає прямого редагування HTML усієї сторінки, але надає компонент Embed («Вбудувати»), за допомогою якого можна вставити користувацький код або віджет. У межах проєкту було використано цей механізм для додання на сторінку наступних елементів інтерфейсу діалогу:

- Текстове поле вводу запитання (HTML-елемент <input type="text"> або <textarea>). Користувач вводить у ньому своє питання щодо історії (наприклад, подій міжвоєнного періоду чи Другої світової війни). Поле має унікальний ідентифікатор (наприклад, id="userQuestion") для подальшого доступу через скрипт.
- Кнопка відправлення запиту (елемент <button>). На кнопці розміщено напис, наприклад, «Запитати» або іконка «Send». При натисканні цієї кнопки спрацьовує сценарій, що зчитує введений текст запитання і запускає процес отримання відповіді. Кнопка теж має власний id (наприклад, id="askButton"), щоб до неї можна було прив’язати обробник події.
- Блок для відображення відповіді. Це може бути простий контейнер , внутрішній текст якого оновлюється скриптом після отримання відповіді від сервера. Спочатку цей блок порожній або містить текст на кшталт «Відповідь з’явиться тут». Після обробки запиту вміст змінюється на текст відповіді ШІ-агента.

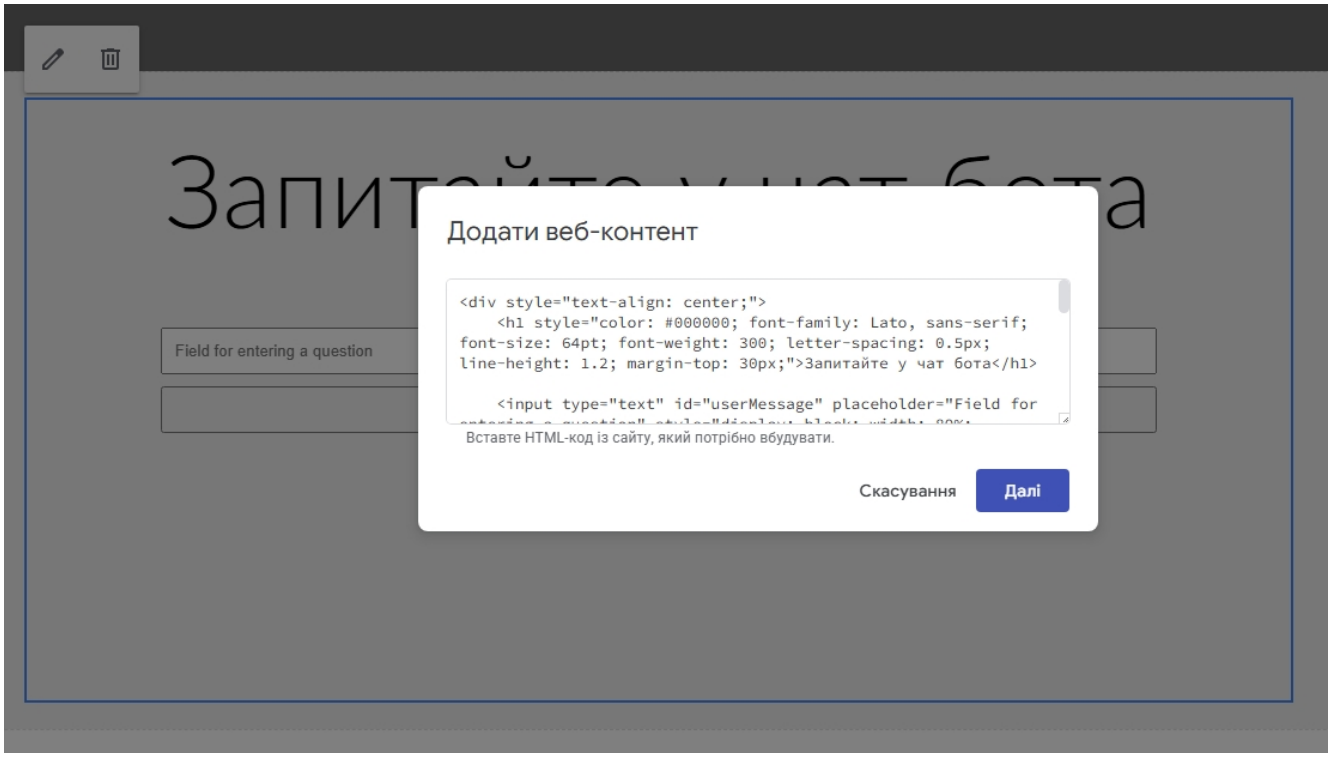


Рис. 2.2 – Вікно додавання коду на Google Sites

Усі ці елементи розташовано на сторінці Sites у логічному порядку, формуючи мінічат: поле вводу і кнопка для відправки нового повідомлення зверху, а під ними область з відповіддю (і, за потреби, з історією запитань-відповідей). Для візуального оформлення можна застосувати мінімальні стилі (CSS), надані конструктором Sites або inline-стилями в коді: наприклад, задати ширину поля вводу, відступи, рамки, фон блоку відповіді тощо, щоб інтерфейс нагадував звичні чат-застосунки.

Найважливішою частиною інтеграції є підключення JavaScript-коду на сторінці, який здійснює взаємодію з сервером. У Google Sites можна вставити блок <script> всередині вбудованого коду. В даному проекті скрипт реалізує логіку зчитування введеного питання і виконання запиту до Apps Script. Замість традиційного відправлення HTML-форм (яке могло б перезавантажувати сторінку) використано сучасний підхід AJAX: виклик Fetch API з боку клієнта. Це дозволяє відправити запит асинхронно і отримати

відповідь без перезавантаження сторінки, що важливо для інтерактивності чат-бота.

Нижче наведено спрощений приклад клієнтського JavaScript-коду, вбудованого в сторінку (Listing 2.1). Він додає обробник на кнопку відправки і визначає послідовність дій після її натискання:

```
<!-- Fragment of embedded HTML in Google Site -->
<input type="text" id="userQuestion" placeholder="Введіть ваше запитання...">
<button id="askButton">Надіслати</button>
<div id="answer"></div>

<script>
  // Додаємо обробник події на кнопку "Надіслати"
  document.getElementById('askButton').addEventListener('click', function() {
    // Зчитуємо текст питання із поля вводу
    const question = document.getElementById('userQuestion').value;
    // Відправляємо запит до вебслужби Apps Script (GET-запит з параметром)
    fetch('https://script.google.com/macros/s/DEPLOY_ID/exec?prompt=' + encodeURIComponent(question))
      .then(response => response.json())              // очікуємо JSON відповідь
      .then(data => {
        // Отримуємо поле "answer" з об'єкта відповіді і вставляємо в HTML
        document.getElementById('answer').innerText = data.answer;
      })
      .catch(error => console.error('Помилка запиту до AI:', error));
  });
</script>
```

Listing 2.1. Фрагмент HTML/JavaScript, вбудований у сторінку Google Sites для відправки запиту і відображення відповіді.

У цьому коді спочатку визначено елементи інтерфейсу: текстове поле для питання (id="userQuestion"), кнопка (id="askButton") і блок для відповіді (id="answer"). Потім у блоці <script> за допомогою document.getElementById отримуються посилання на ці елементи. На кнопку додається слухач події click: коли користувач натискає кнопку, виконується функція, що зчитує текст запитання. Значення question конструюється у рядок запиту HTTP. Далі викликається функція fetch() – це вбудований вебAPI для виконання HTTP-запитів з браузера. В якості аргументу їй передається URL опублікованого веб застосунку Google Apps Script. Ця URL містить унікальний ідентифікатор деплойменту скрипту (позначено як DEPLOY_ID) і шлях /exec для виконання останньої опублікованої версії. До URL додається параметр запиту prompt, значення якого дорівнює тексту питання (попередньо кодованому через encodeURIComponent для коректної передачі спеціальних символів).

Використано метод HTTP GET для спрощення (передача короткого текстового параметра у URL). Альтернативно можна було б застосувати POST і передавати дані в тілі запиту; у цьому проекті достатньо GET-запиту, оскільки питання користувача зазвичай короткі. Функція fetch() повертає проміс, тому далі метод .then() використовується для обробки результату асинхронно. Очікується, що Apps Script поверне JSON-дані, тому викликається response.json(), що знову-таки повертає проміс з об’єктом даних. У наступному .then() цей об’єкт (позначено як data) вже доступний; скрипт отримує з нього поле answer (тобто згенеровану текстову відповідь) і присвоює його як внутрішній текст елементу з id="answer". В результаті текст відповіді з’являється на сторінці під полем запитання. Нарешті, блок .catch() призначений для обробки можливих помилок – якщо мережевий запит не вдався (наприклад, немає з’єднання або сервер повернув помилку), в консоль браузера виведеться повідомлення про помилку.

Таким чином, HTML+JS код, вбудований у Google Sites, здійснює роль фронтенду: він надає користувачу інтерфейс для введення питання і одержання відповіді, а також виконує асинхронний запит до бекенду (Apps Script). Оскільки Sites не допускає запуску власного серверного коду, використання fetch до стороннього сервісу є оптимальним рішенням. Необхідно відзначити, що при прямому виклику зовнішнього домену з JavaScript можуть виникнути питання безпеки (політика CORS). В даному випадку роль стороннього сервісу виконує скрипт, розгорнутий на домені Google (script.google.com), і оскільки запит є GET і Apps Script повертає JSON через ContentService, браузер успішно отримує відповідь. При потребі більш гнучкого налаштування (наприклад, якщо б використовувався POST), можна було б налаштувати Apps Script на повернення JSONP або встановлення заголовка Access-Control-Allow-Origin: *. У нашій реалізації достатньо стандартних можливостей, щоб зв’язати сторінку Sites з Apps Script через запит fetch.

2.3. Реалізація функціоналу Google Apps Script

Серверна частина системи реалізована на платформі Google Apps Script як незалежний скрипт (standalone script), який був опублікований як вебзастосунок. Для публікації проєкту як вебзастосунку було використано стандартний механізм Google Apps Script (див. рис. 2.3).

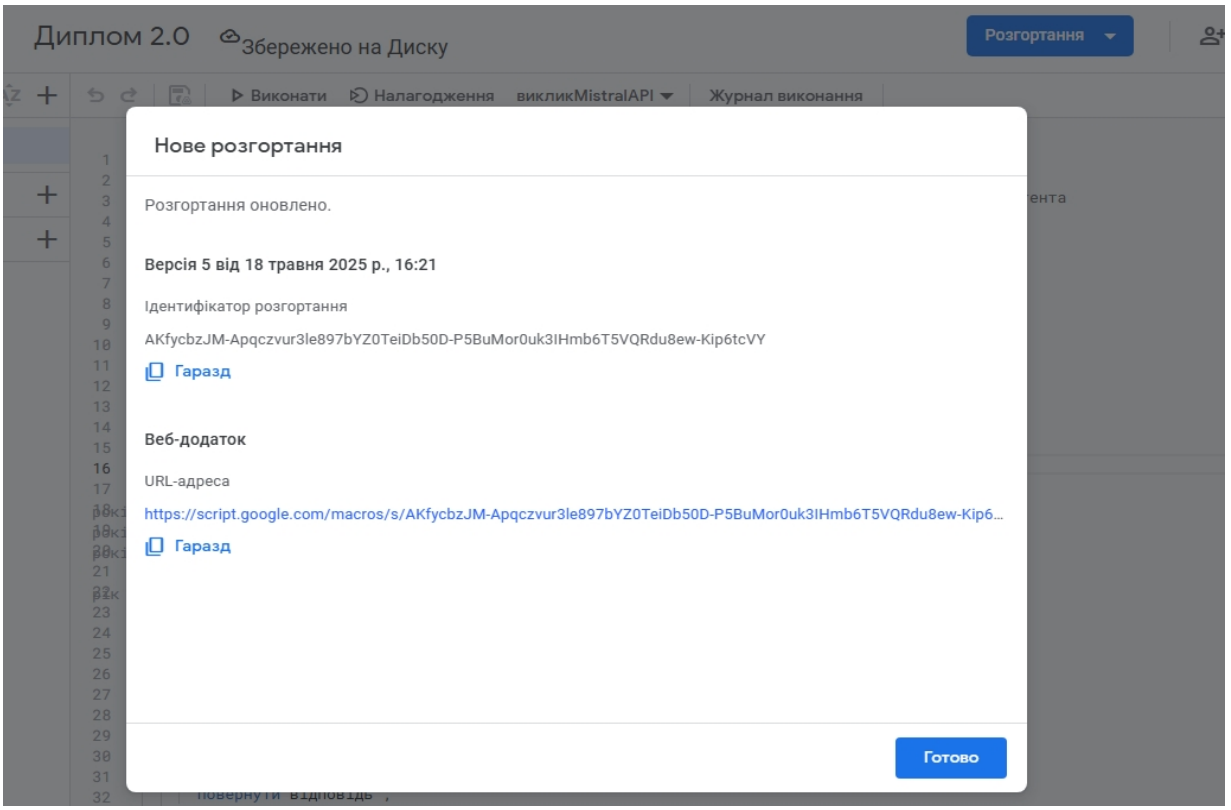


Рис. 2.3 – Вікно публікації (розгортання) вебзастосунку у Google Apps Script.

Після успішного розгортання система отримує публічний URL-адресу, за якою здійснюються HTTP-запити.Основними компонентами бекенд-логіки є функція-обробник HTTP-запиту doGet(e) та допоміжна функція callMistralAPI(prompt), що здійснює звернення до API моделі Mistral. Повний вихідний код серверного скрипта наведено в додатку (листинг Apps Script), а нижче розглядаються основні фрагменти коду з поясненнями.

Функція doGet(e)

Функція doGet(e) викликається автоматично щоразу, коли на URL вебзастосунку надходить HTTP GET запит[19]. Платформа Apps Script формує об’єкт параметрів e із деталями запиту, зокрема e.parameter містить передані параметри рядка запиту. У нашому випадку клієнтський запит передає параметр prompt (текст питання). Завдання doGet – отримати цей текст, передати його до API Mistral для генерації відповіді та повернути результат у зручному для клієнта форматі (JSON). Нижче наведено спрощений код цієї функції (Listing 2.2):

```
function doGet(e) {
  // Отримання тексту питання з параметрів запиту
  var userPrompt = e.parameter.prompt || '';
  // Виклик допоміжної функції, що звертається до Mistral
  var answer = callMistralAPI(userPrompt);
  // Формування JSON-об’єкта відповіді
  var output = { "answer": answer };
  // Повернення відповіді у форматі JSON
  return ContentService.createTextOutput(JSON.stringify(output))
                       .setMimeType(ContentService.MimeType.JSON);
}
```

Listing 2.2. Функція-обробник doGet(e) у Apps Script, що приймає параметр запиту та повертає JSON з відповіддю.

На початку функції з об’єкта e.parameter зчитується значення параметра prompt – саме це поле передає клієнтський інтерфейс при виклику fetch. Значення зберігається у змінній userPrompt. Для надійності використано оператор || '', щоб в разі відсутності параметра змінна прийняла порожній рядок (це дозволить уникнути помилки при порожньому запиті). Далі викликається

функція callMistralAPI(userPrompt), яка виконує основну роботу з ШІ (логіка цієї функції розглядається нижче). Передбачається, що callMistralAPI повертає рядок з відповіддю на питання користувача. Отримане значення поміщається у змінну answer.

Після цього формується об’єкт output, що містить одне поле "answer" з текстом відповіді. Об’єкт перетворюється у формат JSON за допомогою JSON.stringify(output). Потім застосовується сервіс ContentService для генерації HTTP-відповіді: методом createTextOutput створюється текстове представлення JSON, і викликається .setMimeType(ContentService.MimeType.JSON) для встановлення MIME-типу application/json. В результаті Apps Script повертає клієнту HTTP-відповідь з заголовками, що вказують на формат JSON, і тілом – текстом JSON-об’єкта, наприклад:

{"answer": "Текст згенерованої відповіді"}

На стороні браузера цей JSON буде автоматично розпізнаний методом response.json() (див. підрозділ 2.2) і перетворений у JavaScript-об’єкт data з полем answer. Таким чином, doGet виступає «мостом» між HTTP-запитом від фронтенду та функцією ШІ: вона отримує запитання і повертає відповідь у зручному для фронтенду вигляді.

Варто зазначити, що в даному випадку Apps Script не генерує повноцінну HTML-сторінку, а лише надсилає дані. Це відрізняється від сценарію, коли Apps Script використовується для побудови інтерфейсу (HTML Service). Тут ми задіяли Apps Script як вебслужбу (API), що спрощує інтеграцію з Google Sites через fetch.

Функція callMistralAPI(prompt)

Функція callMistralAPI реалізує безпосереднє звернення до зовнішнього API Mistral з метою отримання згенерованої відповіді на заданий текст запиту. Вона інкапсулює усі технічні деталі роботи з HTTP-запитом до сервера Mistral: формування потрібного формату повідомлення, встановлення заголовків авторизації, відправлення запиту і обробка відповіді. Псевдокод (спрощений) цієї функції наведено в Listing 2.3:

```
function callMistralAPI(userPrompt) {
  1. Підготовка даних для запиту
  var apiKey = 'XXXXXXXXXXXX';  // API-ключ для доступу до Mistral
  var url = 'https://api.mistral.ai/v1/chat/completions';
  // Формуємо payload у форматі JSON відповідно до вимог Mistral API
  var payload = {
    "model": "mistral-large-2.1",              // ID моделі Mistral
    "messages": [
      { "role": "user", "content": userPrompt } // повідомлення від користувача
    ],
    "max_tokens": 256,        // обмеження довжини відповіді (токенів)
    "temperature": 0.5        // параметр температури для варіативності
  };
  // 2. Налаштування параметрів HTTP-запиту
  var params = {
    "method": "post",
    "contentType": "application/json",
    "headers": { "Authorization": "Bearer " + apiKey },
    "payload": JSON.stringify(payload)
  };
  // 3. Виконання запиту до API Mistral
  var response = UrlFetchApp.fetch(url, params);
  // 4. Обробка отриманої відповіді
```

```
var data = JSON.parse(response.getContentText());
var answerText = data.choices[0].message.content;
return answerText;
```

Listing 2.3. Спрощена реалізація функції callMistralAPI, що відправляє запит до Mistral Large 2.1 і повертає текст відповіді.

Розглянемо цей код детальніше. На кроці 1 відбувається підготовка даних для запиту. Змінна apiKey містить секретний API-ключ, отриманий заздалегідь від платформи Mistral (для роботи з їх API необхідно зареєструвати обліковий запис та згенерувати ключ доступу). Цей ключ використовується для авторизації запитів. Змінна url – це адреса кінцевої точки API для отримання доповнення в чат-режимі (endpoint *chat completions*). Далі формується об’єкт payload – вміст HTTP-запиту. Згідно документації Mistral API, запит для генерації тексту повинен містити ідентифікатор моделі та список повідомлень діалогу.[18] У полі "model" ми вказуємо назву потрібної моделі – "mistral-large-2.1" (що відповідає останній версії великої моделі Mistral на момент реалізації). В полі "messages" передаємо масив об’єктів, які описують роль і вміст кожного повідомлення діалогу. Тут ми формуємо масив із одного об’єкта: { role: "user", content: userPrompt }, де userPrompt – це текст питання від користувача, отриманий як аргумент функції. Таким чином, ми надсилаємо моделі лише поточне питання, без додаткового контексту (у даній реалізації кожен запит обробляється незалежно, як окремий діалог). Опціонально можна було б додати системне повідомлення (role: "system") з інструкціями для моделі, наприклад, щоб вона відповідала як історичний експерт; у нашому випадку базова поведінка моделі вже передбачає генерування розгорнутих відповідей на запитання. Додатково у payload задано параметри генерації: max_tokens обмежує максимальну довжину відповіді (в токенах) для контролю над обсягом тексту, а temperature встановлює певну випадковість генерації (0.5 – помірний рівень, який дає баланс між детермінованістю та креативністю відповіді).

На кроці 2 задаються параметри HTTP-запиту для функції UrlFetchApp.fetch. Apps Script надає клас UrlFetchApp для виконання зовнішніх HTTP-запитів. Ми вказуємо метод post (HTTP POST запит), оскільки Mistral API очікує саме POST з JSON тілом. Далі, contentType: "application/json" сигналізує, що ми передаємо дані у форматі JSON. У блоці headers додаємо заголовок авторизації: "Authorization": "Bearer <ключ>". Mistral API використовує стандартну схему Bearer Token для API-ключів – ключ вставляється у заголовок Authorization[19] Нарешті, поле payload містить тіло запиту. Ми використовуємо JSON.stringify(payload) щоб перетворити JavaScript-об’єкт з повідомленням у рядок JSON.

На кроці 3 безпосередньо здійснюється виклик API: UrlFetchApp.fetch(url, params). Ця функція надсилає HTTP-запит з заданими параметрами на вказану URL і повертає об’єкт HTTPResponse з даними відповіді. В нашому випадку на сервері Mistral відбудеться обробка запиту: модель Mistral Large 2.1 згенерує продовження діалогу у вигляді відповіді асистента. Результат формується у JSON-форматі, що містить, серед іншого, згенерований текст відповіді.

Крок 4 – обробка відповіді. Викликаємо response.getContentText(), щоб отримати текстовий вміст HTTP-відповіді (який має бути JSON-рядком). Далі застосовуємо JSON.parse() для конвертації цього рядка у JavaScript-об’єкт data. Структура відповіді Mistral API аналогічна до форматів інших мовних моделей (як-от OpenAI API): зокрема, в об’єкті data очікується масив choices, що містить одну або кілька згенерованих відповідей[19]. Кожен елемент масиву choices містить вкладений об’єкт message з полями role (роль «assistant») і content (текст відповіді), а також службову інформацію (наприклад, finish_reason). Тому у коді ми звертаємося до data.choices[0].message.content – це і є текст згенерованої моделю відповіді на запитання користувача. Цей рядок зберігається у змінній answerText. Функція callMistralAPI повертає answerText як результат.

Повернувшись до doGet, нагадаємо, що вона отримує значення, яке повертає callMistralAPI, і вбудовує його у JSON для клієнта. Таким чином, весь цикл обробки завершується: запитання перетворено на відповідь, і ця відповідь готова до відправки користувачу.

Варто підкреслити, що функція callMistralAPI фактично виконується від імені сервера (на стороні Apps Script), тому API-ключ не розкривається користувачеві. Клієнт отримує вже готову відповідь, не знаючи деталей звернення до зовнішнього API. Це важлива перевага серверного підходу: конфіденційні ключі доступу зберігаються в безпечному середовищі (Google Script), а не в браузері. Крім того, Apps Script автоматично забезпечує асинхронність виклику зовнішнього API – поки відповідь генерується (що може тривати сотні мілісекунд або кілька секунд залежно від моделі), користувач просто чекає, а інтерфейс залишається чутливим (завдяки механізму промісів у fetch на клієнті, як описано вище).

Отримана від Mistral відповідь має формат діалогу, узгоджений із надісланим запитом. Модель Mistral Large 2.1 була навчена на різноманітних даних, включаючи історичні факти, тому здатна сформувати змістовну відповідь на питання про міжвоєнний період чи Другу світову війну українською або іншою мовою (модель багатомовна). У разі якщо модель не знає точної відповіді, вона може згенерувати кращу наближену відповідь або пояснення. Наш інтерфейс не застосовує додаткової фільтрації чи пост-обробки відповіді – весь текст, що повернула модель, відразу надсилається користувачу. Це робить інтеграцію максимально простою, хоча в реальних застосуваннях інколи додають модерацію контенту чи форматування відповіді на стороні сервера.

Отже, серверний скрипт Apps Script складається з відносно невеликої кількості рядків коду. Його головна функція doGet приймає запит від вебінтерфейсу, а функція callMistralAPI здійснює взаємодію з AI. Така

розділена структура полегшує налагодження і розширення: наприклад, у майбутньому можна модифікувати callMistralAPI для підтримки контекстного діалогу (надсилати не лише поточне питання, а й кілька попередніх пар "запитання-відповідь" для збереження контексту бесіди) – тоді інтерфейс перетвориться на повноцінний багатокроковий чат. В поточній реалізації кожен запит обробляється ізольовано (контекст діалогу не зберігається між запитами), але завдяки використанню формату messages API внести таке вдосконалення у майбутньому буде нескладно.

2.4. Логіка обробки запитів і формування відповіді

У цьому підрозділі узагальнюється алгоритм, за яким система перетворює введене користувачем питання на відповідь у діалоговому форматі. Взаємодія компонент розглядалась раніше (підрозділ 2.1), тут же робиться акцент на послідовності кроків обробки запиту на рівні програмної логіки із прив’язкою до реалізованих функцій. Основні етапи обробки можна описати наступним чином:

- Введення запиту користувачем: Користувач друкує своє питання в текстовому полі на вебсторінці (Google Sites) і натискає кнопку відправки. Наприклад, запит може бути: *«Які причини Другої світової війни?»*. Користувачські дії відслідковуються скриптом інтерфейсу (подія click на кнопці).
- Передача запиту на сервер: JavaScript-код інтерфейсу (Listing 2.1) зчитує текст питання і формує HTTP-запит до вебслужби Apps Script. Запит надсилається методом GET на URL скрипта, при цьому текст питання додається як параметр URL (наприклад, ...?prompt=Які%20причини%20Другої%20світової%20війни%3F).
  Браузер автоматично здійснює звернення через інтернет до сервера Apps Script (хмарна інфраструктура Google).

- Отримання запиту сервером (Apps Script): Як тільки запит досягає сервера, платформа Google Apps Script виконує наш скрипт, викликаючи функцію doGet(e). Об’єкт e містить поле e.parameter.prompt з текстом запитання (у наведеному прикладі це буде рядок "Які причини Другої світової війни?"). Функція doGet зберігає питання у змінну userPrompt і викликає callMistralAPI(userPrompt) для подальшої обробки.
- Формування запиту до Mistral API: Усередині callMistralAPI текст питання включається до JSON-повідомлення (поле messages з роллю "user"). Готується структура запиту відповідно до вимог Mistral API, як описано раніше. До цієї структури додаються технічні параметри (ідентифікатор моделі, ключ, параметри генерації).
- Виклик моделі Mistral: Функція UrlFetchApp.fetch відправляє HTTPS-запит до сервера Mistral. Запит містить авторизаційний заголовок з API-ключем та JSON-дані у тілі. На стороні Mistral AI відбувається аутентифікація ключа, після чого модель Mistral Large 2.1 отримує повідомлення користувача і генерує відповідь. Цей процес – по суті, робота ШІ-моделі – виконується на високопродуктивних серверах провайдера Mistral. Час генерації залежить від складності запиту та завантаженості сервісу, зазвичай займає від часток секунди до кількох секунд.
- Отримання відповіді від AI: Після завершення генерації Mistral API повертає відповідь. Це HTTP-відповідь, котра містить JSON-об’єкт з даними. В ньому, зокрема, є масив choices з однією або декількома згенерованими відповідями. У нашому випадку запитується одна відповідь, тому choices[0] містить об’єкт message з полем content, де і знаходиться текст згенерованої відповіді моделі[19]. Наприклад, для заданого вище питання модель може повернути текст: *«Причинами Другої світової війни стали невдоволення результатами Першої світової, підйом тоталітарних режимів у Європі (нацизм у Німеччині, фашизм в Італії), агресивна зовнішня політика цих держав, а також*

неспроможність Версальсько-Вашингтонської системи договорів запобігти новому конфлікту.» (це умовний приклад можливої відповіді, згенерованої ШІ).

- Обробка відповіді на сервері: Отримавши HTTP-відповідь від Mistral, функція callMistralAPI зчитує її вміст (response.getContentText()) і розбирає JSON (JSON.parse). Тепер відповідь доступна як об’єкт data. З нього видобувається саме текстова частина – data.choices[0].message.content – і зберігається в змінну answerText. Після цього callMistralAPI повертає answerText у функцію doGet. Функція doGet, у свою чергу, включає цей текст у JSON-структуру {"answer": "...тут відповідь..."} і завершується, повертаючи сформований результат клієнту через ContentService.
- Повернення відповіді клієнту: Скрипт Apps Script відправляє HTTP-відповідь, яка подорожує через інтернет до браузера користувача. Цей процес відбувається прозоро для користувача; на рівні браузера API Fetch отримує Response об’єкт. Метод response.json() зчитує текст і перетворює його на JavaScript-об’єкт data.
- Відображення відповіді в інтерфейсі: Клієнтський код отримує об’єкт data і бере з нього поле answer, що містить рядок відповіді. Потім цей текст встановлюється як вміст HTML-елемента з id="answer". В результаті на вебсторінці, під полем введення питання, з’являється текст відповіді від чат-бота. Користувач бачить повноцінне повідомлення, згенероване ШІ-агентом, як частину діалогового інтерфейсу. Наприклад, після питання про причини війни користувач побачить абзац історичного пояснення, сформований моделлю.
- Подальша взаємодія (цикл продовжується): Після отримання відповіді користувач може задати наступне питання, продовжуючи діалог. Кожне нове питання знову проходить описаний цикл. Нинішня реалізація не зберігає контексту попередніх питань автоматично, тому якщо користувач задасть уточнююче питання без повторення контексту,

модель може не зрозуміти, про що йдеться. Щоб підтримувати продовжений діалог, можна модифікувати код: зберігати історію messages на клієнті або сервері і додавати попередні повідомлення в масив перед викликом API. Це питання вдосконалення функціоналу; в дипломному проєкті акцент зроблено на базовому сценарії питання-відповідь в рамках окремих запитів. Проте навіть у такому режимі користувач отримує цінну інформацію у відповідь на кожне своє запитання.

Перелічені кроки демонструють, як узгоджено працюють фронтенд та бекенд частини для досягнення кінцевого результату. Така подія, як натискання кнопки, ініціює цілу послідовність мережевих звернень і обчислень, яка залишається непомітною для користувача окрім невеликої затримки на отримання відповіді. Завдяки асинхронному характеру JavaScript-операцій інтерфейс не «вісне» під час очікування відповіді – користувач може, наприклад, продовжувати прокручувати сторінку чи навіть скасувати запит.

З точки зору користувача, система проявляє себе як типовий чат: він вводить питання і отримує сформульовану відповідь у межах діалогового вікна. Відповіді моделі Mistral Large 2.1 зазвичай містять розгорнуті пояснення, факти, дати, імена (якщо питання історичне), іноді можуть містити припущення чи уточнення. Формат відповіді – суцільний текст, поділений на речення та абзаци. За потреби, інтерфейс можна було б удосконалити, додавши, наприклад, розбиття тексту на кілька повідомлень, відображення «думок» бота (ілюзії набору тексту), або можливість вибору варіантів відповідей, але ці функції виходять за рамки базової реалізації.

Отже, логіка обробки запиту реалізована досить лінійно і прозоро: Запитання → Вебслужба → Модель → Відповідь. Кожен компонент виконує свою роль і передає результат далі. Така побудова полегшує відстеження можливих помилок: на рівні клієнта можна бачити помилки мережевого

запиту (через .catch або інструменти розробника у браузері), на рівні Apps Script – логувати кроки або ловити винятки (через try/catch у doGet чи callMistralAPI), а на рівні Mistral API – аналізувати коди стану HTTP-відповідей (наприклад, помилки 400/401 при неправильному ключі або 500 при внутрішніх збоях). У процесі розробки проводилося тестування кожного етапу з окремо: спочатку перевірка, що Apps Script правильно приймає і повертає статичний відповідь, потім налагодження виклику зовнішнього API з фіксованим запитом, і нарешті інтеграція всього ланцюжка.

2.5. Забезпечення безпеки та обмеження доступу

При розробці інтерфейсу з інтегрованим ШІ-агентом важливо приділити увагу питанням безпеки, захисту від несанкціонованого доступу та неправильного використання. У даному підрозділі розглянуто поточний рівень безпеки реалізованої системи, потенційні ризики та шляхи їхнього зменшення. Враховуючи, що рішення базується на публічному вебзастосунку Apps Script і зовнішньому API, необхідно забезпечити захист як API-ключа, так і самого сервісу від зловмисних запитів.

Поточна конфігурація доступу. На етапі прототипування скрипт Google Apps Script було розгорнуто з параметрами доступу “Allow anonymous” (Дозволити всім) – це потрібно, щоб будь-який користувач Інтернету (в тому числі неавторизований Google) міг звернутися до вебслужби і отримати відповідь.

Відповідно, вебсторінка Google Sites також має публічний доступ. Таким чином, чат-бот доступний широкому загалу: будь-хто, знаючи URL сайту, може вводити запити і отримувати відповіді. Вебслужба Apps Script налаштована виконуватися від імені власника (скрипт працює під обліковим записом розробника), що дозволяє викликати її без передачі кінцевому користувачу прав доступу чи ключів. Код скрипту та API-ключ Mistral

зберігаються на сервері і безпосередньо не видимі користувачам. Це певний базовий рівень безпеки: запити від користувача не містять нічого, окрім тексту питання, і навіть у відповіді немає конфіденційних даних (лише згенерований текст). Отже, ключ Mistral API не передається клієнту і залишається прихованим у коді Apps Script.

Водночас, з таким публічним налаштуванням пов’язані кілька ризиків:

- Витік API-ключа. Хоча користувачі не бачать ключ безпосередньо, він присутній у коді скрипта. Якщо зловмисник отримав би доступ до вихідного коду Apps Script (через уразливість або через недбало наданий доступ), він міг би скопіювати ключ. Або ж, теоретично, атакувальник може спробувати відкрити виконуваний вебапп у режимі відладки (Google цього не дозволяє, але припустимо витік через логування). API-ключ Mistral дає можливість надсилати запити до моделі поза контролем нашого додатка, що може призвести до зловживань (наприклад, великих рахунків за використання або порушення умов використання сервісу). Отже, зберігання ключа прямо в коді – вразливе місце. Для покращення безпеки рекомендується щонайменше винести ключ із коду в сховище налаштувань Apps Script (Properties Service), щоб він не був видно у самому тексті скрипта[22]. Ще краще – використовувати проміжний проксі-сервер чи хмарний секрет-менеджер. У дипломному проєкті, зважаючи на академічний характер, допустимо зберігати ключ у коді, але при промисловому розгортанні це слід виправити.
- Зловмисне використання API (DoS/DDoS). Оскільки вебслужба відкрита публічно, ніхто не заважає будь-якому користувачу або боту надсилати до неї запити. Зловмисник може автоматизувати масове надсилання великої кількості запитань (наприклад, у циклі чи за допомогою ботнету). Це створює ризик DDoS-атаки – перевантаження сервісу. В такому разі може постраждати як власне Apps Script (є ліміти на кількість виконань скрипта в секунду, кількість викликів UrlFetchApp на

день тощо), так і зовнішній API Mistral (кількість запитів за хвилину, добова квота токенів). Без захисту, зловмисник міг би вичерпати квоту API або спричинити фінансові витрати, якщо використання моделі платне. Наразі система не має вбудованих обмежень частоти запитів – будь-який клієнт може викликати API стільки разів, скільки дозволить інфраструктура. Це потенційна уразливість.

- Несанкціонований доступ і зміна функціоналу. Публічний Apps Script дозволяє всім відправляти запити, але не дозволяє нікому змінювати код чи налаштування – це добре, код захищений обліковим записом власника. Однак, якщо розробник помилково надав би спільний доступ до проекту Apps Script стороннім або зробив би його редагованим публічно, зловмисник міг би змінити код (наприклад, вставити шкідливий контент у відповіді або перенаправити запити на інший ресурс). Тому важливо тримати код приватним. У нашому випадку код знаходиться у приватному репозиторії Google Apps Script і лише опублікований для виконання, без доступу на читання/редагування стороннім.

Заходи покращення безпеки. Для підвищення рівня безпеки та стійкості системи пропонується реалізувати такі механізми:

- Додавання CAPTCHA або перевірки користувача. Вбудувати у вебінтерфейс перевірку, що запит відправляє реальна людина, а не бот. Наприклад, перед відправкою питання вимагати пройти CAPTCHA (reCAPTCHA) – тоді автоматизовані скрипти не зможуть масово слати запити. Google Sites не має вбудованої капчі, але можна вставити віджет reCAPTCHA через Embed (потрібна відповідна інтеграція з фронтенд-кодом). Це ускладнить використання чат-бота, але значно захистить від ботів-зловмисників.
- Обмеження частоти запитів (throttling). На рівні Apps Script можна вбудувати логіку ліміту: наприклад, зберігати в Properties Service часові

мітки останніх запитів від певного клієнта (ідентифікуючи його за IP-адресою, що приходить в е даних, або за допомогою токена сеансу). Якщо один клієнт робить запити надто часто (наприклад, більше N запитів за хвилину), тимчасово блокувати або ігнорувати їх. Це запобігатиме зловживанням і ненавмисному перевантаженню. В Apps Script можна реалізувати простий лічильник, проте треба пам’ятати, що Apps Script має обмеження на час виконання, і надмірно складна логіка може погіршити продуктивність.

- Авторизація або обмеження кола користувачів. Якщо чат-бот призначено для певної аудиторії (наприклад, студентів конкретного курсу), доцільно захистити його паролем або зробити сайт приватним (лише для залогінених користувачів Google, яким надано доступ). У Google Sites можна налаштувати доступ за обліковими записами. У Apps Script при депLOY можна обрати режим виконання від імені користувача і вимагати авторизацію. Однак це ускладнить доступ для кінцевих користувачів і вимагатиме, щоб у них був Google-акаунт з правами. Для відкритого освітнього ресурсу такий шлях небажаний, але для внутрішнього використання – цілком прийнятний. Авторизація усуває анонімний доступ, що саме по собі зменшує ризики.
- Приховування та захист API-ключа. Як згадувалося, бажано не тримати ключ у відкритому коді. Застосування Properties Service Apps Script дозволяє зберегти ключ окремо і зчитувати його під час виконання[22]. Також Mistral (якщо це передбачено) можна було б обмежити використання ключа по доменах або IP. Наприклад, деякі API дозволяють прив’язати ключ до конкретного реферера або серверної IP-адреси – тоді навіть у разі компрометації ключ не можна використати з іншого середовища. Необхідно вивчити, чи надає Mistral AI можливості обмеження ключів. Якщо ні, то покладатися слід на секретність та часту ротацію ключа в разі підозри на витік.

- Моніторинг і логування. У продакшн-режимі варто відстежувати активність: скільки запитів надходить, які обсяги даних генеруються, чи немає піків активності у незвичайний час. Google Apps Script дозволяє вбудувати логування (через Logger.log або шляхом запису в Google Sheet) для кожного запиту: наприклад, записувати мітку часу, перші кілька символів питання, можливо, IP користувача (який можна отримати через e заголовки). Це допоможе виявити спроби зловживань та аналізувати, як використовується бот. У разі DDoS можна оперативно вимкнути скрипт (скасувати деплой) або змінити URL (перевипустити деплой з новим ID), щоби зупинити атаку.

З точки зору захисту даних, важливо зазначити, що наш чат-бот не оперує конфіденційною інформацією користувачів – запитуються загальновідомі історичні факти. Проте, якщо користувачі почнуть вводити персональні дані чи інші чутливі питання, слід бути обережними: ці дані передаються третій стороні (Mistral AI) для генерації відповіді. Політика приватності Mistral AI виходить за рамки нашого дослідження, але в реальному застосуванні варто переконатися, що користувачі розуміють: їхні питання можуть оброблятися зовнішнім сервісом. Для повноти безпеки можна додати на сайт повідомлення про збір і обробку введених даних.

Висновки до підрозділу: Поточна реалізація забезпечує базовий рівень безпеки (ключ прихований на сервері, код захищено від редагування). Але для переходу від прототипу до production-рішення необхідно впровадити додаткові заходи: CAPTCHA, обмеження частоти, авторизація доступу, безпечне зберігання ключів. Це захистить систему від зловживань і гарантовано збереже працездатність навіть під навантаженням або спробами атак. Баланс між зручністю користувача і рівнем захищеності має визначатися виходячи з контексту використання чат-бота. Для відкритого освітнього ресурсу мінімально варто реалізувати хоча б примітивне обмеження частоти запитів та моніторинг.

На завершення, відзначимо, що використання Google Sites і Apps Script само по собі надає певний рівень захисту: передача даних відбувається по HTTPS, Google забезпечує ізоляцію виконання скриптів, а також застосовує квоти та захист від надмірного використання ресурсів. Ці вбудовані механізми доповнюють описані вище заходи і сприяють надійності розробленого інтерфейсу з інтегрованим ШІ-агентом на основі Mistral.

Розділ 3. Тестування та можливості створенного сайту та вебінтерфейсу

3.1. Функціональні можливості сайту

Результатом практичної роботи є вебсайт «The Great Wars», розроблений на платформі Google Sites з вбудованим чат-ботом на базі моделі Mistral. Сайт присвячений тематиці «великих війн» XX століття (передусім Першій та Другій світовим війнам) і містить структуровану історичну інформацію. Інтеграція штучного інтелекту у вигляді чат-бота дозволяє користувачам ставити запитання по темі прямо на сторінках сайту та отримувати інтерактивні відповіді. Важливо, що реалізація відбувається без виділеного сервера – весь функціонал розгорнуто в хмарному середовищі Google (за допомогою можливостей Google Sites та пов'язаних сервісів). Весь програмний код, що забезпечує роботу чат-бота, винесено за межі сторінок і буде наданий у додатках до роботи (для збереження чистоти викладу в основному тексті).



Рис. 3.1. Головна сторінка сайту.

Переглянути сайт та протестувати його роботу можна використовуючи QR-код див. рис. 3.2.



Рис. 3.2. QR-Посилання на сайт.

Структура сайту. Сайт складається з кількох основних розділів, доступних через навігаційне меню (рис. 3.1). На момент розробки повністю реалізовані такі сторінки:

Вступ. Загальна інформація про історичний період між двома світовими війнами. Сторінка є вступною та слугує контекстом для інших розділів.

Штучний інтелект. Опис інтегрованого чат-бота на основі моделі Mistral. Наведено пояснення його можливостей та інструкції щодо користування.

Франція та Німеччина. Ці сторінки містять короткі огляди історичних подій відповідних країн у період світових воєн із інтегрованим віджетом чат-бота внизу.

Заплановано створення додаткових сторінок («Велика Британія», «СРСР», «Італія», «Японія», «США», «Польща», «Україна», «Інші держави»), що наразі

знаходяться в процесі розробки. Структура є масштабованою та дозволяє розширювати контент із збереженням єдиного стилю.

Структура сторінок і дизайн. Кожна сторінка сайту має єдиний шаблон: заголовок розділу поверх тематичного фонового зображення, текстовий контент з ілюстраціями (фото, карти) і інтерактивний блок чат-бота з полем вводу запитання та кнопкою «Send» (рис. 3.3). Інтерфейс забезпечує зручну навігацію між сторінками та інтерактивну взаємодію користувача з матеріалом.

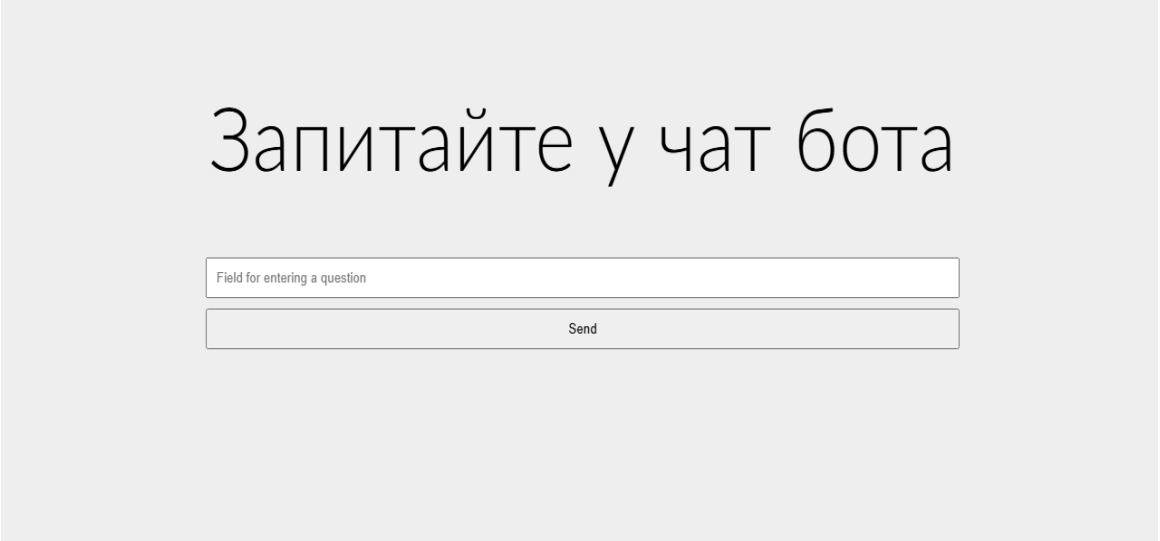


Рис 3.3 – Вбудований на сторінці віджет чат-бота (поле для вводу запитання та кнопка відправки) під закликом «Запитайте у чат бота»

Інтеграція чат-бота. Оскільки середовище Google Sites не дозволяє розміщувати користувацький серверний код безпосередньо, було використано можливість вбудовування HTML/JavaScript. Google Sites підтримує вставлення користувацького коду у вигляді iframe-гаджету[23], завдяки чому на сторінки сайту вдалося інтегрувати власний віджет чату. Суть реалізації полягає в тому, що поле вводу і кнопка «Send» прив’язані до скрипту, який відправляє введений текст на обробку мовною моделлю Mistral 7B. Модель Mistral 7B – це сучасна відкрита велика мовна модель з ~7 мільярдами параметрів, випущена восени 2023 року під відкритою ліцензією Apache 2.0

(дозволено комерційне і дослідницьке використання без обмежень)[24]. Ця модель була обрана через оптимальне поєднання якості відповіді та відносно невисоких вимог до обчислювальних ресурсів. За заявою розробників, Mistral 7B за багатьма тестами перевершує навіть деякі більші 13-мільярдні моделі (наприклад, Llama-2 13B) і наближається за продуктивністю до моделей масштабу 34B[24]. Така ефективність робить можливим її використання в хмарних сервісах без виділеного серверного обладнання.

У реалізації чат-бота на сайті використано хмарні функції для обробки запитів: після натискання кнопки «Send» запит користувача надсилається до API, що викликає модель Mistral (розгорнуту у хмарі). Отриману від моделі відповідь скрипт відображає на сторінці у вигляді розгорнутої текстової відповіді від AI-агента. Весь обмін відбувається асинхронно, невидимо для користувача – на сторінці він бачить лише інтерфейс чату. Таким чином, вдалося впровадити інтелектуальну функціональність, не виходячи за межі екосистеми Google (Google Sites + сторонній AI через вебслужбу). Відсутність окремого бекенд-сервера означає менші витрати на хостинг та простоту розгортання – достатньо публікації сайту на Google Sites і налаштування доступу до моделі Mistral у хмарі. Вся логіка інтеграції (HTML/JS код віджета, налаштування запитів до моделі тощо) буде детально наведена у додатках.

Отже, функціонально сайт «The Great Wars» забезпечує користувачу можливість отримувати інформацію двома шляхами: пасивно, читаючи підготовлений контент розділів, та активно, задаючи питання чат-боту. Такий інтерфейс поєднує статичний зміст з динамічно згенерованими відповідями, підвищуючи ефективність засвоєння матеріалу.

3.2. Приклади взаємодії користувача з ботом

Для оцінки роботи інтегрованого ШІ-агента було проведено тестові сеанси спілкування: декілька користувачів, серед яких студенти та знайомі автора, сформулювали запити до чат-бота українською мовою. Загалом у тестуванні взяли участь 8 осіб, які взаємодіяли з ботом, ставлячи різні типи запитів — від фактологічних до аналітичних. У процесі тестування було визначено загальний рівень задоволеності відповідями, точність поданої інформації та здатність чат-бота адекватно інтерпретувати запити. Нижче наведено 3 приклади реальних запитів і відповідей бота, що демонструють його можливості:

- Запит 1: *«Коли почалась Друга світова війна?»* – Відповідь бота: «Друга світова війна розпочалася 1 вересня 1939 року, коли Німеччина напала на Польщу. Цей акт агресії призвів до того, що вже 3 вересня 1939 року Великобританія і Франція оголосили війну Німеччині, офіційно розпочавши глобальний конфлікт». *Аналіз:* Відповідь коректно вказує дату початку війни і додає контекст про оголошення війни союзниками, що є релевантним і повним поясненням до простого запитання. Мова відповіді достатньо формальна, стилістично нейтральна, без грубих помилок.
- Запит 2: *«Яку роль відігравала Франція у Другій світовій війні?»* – Відповідь бота: «Франція на початку Другої світової війни була серед союзників, але зазнала поразки від німецького наступу у 1940 році. Після падіння Парижа і підписання Комп’єнського перемир’я було створено режим Віші, який співпрацював з нацистами. Водночас частина французів продовжила боротьбу – генерал Шарль де Голль очолив рух «Вільна Франція» з центром у Лондоні, а всередині країни

діяв Рух Опору. У 1944 році союзники висадилися в Нормандії, що привело до визволення Франції. Таким чином, Франція пройшла шлях від поразки до участі в перемозі над нацизмом у 1945 році.» *Аналіз:* Ця відповідь демонструє повноту – бот згадав усі ключові моменти: поразку 1940 р., колабораціоністський уряд, рух Опору, «Вільну Францію» де Голля, висадку в Нормандії і визволення. Відповідь структурована хронологічно, що свідчить про розуміння запиту. Релевантність висока: інформація стосується саме питання. Мова відповіді літературна, зрозуміла, стилістично наближена до нейтрально-наукової (що доречно для історичної тематики). За відгуками тестувальників, така відповідь виглядає схожою на коротку енциклопедичну довідку, що є позитивною ознакою.

- Запит 3: *«Що таке штучний інтелект і як він використаний на цьому сайті?»* – Відповідь бота: «Штучний інтелект (ШІ) – це розділ інформатики, що займається створенням комп'ютерних систем, здатних виконувати завдання, які зазвичай потребують людського інтелекту. Приклади таких задач – розпізнавання мовлення, прийняття рішень, переклад мов тощо. На цьому сайті ШІ використано у формі чат-бота на основі моделі Mistral. Цей бот інтегровано в сторінки, і він може відповідати на запитання користувачів щодо тематики сайту (історичних подій). Тобто, коли ви вводите запитання у чат, модель ШІ аналізує текст і генерує відповідь українською мовою, спираючись на свої навчені знання.» *Аналіз:* Відповідь починається з загального пояснення поняття ШІ – тобто модель коректно визначила контекст і подала базове визначення терміну. Далі бот прямо вказав, як штучний інтелект застосований на сайті, фактично описавши власну роль. Це свідчить, що бот розуміє інструкцію/контекст сайту (можливо, інформація зі сторінки «Штучний інтелект» вплинула на відповідь). Якість відповіді в аспекті змісту висока: дано і визначення, і

конкретизацію щодо сайту. Мовно відповідь грамотна, стиль доброзичливий і доступний для неспеціаліста.

Загальний аналіз відповідей показав, що інтегрований бот адекватно інтерпретує запити користувачів та генерує змістовні відповіді. На запитання з історичної тематики (приклади 1–2) модель Mistral видала фактично правильну інформацію, близьку до енциклопедичної. Жодних серйозних фактичних помилок у відповідях не виявлено — відповіді узгоджуються із загальновідомими історичними даними. Відповіді є розгорнутими та містять контекстні деталі, що підвищує їхню інформативність.

Щодо якості мови, тестувальники відзначили, що відповіді бот надає українською мовою, зрозумілою та стилістично коректною. Мова побудована грамотно, використана термінологія відповідає тематиці, стиль нейтральний. Незначні недоліки спостерігалися у випадках надмірно книжної побудови речень, що іноді створювало враження формальності.

Релевантність і повнота відповідей оцінена як висока: у більшості випадків бот відповідав по суті поставлених запитань. Найкраще модель справляється із запитаннями, що мають однозначну фактологічну відповідь (дата, назва, визначення). Запитання, що потребують узагальнень або аналітики, також обробляються коректно, хоча в деяких випадках відповіді могли бути більш деталізованими. Водночас, за відгуками тестувальників, модель не відхиляється від тематики сайту, що свідчить про належну фільтрацію запитів.

Враження користувачів були загалом позитивними. Респонденти відзначили зручність взаємодії з ботом та можливість миттєво уточнювати інформацію, що значно підвищує інтерактивність сайту. Загальний

зворотний зв'язок свідчить про високу корисність чат-бота у процесі взаємодії з історичними матеріалами.

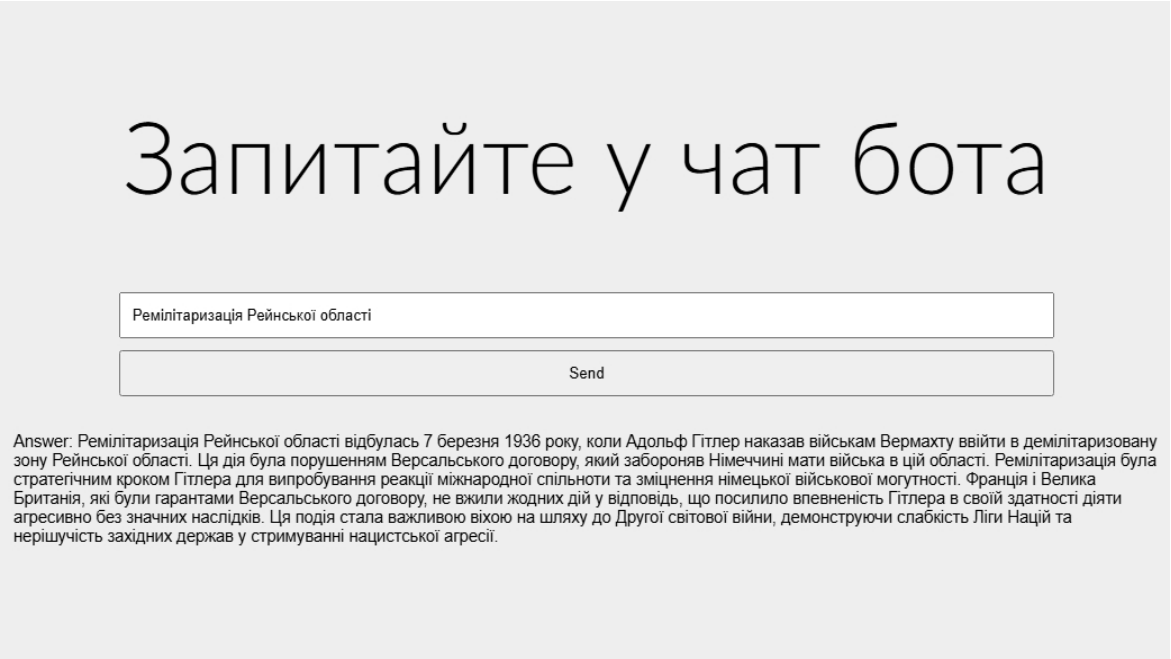


Рис. 3.3 – Приклад відповіді чат-бота на запитання користувача (скриншот діалогового вікна на сайті)

3.3. Оцінка ефективності та перспективи вдосконалення

Швидкодія та стабільність. Під час практичного тестування було приділено увагу швидкості відповіді чат-бота та загальній стабільності його роботи в умовах обмежених ресурсів. Середній час відгуку бота на запит становить приблизно *5–10 секунд*, залежно від складності питання та завантаженості хмарного сервера, що виконує модель. Такий час є прийнятним для прототипу: користувач відчуває невелику затримку, проте з огляду на обсяг згенерованого тексту (часто 3–5 речень) це виглядає логічно. Усі відповіді надходили успішно, зависань чи збоїв під час тестових сесій не спостерігалось. Стабільність роботи забезпечується тим, що Google Sites хостить лише інтерфейс, а сам AI-відповідач працює у надійному хмарному середовищі. Відсутність власного сервера означає, що немає проблем з підтриманням серверного ПЗ чи його падінням – вся відповідальність за

виконання моделі покладена на зовнішню службу, яка спроектована для обробки AI-запитів. Проте існує залежність від інтернет-з’єднання: при повільному з’єднанні користувача або тимчасовій недоступності API відповідь може затриматися або не прийти. За час випробувань таких випадків не зафіксовано, тобто надійність рішенння можна оцінити як високу для навчального прототипу.

Оцінка ефективності функціоналу. Ефективність інтегрованого підходу UI + AI проявляється у підвищенні залученості користувачів. Якщо раніше відвідувач сайту був обмежений статичним контентом, то тепер він фактично отримує персоналізований довідник. Враховуючи позитивні відгуки тестувальників про корисність бота, можна стверджувати, що мети розробки досягнуто: інтерфейс користувача став інтерактивним і «розумнішим». Таким чином, ефективність рішення оцінюється не тільки швидкодією, але й користю для аудиторії.

Перспективи розвитку. Проект має значний потенціал для розширення та вдосконалення, зокрема в таких напрямах:

- Розширення тематичного наповнення. На даний момент сайт містить наповнені сторінки лише по декількох країнах (Франція, Німеччина тощо). Планується доповнити усі заплановані розділи (Велика Британія, СРСР, США, Японія, тощо) повноцінними матеріалами. Це збагатить контент і дозволить ставити боту більше різноманітних питань. Модель Mistral володіє знаннями і про інші аспекти Другої світової війни, тож такий крок лише покращить релевантність її відповідей, коли користувачі питатимуть про події в цих країнах.
- Багатомовність. Наразі інтерфейс сайту і бот орієнтовані на україномовного користувача. В перспективі можна реалізувати мультимовну підтримку: хоча б додати англійську версію сторінок і налаштувати бота на двомовність. Модель Mistral (поточна версія)

непогано розуміє англійську та інші популярні мови, а розробники планують покращувати багатомовну підтримку в нових версіях[26]. Можна, наприклад, визначати мову запитання і відповідно або перекладати його, або використовувати окрему модель. Багатомовність суттєво розширила б аудиторію сайту, зробивши його корисним для іноземних студентів з історії.

- Історія діалогу та контекстність. Поточна реалізація чат-бота сприймає кожен запит ізольовано, не пам’ятаючи попередніх питань користувача. Тобто діалог не зберігається – якщо спитати уточнення на основі попередньої відповіді, модель може не зрозуміти контекст (бо він не передається повторно). Перспективне вдосконалення – реалізувати збереження історії діалогу в межах сеансу. Це можна зробити шляхом додавання до кожного наступного запиту кількох попередніх пар «запит-відповідь». Таким чином бот матиме контекст і зможе відповідати на уточнюючі питання більш предметно. Додання пам’яті діалогу підвищить «людяність» спілкування з ботом і наблизить його до повноцінного віртуального співрозмовника.
- Поліпшення моделі та знань. Mistral 7B – доволі потужна модель, але її знання базуються на даних до 2023 року і можуть мати прогалини або неточності. У майбутньому варто розглянути тонке налаштування (fine-tuning) моделі під специфічний корпус з історії, або використати механізми Retrieval-Augmented Generation (RAG), коли бот перед формуванням відповіді шукає потрібну інформацію в базі даних/вікі. Для цього, щоправда, знадобиться більш складна архітектура із залученням серверної логіки. Але навіть у рамках існуючої платформи можна періодично оновлювати модель на новішу версію (якщо Mistral або інші open-source моделі випустять покращені варіанти). За даними компанії Mistral AI, 7-мільярдний модель – лише перша в серії, плануються більш досконалі моделі з покращеними можливостями

міркування та ширшою мовною підтримкою[26]. Отже, існує шлях апгрейду інтелектуальної складової проекту.

- Залучення мультимедійних та інтерактивних елементів. Окрім текстового чат-бота, можна розглянути впровадження інших AI-функцій: наприклад, голосового помічника (щоб бот озвучував відповіді), або генерування інтерактивних карт чи хронологічних шкал на основі запиту користувача. Це виходить за рамки текстового інтерфейсу, але могло б стати цікавим напрямом для подальшої роботи.

Поточний рівень реалізації сайту можна оцінити як прототип із завершеним основним функціоналом. Усі поставлені завдання – створення інформаційного сайту, інтеграція AI-агента, забезпечення базової інтерактивності – виконані. Сайт успішно працює в тестовому режимі, контент та бот доступні. Водночас є незавершені моменти (заповнення всіх розділів контентом, оптимізація деяких технічних показників), які позначені як плани доопрацювання. Описані вище перспективи розвитку свідчать, що проект має куди зростати, що є нормальним для будь-якого програмного продукту. У наступному підрозділі розглянуто результати тестування сайту з технічного погляду та окреслено конкретні кроки з оптимізації.

3.4. Тестування та оптимізація сайту

Для комплексної оцінки якості розробленого вебсайту було застосовано інструменти Chrome DevTools, зокрема аудит Lighthouse. Lighthouse генерує автоматичний звіт за кількома ключовими категоріями: Performance (продуктивність), Accessibility (доступність), Best Practices (кращі практики) та SEO (пошукова оптимізація)[27][28]. Аудит проводився для опублікованої версії сайту «The Great Wars» у режимі емуляції настільного пристрою. На рисунку 3.4 наведено зведення результатів Lighthouse для головної сторінки (розділ «Вступ»):

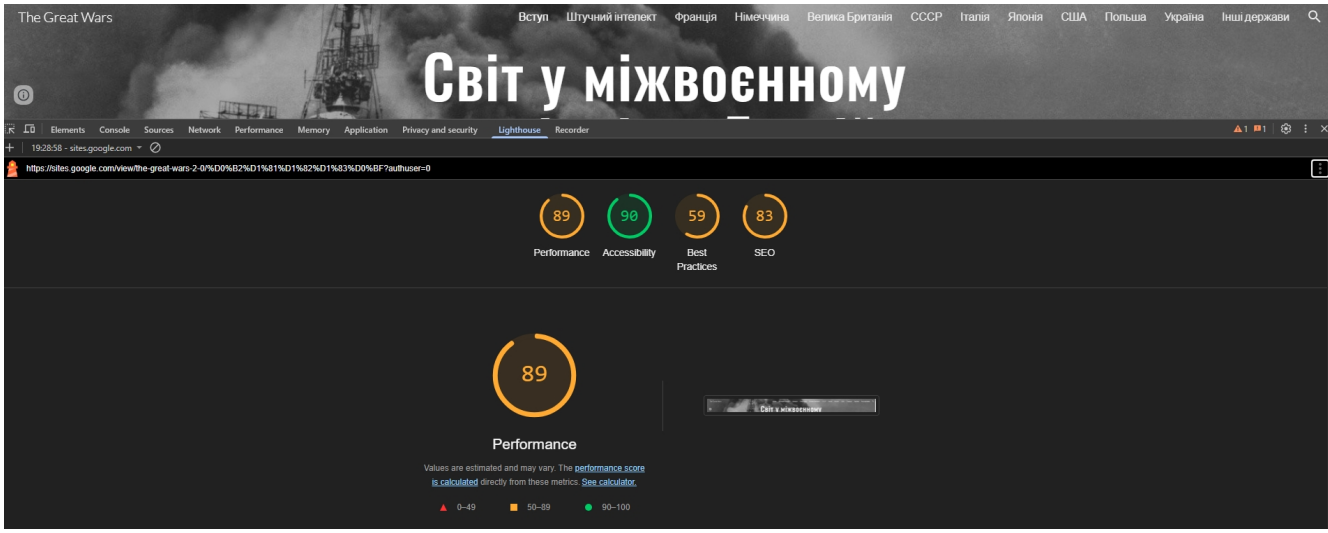


Рис 3.4 – Результати автоматичного аудиту Lighthouse для сайту (підсумкові бали за категоріями Performance, AcceMistssibility, Best Practices, SEO)

Як видно з рис. 3.4, показники мають такий вигляд: Performance – 89, Accessibility – 90, Best Practices – 59, SEO – 83 (значення наведені у відсотках відповідності еталонам). Детально проаналізуємо кожен з цих аспектів, причини просідань та можливі шляхи оптимізації:

- Performance (89) – продуктивність. Даний показник відображає, наскільки швидко завантажується сторінка і наскільки плавно вона реагує на дії користувача[29]. Значення 89 із 100 свідчить про досить хорошу оптимізацію, хоча є певні резерви для покращення. Lighthouse аналізує такі метрики, як First Contentful Paint, Time to Interactive, Speed Index тощо, які у нашому випадку були близькі до нормальних значень (конкретні цифри доступні в детальному звіті). Факторами, що трохи знижують продуктивність, є:
    - Великі зображення на фоні. Фонові історичні світлини у високій роздільності покращують візуальне сприйняття, але впливають на час завантаження. Рекомендація – компресувати зображення або

використовувати сучасні формати (WebP) для зменшення їх розміру без втрати якості.

- Відсутність довготривалого кешування статичних ресурсів. Аудит виявив, що деякі статичні файли (зображення, скрипти) не мають ефективної кеш-політики – браузер не отримує вказівок зберігати їх довго. Через це при повторних відвідинах частина контенту завантажується наново. Рекомендовано налаштувати кешування: наприклад, додати заголовки Cache-Control з терміном зберігання для статичних ресурсів[30][31]. У контексті Google Sites не всі такі налаштування доступні розробнику, але можна частково вирішити проблему, завантаживши зображення в оптимізованому вигляді (меншого розміру) – тоді навіть повторне їх завантаження не буде тривалим.
- Незначна затримка від стороннього скрипта. Вбудований віджет чат-бота підключає JavaScript (через iframe), який додає трохи до часу повного завантаження сторінки. Lighthouse може кваліфікувати це як render-blocking script. Хоч ефект і малий, можна оптимізувати, зробивши завантаження скрипта відкладеним (defer/async) якщо це дозволяється вбудовою.

В цілому, продуктивність сайту для невеликого обсягу контенту є на рівні. Досягнення 100/100 не є самоціллю, адже, наприклад, використання великих зображень виправдане тематично. Однак впровадження зазначених оптимізацій (стиснення графіки, кешування) може підняти показник Performance в майбутньому.

- Accessibility (90) – доступність. Цей показник оцінює, наскільки сайт пристосований для користувачів з особливими потребами (наприклад, чи можуть його успішно використовувати люди з порушеннями зору за допомогою скрінрідерів)[32]. Оцінка 90 говорить про високий рівень доступності, хоча є кілька зауважень:

- Alt-атрибути зображень. Lighthouse звернув увагу, що не всі зображення мають заповнений атрибут alt. Alt-текст потрібен для опису вмісту зображення невізуальним користувачам[32]. У нашому випадку декоративні фонові зображення можна позначити порожнім alt (що допустимо), але, наприклад, емблему сайту чи інфографіку слід забезпечити описовим текстом. План дій – додати змістовні описи до всіх важливих зображень.
- Контрастність тексту. Тест показав, що кольорові контрасти тексту на фоні ледь відповідають вимогам (наприклад, білий текст на світлинах). Стиль було вибрано для естетики, але для кращої доступності можна затемнити трохи фон під текстом або використати тінь для шрифту, аби збільшити контраст. Невеликий коригування стилів підвищить цей аспект.
- Навігація з клавіатури. Lighthouse не виявив проблем із навігацією сайтом через клавіатуру – всі посилання в меню фокусуються і доступні. Це важливий момент, який свідчить про хорошу базову шаблонну структуру Google Sites.

Отже, після внесення alt-текстів та дрібного удосконалення стилів ми можемо прагнути до 100% доступності. Важливо, що увага до цих деталей не лише покращує оцінку Lighthouse, а й робить сайт зручнішим для всіх категорій користувачів.

- Best Practices (59) – найнижчий з отриманих показників, що свідчить про наявність зауважень щодо дотримання сучасних вебстандартів та безпеки. Lighthouse виконує 16 перевірок, зосереджених переважно на питаннях безпеки та правильного використання вебтехнологій[33]. Причини низького балу та шляхи виправлення:

  - Відсутність HTTPS-вмісту. Сайт Google Sites автоматично обслуговується по HTTPS, тож проблема не у протоколі. Однак,

якщо б якісь ресурси (скрипти чи зображення) вантажились по незахищеному HTTP, Lighthouse знизив би бал. Слід переконатися, що всі зовнішні посилання і ресурси, якщо такі є, використовують https://. У нашому випадку, посилання на API моделі Mistral чи інші – захищені, тому тут все гаразд.

- Вразливості сторонніх бібліотек. Lighthouse аналізує підключені JS-бібліотеки на наявність відомих вразливостей. Ймовірно, використаний нами скрипт для чат-бота або елементи Google Sites містять бібліотеку, яку Lighthouse позначив як застарілу/уразливу. Наприклад, часто сигналізується про jQuery, якщо версія не остання. Вплив на наш сайт мінімальний (зловмисний код додати важко), але для кращого показника можна оновити бібліотеки за можливості. У випадку Google Sites це робить сама платформа, тому прямого контролю немає.
- Використання застарілих API. Lighthouse радить уникати старих підходів, таких як document.write() чи неполіфіллених функцій. У нашому коді таких немає, але Google Sites могла внутрішньо щось таке використати. Ця проблема теж не цілком під нашим контролем, окрім як повідомити про неї на форумах розробників Google.
- Наявність noopener для зовнішніх посилань. Поширена дрібниця – всі посилання з target="_blank" повинні мати rel="noopener" для безпеки (щоб нова вкладка не мала доступу до сторінки-джерела). Якщо у контенті сайту є зовнішні посилання, треба вручну переконатися, що додано noopener. Це можна виправити при редагуванні текстів (Google Sites дозволяє редагувати HTML посилань).

Сконцентрувавшись на тих пунктах, що можемо виправити (посилання, зовнішні ресурси), можна очікувати підвищення показника Best

Practices. В цілому ж, 59 – це сигнал звернути увагу, але не критична помилка. Сайт все одно функціонує правильно; даний бал більше впливає на технічну «чистоту» проекту.

- SEO (83) – пошукова оптимізація. Цей показник відображає, наскільки сайт відповідає базовим критеріям, що дозволяють пошуковим системам легко індексувати і розуміти контент[34]. Результат 83 означає, що є кілька моментів, які потрібно виправити, щоб сторінки краще ранжувалися та відображалися у пошуку:

  - Відсутній meta description. Lighthouse вказав, що на сторінках відсутній мета-опис (тег <meta name="description"> у HTML-заголовку). Meta description не впливає напряму на ранжування, але саме він відображається у фрагменті (сніпеті) результатів Google. Наразі при поширенні посилання або перегляді в пошуку може бути автоматично витягнутий текст зі сторінки, що не завжди вдало. Треба додати змістовні описи 150–160 символів для кожної сторінки (в Google Sites це можна зробити через налаштування сторінки – поле «Description»). Після цього SEO-оцінка суттєво зросте.
  - Структура заголовків. Сторінки слід переконфігурувати, щоб мали один H1 (назву розділу) і логічну ієрархію H2/H3 для підзаголовків. Google Sites, ймовірно, вже робить назву сторінки H1. Додатково, якщо текст сторінки довгий, доцільно поділити на підрозділи з підзаголовками. Lighthouse звертає увагу на ієрархію заголовків для кращого SEO.
  - Adaptive/Mobile friendliness. Хоча окремого показника PWA ми не аналізували, SEO-аудит враховує, чи сайт придатний для мобільних (в нашому випадку сайт адаптивний, Google Sites це забезпечує автоматично). Оцінка 83 натякає, що, можливо, є дрібні недоліки в мобільній версії, але загалом вимоги виконано.

- Semantic HTML. Lighthouse SEO перевіряє наявність семантичної розмітки (навігаційні елементи, правильне використання тегів). Наш сайт побудований на шаблоні, тому тут особливих проблем немає.

Виконавши рекомендацію щодо meta description і переконавшись у коректності заголовків, можна очікувати, що SEO-показник наблизиться до 100. Важливо розуміти, що Lighthouse перевіряє лише базові речі (наявність опису, title, тегів). Отримати 100/100 легко після виправлення, але для реального просування сайту в інтернеті потрібні глибші SEO-заходи (які виходять за рамки дипломної роботи, але можуть бути враховані якщо роботу буде продовжено).

Значення показників Lighthouse. На завершення відзначимо, чому досягнення високих балів за цими метриками є важливим.

- Висока Performance гарантує швидке завантаження, а значить – кращий досвід користувачів і нижчий відсоток відмов (користувач не закриє сторінку, чекаючи завантаження). Для освітнього ресурсу це критично, адже студенти часто заходять з мобільних пристроїв, де мережа може бути повільнішою.
- Accessibility безпосередньо впливає на охоплення аудиторії: чим більш доступний сайт, тим більшій кількості людей він приносить користь. Крім того, доступність – це вимога етична і юридична (в багатьох країнах стандарти WCAG є обов’язковими для освітніх сайтів). Отже, робота над доступністю робить проект зрілішим.
- Best Practices відображають дотримання сучасних підходів розробки. Хоч користувач їх не бачить напряму, але висока оцінка означає, що сайт більш захищений і підтримуваний. Наприклад, використання HTTPS і нових API знижує ризики витоку даних та підвищує сумісність із новими браузерами.

- SEO важливе, якщо ми хочемо, щоб сайт знаходили через пошук. Освітній контент має потенціал залучати широку аудиторію (учнів, студентів, викладачів історії), тому базова оптимізація необхідна. Хоча наш сайт створений в рамках навчального проекту і не претендує на високі позиції у видачі, правильне заповнення метаданих, заголовків і alt-текстів – це добра практика створення контенту.

Практичні кроки оптимізації. На основі аудиту Lighthouse було складено короткий план конкретних дій:

- Додати опис (meta description) для кожної сторінки через налаштування сайту.
- Переглянути всі зображення: вписати alt-тексти, а також за можливості замінити великі файли оптимізованими (наприклад, стиснути JPEG чи конвертувати в WebP).
- Перевірити всі гіперпосилання у контенті: додати атрибут rel="noopener" для зовнішніх лінків, переконатися в актуальності протоколів.
- Переглянути звіт Best Practices на предмет конкретних уразливих бібліотек – за можливості оновити або видалити їх. Якщо проблема стосується інтегрованого JS-скрипта, спробувати знайти більш сучасну версію або альтернативу.
- Внести дрібні правки у дизайн (контраст тексту на фоні) для покращення видимості.
- Розглянути можливість увімкнення кешування через сторонній CDN для статичних файлів, якщо Google Sites дозволятиме (альтернативно – миритися, оскільки контенту небагато).

Виконання цих кроків, ймовірно, покращить усі чотири показники Lighthouse. Важливо підкреслити, що оптимізація – це ітеративний процес:

після виправлень слід повторно запускати аудит, перевіряти нові показники, поки всі критичні зауваження не будуть усунені.

На завершення розділу слід зазначити, що всі фрагменти програмного коду, конфігурацій та сценаріїв, реалізованих у ході розробки (включно з кодом віджета чат-бота, налаштуваннями Google Sites тощо), будуть наведені у додатках. Це дозволить детально ознайомитися з технічною реалізацією інтеграції Mistral-бота у вебсайт без виділеного сервера, що може стати у нагоді майбутнім дослідникам або розробникам подібних систем. Таким чином, дипломна робота досягає своєї мети – не лише теоретично, але й практично демонструє можливість розробки сучасного інтерфейсу з штучним інтелектом, оцінює його ефективність та окреслює напрямки подальшого розвитку проекту.

ВИСНОВКИ

У кваліфікаційній роботі успішно реалізовано поставлену мету – розроблено вебінтерфейс з інтегрованим чат-ботом на основі моделі Mistral, орієнтований на надання історичної інформації про міжвоєнний період та події Другої світової війни. Відповідно до сформульованих завдань було проведено аналіз предметної області та сучасних технологій штучного інтелекту для чат-ботів, спроєктовано архітектуру програмного рішення, розроблено користувацький інтерфейс на платформі Google Sites та реалізовано інтеграцію з Mistral API засобами Google Apps Script. Результатом є функціонуючий вебсайт «The Great Wars» з інтерактивним чат-ботом, що коректно відповідає на запити користувачів з історичної тематики.

Практична значущість розробки полягає у створенні доступного інструмента для здобуття знань з історії міжвоєнної доби і Другої світової війни у форматі діалогу з віртуальним співрозмовником. Вебінтерфейс із чат-ботом може бути використаний у закладах освіти та для самостійного навчання, оскільки поєднує структурований історичний контент (статті про події та держави згаданого періоду) з можливістю ставити уточнювальні питання чат-боту. На відміну від статичних джерел, інтерактивний чат-бот забезпечує персоналізовану взаємодію – користувач може отримати довідкову інформацію чи пояснення у зручній розмовній формі. Це підвищує залученість аудиторії та сприяє кращому засвоєнню матеріалу за рахунок активної форми навчання. Розроблений вебресурс має інтуїтивно зрозумілий інтерфейс і не потребує встановлення спеціального програмного забезпечення, що розширює коло потенційних користувачів.

Технічні здобутки роботи включають створення оригінальної архітектури програмного рішення з використанням хмарних сервісів Google та сучасної моделі штучного інтелекту Mistral. Застосовано зв’язку «Google Sites + Google Apps Script», яка дозволила вбудувати кастомізований HTML/JS-код

чат-інтерфейсу у сторінку Google Sites та здійснити обробку повідомлень через Apps Script. За допомогою служби Apps Script UrlFetchApp реалізовано запити до Mistral API для отримання згенерованих відповідей на основі введених користувачем історичних запитів. Архітектура розподіляє функції між клієнтською частиною (інтерфейс чату, що приймає вводи) та серверною логікою в Apps Script (взаємодія з AI-моделлю і повернення відповіді), що забезпечує чітке розмежування ролей і полегшує підтримку коду. Інтеграція Google Apps Script з зовнішнім AI-сервісом Mistral продемонструвала можливість швидкого створення вебзастосунків без потреби у виділеному сервері: уся інфраструктура розгорнута на базі хмарних сервісів Google. Це підтверджує ефективність обраних технологій для задач розробки подібних інформаційних систем. Окремо варто відзначити, що забезпечено базові заходи безпеки та конфіденційності: використано захищений протокол HTTPS для API-запитів, ключ API Mistral зберігається на стороні серверного скрипта (прихований від користувача), а доступ до нього здійснюється з належними обмеженнями.

Перспективи подальшого розвитку розробленого рішення охоплюють як розширення функціональності, так і поглиблення змістового наповнення та підвищення надійності системи. По-перше, планується поступове додавання контенту – розширення бази історичних даних, включення нових тематичних розділів (наприклад, детальніший опис подій на інших театрах війни, біографії ключових постатей міжвоєнного періоду тощо) та мультимедійних матеріалів. Це дозволить збагатити знання, які користувач може отримати від чат-бота, і зробить ресурс кориснішим для широкої аудиторії. По-друге, актуальним є удосконалення безпеки та якості відповіді: впровадження фільтрації потенційно некоректного або упередженого контенту, налаштування моделей модерування для запобігання небажаних відповідей на чутливі історичні теми, а також періодичне оновлення AI-моделі (за появи нових версій Mistral чи аналогів) для поліпшення точності та повноти відповідей. По-третє, можна

реалізувати додаткові можливості UI/UX – наприклад, підтримку голосового введення та озвучення відповідей для підвищення зручності, інтеграцію функції пошуку по матеріалах сайту, адаптацію інтерфейсу для мобільних пристроїв тощо. Нарешті, подальшим кроком розвитку може стати залучення спільноти істориків та викладачів до наповнення і перевірки знань чат-бота, аби забезпечити академічну достовірність інформації. Таким чином, проєкт має значний потенціал для масштабування та вдосконалення, що в перспективі перетворить його на повноцінну платформу інтерактивної історичної просвіти із застосуванням штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cengage Group. (2024). *2024 in Review: AI & Education*. Retrieved from https://www.cengagegroup.com/news/perspectives/2024/2024-in-review-ai--education/#:~:text=,since%20last%20year
2. FreeTech4Teach. (2024). *YourStory: Chat with History*. Retrieved from https://freetech4teach.classwork.com/2024/10/yourstory-chat-with-history/#:~:text=What%20Makes%20YourStory%20Stand%20Out%3F
3. Chatbot.com. (2024). *The Future of Chatbots*. Retrieved from https://www.chatbot.com/blog/future-of-chatbots/#:~:text=What%20is%20the%20Future%20of,the%20norm%20for%20customer%20service
4. Wikipedia. (2024). *DeepSeek*. Retrieved from https://en.wikipedia.org/wiki/DeepSeek#:~:text=Released%20under%20the%20MIT%20License,14
5. Wikipedia. (2024). *Gemini (language model)*. Retrieved from https://en.wikipedia.org/wiki/Gemini_(language_model)#:~:text=Gemini%20was%20said%20to%20be,popularity%20had%20been%20aggressively%20challenged
6. Wikipedia. (2024). *Gemini: DeepMind CEO Demis Hassabis*. Retrieved from https://en.wikipedia.org/wiki/Gemini_(language_model)#:~:text=DeepMind%20CEO%20Demis%20Hassabis%20,5
7. Wikipedia. (2024). *Gemini: Pichai announcement on December 6, 2023*. Retrieved from https://en.wikipedia.org/wiki/Gemini_(language_model)#:~:text=On%20December%206%2C%202023%2C%20Pichai,It%20was%20made
8. Wikipedia. (2024). *Gemini: Impending impact on the industry*. Retrieved from https://en.wikipedia.org/wiki/Gemini_(language_model)#:~:text=With%20n

ews%20of%20Gemini%27s%20impending,12

9. Mistral AI. (2024). *Mistral Large 2.1: Advanced Function Calling Capabilities*. Retrieved from https://mistral.ai/news/mistral-large-2407#:~:text=Today%2C%20we%20are%20announcing%20Mistral,and%20advanced%20function%20calling%20capabilities
10. TestingCatalog. (2024). *Mistral AI gears up to launch multimodal Large 2.1*. Retrieved from https://www.testingcatalog.com/mistral-ai-is-gearing-to-launch-multimodal-large-2-1-with/#:~:text=Overall%2C%20Mistral%20AI%20continues%20to,transparency%20in%20its%20development%20approach
11. Nature. (2025). *GPT and Open-Source LLMs*. Retrieved from https://www.nature.com/articles/s41591-025-03727-2#:~:text=including%20GPT,source%20LLMs
12. University of Michigan. (2024). *Embed CSS, HTML, JavaScript in a webpage*. Retrieved from https://documentation.its.umich.edu/node/922#:~:text=Embed%20CSS%2C%20HTML%2C%20JavaScript%2C%20or%C2%A0a,webpage
13. University of Michigan. (2024). *How to insert a preview and click Insert*. Retrieved from https://documentation.its.umich.edu/node/922#:~:text=You%20are%20also%20able%20to,a%20Preview%2C%20then%20click%20Insert
14. Google Developers. (2024). *Apps Script Overview*. Retrieved from https://developers.google.com/apps-script/overview#:~:text=Apps%20Script%20is%20versatile,other%20things%2C%20you%20can
15. Mistral AI. (2024). *Mistral Large 2.1: Advanced Function Calling Capabilities*. Retrieved from https://mistral.ai/news/mistral-large-2407#:~:text=Today%2C%20we%20are%20announcing%20Mistral,and%20advanced%20function%20calling%20capabilities

16. Mistral AI. (2024). *Mistral Large 2: Supported Programming Languages and Advanced Integration*. Retrieved from https://mistral.ai/news/mistral-large-2407#:~:text=Mistral%20Large%202%20has%20a,C%2C%20C%2B%2B%2C%20JavaScript%2C%20and%20Bash
17. Google Developers. (2024). *Building Web Apps with Google Apps Script*. Retrieved from https://developers.google.com/apps-script/guides/web#:~:text=If%20you%20build%20a%20user,it%20directly%20from%20their%20browsers
18. Google Developers. (2024). *Google Apps Script Web App Services and Objects*. Retrieved from https://developers.google.com/apps-script/guides/web#:~:text=,service%20%20%20114%20object
19. Mistral AI. (2024). *API Documentation: Chat Completion and Embeddings*. Retrieved from https://docs.mistral.ai/api/#:~:text=%2A%20,0
20. Mistral AI. (2024). *Using Chat Completion and Embeddings: Documentation Overview*. Retrieved from https://docs.mistral.ai/api/#:~:text=Our%20Chat%20Completion%20and%20Embeddings,learn%20how%20to%20use%20it
21. Mistral AI. (2024). *Mistral API Documentation*. Retrieved from https://docs.mistral.ai/api/#:~:text=%2A%20
22. Reddit. (2024). *How to Reasonably Secure an API Key*. Retrieved from https://www.reddit.com/r/GoogleAppsScript/comments/m730z4/how_to_reasonably_secure_an_api_key/#:~:text=Reddit%20www,coded%20and
23. Google Workspace Updates. (2017). *Embed HTML and JavaScript in New Google Sites*. Retrieved from https://gsuiteupdates.googleblog.com/2017/12/embed-html-and-javascript-in-new-google-sites.html#:~:text=We%20recently%20made%20it%20possible,code%20directly%20into%20your%20sites
24. SiliconANGLE. (2023). *Mistral AI Releases Open Source Language Model with 7B Parameters*. Retrieved from

https://siliconangle.com/2023/09/27/mistral-ai-releases-open-source-language-model-7b-parameters/#:~:text=on%20GitHub%20under%20an%20Apache,both%20research%20and%20commercial%20purposes

25. SiliconANGLE. (2023). *Mistral 7B: Efficiency and Comparison to Llama 2*. Retrieved from https://siliconangle.com/2023/09/27/mistral-ai-releases-open-source-language-model-7b-parameters/#:~:text=The%20company%20claims%20that%20Mistral,a%20predecessor%20to%20Llama%202
26. SiliconANGLE. (2023). *Mistral 7B: The First Open Source LLM Designed for Enterprises*. Retrieved from https://siliconangle.com/2023/09/27/mistral-ai-releases-open-source-language-model-7b-parameters/#:~:text=Mistral%207B%20is%20the%20first,neural%20networks%20for%20the%20enterprises
27. Conductor Academy. (2024). *Google Lighthouse: Open Source Tool for Improving Web Quality*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=Lighthouse%20is%20an%20open%20source,Accessibility%2C%20Best%20Practices%20and%20SEO
28. Conductor Academy. (2024). *SEO and Google Lighthouse*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=SEO
29. Conductor Academy. (2024). *Performance Metrics in Google Lighthouse*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=Performance
30. WP Rocket. (2024). *Serve Static Assets with an Efficient Cache Policy*. Retrieved from https://wp-rocket.me/google-core-web-vitals-wordpress/serve-static-assets-with-an-efficient-cache-policy/#:~:text=Implementing%20browser%20caching%20will%20help,web%20browser%20of%20your%20visitors

31. WP Rocket. (2024). *Optimizing Web Page Speed and Reducing Bandwidth Usage*. Retrieved from https://wp-rocket.me/google-core-web-vitals-wordpress/serve-static-assets-with-an-efficient-cache-policy/#:~:text=Speeding%20up%20your%20web%20page,bandwidth%20usage%20and%20server%20load
32. Conductor Academy. (2024). *Google Lighthouse Accessibility Audits: Improving User Experience for Visually Impaired*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=Lighthouse%E2%80%99s%20accessibility%20audits%20examine%20how,readers%20for%20visually%20impaired%20users
33. Conductor Academy. (2024). *The 16 Best Practices Tested in Google Lighthouse Audits*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=The%2016%20Best%20Practices%20tested,write%28%29%2C%20or%20incorporating%20antiquated%20APIs
34. Conductor Academy. (2024). *Exploring SEO Opportunities with Google Lighthouse Audits*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=Lighthouse%20runs%20various%20tests%20to,which%20should%20certainly%20be%20explored
35. Conductor Academy. (2024). *Google Lighthouse Audits and Search Engine Crawling*. Retrieved from https://www.conductor.com/academy/glossary/google-lighthouse/#:~:text=Lighthouse%20currently%20runs%2013%20audits,crawled%20by%20search%20engine%20bots
36. Міністерство освіти і науки України. (2024). *Освітній чат-бот*. Retrieved from https://mon.gov.ua/osvita-2/tsifrova-transformatsiya-osviti-i-nauki/osvitniy-chat-bot(Ministry of Finance of Ukraine)
37. ITC.ua. (2024). *Чат-боти — останні статті і новини*. Retrieved from https://itc.ua/ua/tag/chat-boti-ua/(ITC.ua)

38. SendPulse UA. (2024). *Як створити чат-бот для сфери освіти*. Retrieved from https://sendpulse.ua/blog/chatbot-for-education-sphere(SendPulse)
39. UKR.NET. (2025, April). *X інтегрує свій AI-чатбот Grok з Telegram*. Retrieved from https://www.ukr.net/news/details/technologies/110399182.html(https://www.ukr.net/)
40. AIN.ua. (2024). *Чат-боти – новини та статті*. Retrieved from https://ain.ua/tag/chatbots/(Ain)
41. Osvitoria. (2024). *Безпека, навчання, здоров'я: 13 корисних чат-ботів під час війни*. Retrieved from https://osvitoria.media/opinions/bezpeka-navchannya-zdorov-ya-13-korysnyh-chat-botiv-pid-chas-vijny/(Освіторія Медіа)
42. Teach-Hub. (2024). *Чат-боти для навчання: огляд найпопулярніших та особливості використання*. Retrieved from https://teach-hub.com/chat-boty-dlia-navchannia-ohliad-naypopuliarnishykh-ta-osoblyvosti-vykorystannia/(TeachHub)

ДОДАТКИ

ДОДАТОК А. Код Google Apps Script.

```
function callMistralAPI(userMessage) {
  const apiKey = 'y6j4OGKr0GGn62BIQaEhyBgpDT1LiTpZ';
  const agentId = 'ag:ceb0c73c:20250330:untitled-agent:69129147'; // Ваш агент ID

  const url = "https://api.mistral.ai/v1/agents/completions";
  const payload = {
    "agent_id": agentId,
    "messages": [{
      "role": "user",
      "content": userMessage // Повідомлення користувача
    }]
  };

  const options = {
    "method": "post",
    "contentType": "application/json",
    "headers": {
      "Authorization": "Bearer " + apiKey,
      "Accept": "application/json"
    },
    "payload": JSON.stringify(payload),
     "muteHttpExceptions": true // Щоб побачити повну відповідь від сервера при помилці
  };

  try {
    const response = UrlFetchApp.fetch(url, options);
    const jsonResponse = JSON.parse(response.getContentText());
```

```
    // Перевірка наявності даних у відповіді
    if (jsonResponse.choices && jsonResponse.choices.length > 0) {
      const reply = jsonResponse.choices[0].message.content;
      return reply;
    } else {
       return 'No valid response from Mistral API. Please check the request.';
    }
  } catch (e) {
    Logger.log('Error: ' + e);
    return 'Error occurred while fetching response from Mistral API: ' + e.message;
  }
}
```

Функція для обробки GET-запитів

```
function doGet(e) {
  const userMessage = e.parameter.message || 'Hello World'; // Повідомлення за замовчуванням
  const response = callMistralAPI(userMessage);

  Формуємо відповідь у вигляді JSON
  const jsonOutput = {
    "response": response
  };

  // Повертаємо результат у форматі JSON
  return ContentService.createTextOutput(JSON.stringify(jsonOutput))
    .setMimeType(ContentService.MimeType.JSON);}
```

ДОДАТОК Б. HTML та JS код

```
<div style="text-align: center;">

    <h1 style="color: #000000; font-family: Lato, sans-serif; font-size: 64pt; font-weight: 300; letter-spacing: 0.5px; line-height: 1.2; margin-top: 30px;">Запитайте у чат бота</h1>


    <input type="text" id="userMessage" placeholder="Field for entering a question" style="display: block; width: 80%; padding: 10px; margin: 10px auto;">


    <button onclick="sendMessageToMistral()" style="display: block; width: 80%; padding: 10px 20px; margin: 10px auto;">Send</button>


    <div id="response" style="margin-top: 20px; padding: 10px; text-align: left; font-family: Arial; font-size: 11pt; font-weight: 400; color: #000000;"></div>

</div>


<script>

    function sendMessageToMistral() {

        const message = document.getElementById('userMessage').value;

        const scriptURL = 'https://script.google.com/macros/s/AKfycbx2DZDMMl64PIHqdCpvuSYRyInoxNkx-Jjqd5BUNsJ2izbfIPEvPbJVQqUZBdCnoTFV/exec?message=' + encodeURIComponent(message);


        fetch(scriptURL)
```

```
            .then(response => response.json())
            .then(data => {
                document.getElementById('response').innerHTML = 'Answer: ' + data.response;
            })
            .catch(error => {
                document.getElementById('response').innerHTML = 'Something is wrong: ' + error;
            });
    }
</script>
```

ДОДАТОК В. Приклад відповіді API Mistral (JSON)

```
{
  "response": "Німеччина розпочала вторгнення в Польщу 1 вересня 1939 року. Це стало формальним початком Другої світової війни."
}
```