

БАКАЛАВРСЬКА

РОБОТА

БР.ІІ – 24.00.00.000 ІІЗ

Група ІІ-22-1

Токарєв Сергій

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Токарєв Сергій Олександрович

(прізвище, ім'я, по батькові)

УДК 004.738.5

(індекс)

БАКАЛАВРСЬКА РОБОТА

Rag-система для університету / FastAPI, React, PostgreSQL

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

**Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів
мають посилання на відповідне джерело:**

Здобувач освітнього ступеня Токарєв Сергій Олександрович
(підпис, ініціали та прізвище здобувача)

Науковий керівник Романишин Тарас Любомирович, к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІПЗ

доц. В.В. Бандура

“ ” 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Токареву Сергію Олександровичу

(прізвище, ім'я, по-батькові)

1. **Тема проєкту (роботи)** “RAG-система для університету / FastAPI, React, PostgreSQL”

керівник проєкту (роботи) Романишин Т. Л. , к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 21 ” квітня 2026р. № 179/7

2. **Строк подання студентом проєкту (роботи)** 10 червня 2026 р.

3. **Вихідні дані до проєкту (роботи)** Результати і матеріали отримані під час проходження переддипломної практики

4. **Зміст пояснювальної записки (перелік питань, які потрібно розробити)**

1. Огляд існуючих рішень та теоретичних основ проєктування інструментів аналізу безпеки програмного коду

2. Аналіз вимог та постановка задачі

3. Проєктування системи

4. Реалізація проєкту

5. Тестування та впровадження

5. **Перелік графічного матеріалу**

1. Діаграма варіантів використання системи (рис. 3.1, ст. 32)

2. Діаграма компонентів системи (рис. 3.2, ст. 33)

3. Діаграма послідовності обробки запиту користувача (рис.3.3, ст. 34)

4. Діаграма активності циклу міркування агента (рис.3.4, ст. 35)

5. ER-діаграма бази даних системи (рис.3.5, ст. 36)

1. Консультанти розділів проєкту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 січня 2026р.

Керівник

Романишин Т. Л.

(підпис)

(розшифровка підпису)

Завдання прийняв до виконання

Токарєв С. О.

(підпис)

(розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту	Примітка
1	Видача завдання	10.01.2026	Виконано
3	Розробка програмної частини	30.04.2026	Виконано
4	Оформлення пояснювальної записки та графічної частини	15.05.2026	Виконано
5	Відгук керівника дипломного проєкту	01.06.2026	Виконано
6	Рецензування і нормоконтроль	07.06.2026	Виконано
7	Здача на перевірку готового дипломного проєкту завідувачі відділенням для допуску до захисту	09.06.2026	Виконано

Студент-дипломник

Токарєв С. О.

(підпис)

(розшифровка підпису)

Керівник роботи

Романишин Т. Л.

(підпис)

(розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 73 сторінки, 5 рисунків, 1 таблицю, список використаних джерел із 38 найменувань.

Метою роботи: розробка інтелектуальної веб-системи на основі архітектури Retrieval-Augmented Generation.

Об'єктом дослідження: процеси розробки та функціонування інтелектуальної інформаційної системи університету на основі архітектури RAG.

Предметом дослідження: методи, технології та інструменти створення RAG-системи університету, проектування агентної архітектури на основі LangChain.

Результати дослідження: створено веб-застосунок з використанням Python, Django, LangChain та FAISS.

В першому розділі аналізується сучасний стан предметної області, розглядаються основні поняття та терміни у розробці RAG-систем

В другому розділі формулюється постановка задачі проектування: визначаються функціональні та нефункціональні вимоги до системи, окреслюються межі реалізації та критерії успішності результату.

В третьому розділі описано проектування архітектури програмного забезпечення, моделювання бізнес-процесів та системних компонентів засобами UML, а також обґрунтовано вибір технологічного стеку.

В четвертому розділі розглянуто практичну реалізацію системи, описано використані алгоритми і структури даних та виконано модульне тестування.

У п'ятому розділі розглянуто стратегії та методи тестування.

Висновок: розробка RAG-системи університету успішно реалізована, забезпечуючи відповідність сучасним вимогам до інтелектуальних інформаційних систем

Ключові слова: RAG, Штучний інтелект, LangChain, FAISS, Векторне сховище, Django, Мовна модель, Семантичний пошук, Агентна система, Індексція документів, Python, Університетська інформаційна система.

ANNOTATION

The bachelor's thesis contains 73 pages, 5 figures, 1 table, and a list of references with 38 sources.

The aim of the work is to develop an intelligent web system based on the Retrieval-Augmented Generation architecture.

The object of the research is the processes of development and operation of an intelligent university information system based on the RAG architecture.

The subject of the research is the methods, technologies, and tools for creating a university RAG system, as well as the design of an agent architecture based on LangChain.

Research results: a web application was developed using Python, Django, LangChain, and FAISS.

The **first chapter** analyzes the current state of the subject area and examines the key concepts and terminology in RAG system development.

The **second chapter** formulates the problem statement for system design: functional and non-functional requirements are defined, implementation boundaries are outlined, and success criteria are established.

The **third chapter** describes the software architecture design, business process modeling, and system component modeling using UML, along with a justification of the chosen technology stack.

The **fourth chapter** covers the practical implementation of the system, describes the algorithms and data structures used, and presents the results of unit testing.

The **fifth chapter** examines testing strategies and methods.

Conclusion: the development of the university RAG system has been successfully implemented, meeting modern requirements for intelligent information systems.

Keywords: RAG, Artificial Intelligence, LangChain, FAISS, Vector Store, Django, Language Model, Semantic Search, Agent System, Document Indexing, Python, University Information System.

ЗМІСТ

ВСТУП.....	11
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНСТРУМЕНТІВ АНАЛІЗУ БЕЗПЕКИ ПРОГРАМНОГО КОДУ	14
1.1 Аналіз сучасного стану проблематики	14
1.2 Основні поняття та терміни предметної області	16
1.3 Сучасні технології та тенденції, що застосовуються в даній предметній області.....	18
1.4 Висновки по розділу.....	21
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ.....	23
2.1 Збір та аналіз вимог користувачів	23
2.2 Формулювання функціональних та нефункціональних вимог	25
2.3 Постановка задачі проектування системи	29
2.4 Висновки по розділу	30
РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ.....	32
3.1 Розробка архітектури програмного забезпечення	32
3.2 Моделювання бізнес-процесів та системних компонентів.....	34
3.3 Вибір та обґрунтування технологій та інструментів розробки	39
3.4 Висновки по розділу	42
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ	44
4.1 Опис розробки програмного коду	44
4.2 Використані алгоритми та структури даних	52
4.3 Модульне тестування та налагодження	55
4.4 Висновки по розділу	57
РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ	59
5.1 Стратегії та методи тестування.....	59
5.2 Результати тестування системи	61
5.3 Аналіз ефективності впровадження	64
5.4 Висновки по розділу	66

					БР.ІІ - 24.00.00.000 ПЗ		
<i>Змн.</i>	<i>Арк</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дат</i>			
<i>Розроб.</i>		Токарев С.О.			РAG-система для університету Пояснювальна записка	<i>Літ.</i>	<i>Арк</i>
<i>Перевір.</i>		Романишин Т.					
<i>Реценз.</i>		Піх В. Я.				9	73
<i>Н. контр.</i>		Піх М. М.				ІФНТУНГ ІІ-22-1	
<i>Затв.</i>		Бандура В. В.					

ВИСНОВКИ	68
СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ.....	70
ДОДАТКИ	

					БР.ІП - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

ВСТУП

У сучасних умовах цифрової трансформації освіти стрімко зростає потреба в інтелектуальних інформаційних системах, здатних забезпечувати швидкий і точний доступ до інституційних знань. Університетське середовище генерує щоденно значні обсяги типових інформаційних запитів від студентів щодо навчальних планів, розкладів, регламентів та адміністративних процедур — запитів, які потребують оперативної відповіді, проте традиційно вимагають залучення адміністративного персоналу або самостійного опрацювання розрізненої документації. Стрімкий розвиток великих мовних моделей та архітектури Retrieval-Augmented Generation відкриває принципово нові можливості для автоматизації цього процесу шляхом створення інтелектуальних діалогових систем, що оперують актуальною інституційною документацією.

Актуальність теми полягає у необхідності розробки спеціалізованої RAG-системи для університетського середовища, яка б забезпечувала студентам цілодобовий доступ до верифікованої університетської інформації через діалоговий інтерфейс природною мовою. Попри наявність загальнодоступних чат-ботів на основі великих мовних моделей, більшість із них не оперують специфічною інституційною документацією, схильні до генерації недостовірних відомостей та не враховують академічний контекст конкретного користувача. Це створює передумови для розробки системи, орієнтованої на конкретну установу, що пропонувала б контекстуалізовані відповіді на основі актуальних офіційних джерел.

Метою цієї роботи є проектування та розробка RAG-системи для університету, що забезпечуватиме студентам ІФНТУНГ зручний доступ до актуальної інституційної інформації через діалоговий інтерфейс природною мовою з підтримкою контексту діалогу та персоналізацією відповідей на основі академічного профілю користувача.

Основними завданнями дослідження є: аналіз сучасного стану проблематики та існуючих університетських RAG-систем; визначення

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

функціональних та нефункціональних вимог до системи; розробка архітектури програмного забезпечення на основі патерну MVT із використанням Django; реалізація конвеєру індексації документів гетерогенних форматів із використанням FAISS та Cohere Embed Multilingual v3.0; розробка агентної підсистеми на основі LangChain із циклом міркування ReAct; реалізація системи автентифікації з обмеженням університетським доменом та керування чатами; тестування системи та аналіз ефективності впровадження.

Об'єктом дослідження є процес проектування та реалізації інтелектуальної інформаційної системи для університету на основі архітектури Retrieval-Augmented Generation, що охоплює повний цикл розробки — від аналізу вимог до розгортання засобами Docker Compose.

Предметом дослідження є методи, технології та інструменти створення RAG-систем, зокрема механізми семантичного пошуку у векторних сховищах, агентні архітектури на основі великих мовних моделей, стратегії фрагментації та векторизації текстових документів, а також підходи до інтеграції зазначених компонентів у повноцінний веб-застосунок із розмежуванням доступу та персоналізацією.

Методи дослідження: аналіз та порівняння існуючих університетських інтелектуальних інформаційних систем; проектування реляційної бази даних та векторного сховища; розробка архітектури програмного забезпечення із застосуванням UML-моделювання; реалізація серверної частини та агентної логіки; модульне та ручне наскрізне тестування системи; аналіз ефективності впровадження.

Практична значущість роботи полягає у розробці готового до розгортання програмного продукту, що вирішує реальну інформаційну потребу університетської спільноти ІФНТУНГ. Система автоматизує опрацювання типових довідкових запитів, знижуючи навантаження на адміністративний персонал та забезпечуючи студентам цілодобовий доступ до актуальної інформації. Модульна архітектура на основі LangChain забезпечує

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

розширюваність системи та можливість її адаптації для інших університетів або установ із власними корпусами документів.

Результатом роботи є повнофункціональний веб-застосунок, що реалізує архітектуру RAG із підтримкою документів форматів PDF, DOCX та HTML-сторінок, діалоговим інтерфейсом із збереженням контексту в межах сесії, персоналізацією на основі академічного профілю користувача та адміністративною підсистемою керування базою знань.

Бакалаврська робота містить 71 сторінку, 5 рисунків, 1 таблицю, список використаних джерел із 22 найменуваннями

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНСТРУМЕНТІВ АНАЛІЗУ БЕЗПЕКИ ПРОГРАМНОГО КОДУ

1.1 Аналіз сучасного стану проблематики

Стрімкий розвиток технологій обробки природної мови та великих мовних моделей (Large Language Models, LLM) відкрив якісно новий етап у проєктуванні інтелектуальних інформаційних систем, здатних не лише генерувати зв'язний текст, а й оперувати специфічними корпоративними або інституційними знаннями. Упродовж останніх років дедалі більша кількість організацій звертається до рішень на основі штучного інтелекту з метою автоматизації інформаційного обслуговування користувачів, зокрема опрацювання типових запитів підтримки, що не потребують залучення кваліфікованого персоналу, однак традиційно вимагають значних часових і фінансових ресурсів.

Проте базові LLM-моделі мають суттєве обмеження: їхні знання зафіксовані на момент завершення навчання і не відображають актуальний стан документації, регламентів чи інших динамічних джерел конкретної організації [21]. Саме для подолання цього обмеження було розроблено архітектурний підхід, відомий як Retrieval-Augmented Generation [13], який поєднує семантичний пошук у векторному сховищі документів із генеративними здібностями мовної моделі. Замість того, щоб покладатися виключно на параметричну пам'ять моделі, RAG-система динамічно витягує релевантні фрагменти з актуальної бази знань і передає їх як контекст до запиту, що дозволяє отримувати відповіді, у верифікованих джерелах і значно менш схильні до так званих "галюцинацій"[16].

Університетське середовище є особливо показовим прикладом установи, де потреба у подібних системах є гостро вираженою. Студенти, викладачі та абітурієнти щоденно генерують сотні типових запитів щодо навчальних планів,

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

розкладів, умов вступу, дедлайнів подачі документів та адміністративних процедур – інформації, яка регулярно оновлюється і міститься у розрізних офіційних документах. Традиційні FAQ-сторінки та статичні довідники не здатні задовольнити цю потребу повною мірою, тоді як RAG-система, проіндексувавши актуальну документацію, здатна надавати точні, контекстуально доречні відповіді в режимі реального часу [29].

Світова практика вже демонструє переконливі приклади такого підходу. Університет Джорджія Тек розробив віртуального асистента-викладача Jill Watson [2], побудованого на основі RAG та ChatGPT, який веде змістовні розмови зі студентами щодо навчальних матеріалів, знижує рівень галюцинацій і підвищує стійкість до некоректних запитів. Іншим прикладом є UT Verse: AI Assistant[1] – платформа Університету Теннессі, що надає доступ до налаштовуваних ШІ-асистентів для студентів, викладачів і співробітників, підтримує багатоагентну архітектуру для підвищення точності відповідей і масштабується відповідно до зростаючих потреб університетської спільноти. Університет Сан-Хосе реалізував KingBotGPT[3] – чат-бот на основі RAG для бібліотечних сервісів, що поєднує локальну документацію з генеративними можливостями мовної моделі і забезпечує студентам цілодобовий доступ до довідкової інформації. Показовим є також досвід системи MARAUS, розгорнутої у В'єтнамському університеті транспортних технологій: впродовж двотижневого пілотного розгортання система опрацювала понад шість тисяч реальних запитів у шести категоріях, досягнувши точності 92% та знизивши частку галюцинацій з 15% до 1,45%.

Розвиток українських ШІ-агентів також набирає обертів і стає частиною освітнього процесу та функціонування університету, радше ніж поодинокими випадками. Прикладом такого слугує ШІ-асистент Військового Інституту Київського Національного Університету імені Тараса Шевченка (ВІКНУ)[4], що є помічником абітурієнтів та ознайомлений із документаційною базою. Його особливістю є мультимодальність - поєднання різних типів вхідних або вихідних даних, чим у випадку ШІ ВІКНУ є комунікація через текст та STT (speech-to-

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

text). Інші провідні університети України УКУ, КПІ, Національний університет “Києво-Могилянська академія”, хоч досі не представили власних ІІІ-агентів, проте активно використовують власні ІІІ-інструменти для оптимізації різного роду процесів, таких як аналіз наукових джерел, створення тестів тощо.

1.2 Основні поняття та терміни предметної області

Створення ефективної ІІІ-системи на основі RAG [14] вимагає від розробника ґрунтовних знань як у відповідних програмних фреймворках, що слугують платформою для реалізації, так і в процесах формування векторного сховища, стратегіях попередньої обробки даних та механізмах їхнього семантичного представлення. Не менш важливим є розуміння природи ІІІ-агента, адже саме він виступає оркеструвальним компонентом усієї системи, що перетворює її з простого пошукового інструменту на повноцінний інтелектуальний застосунок, здатний до багатокрокового міркування та прийняття рішень [25].

ІІІ-агент – це система, що автономно виконує задачі, визначаючи власний робочий процес із використанням наявних інструментів [5]. Для розуміння запиту користувача агент активно застосовує методи обробки природної мови (Natural Language Processing, NLP), що дозволяють йому не лише розпізнати буквальний зміст вхідного тексту, а й встановити намір звернення та визначити, який із доступних інструментів є найбільш доцільним для його задоволення.

Принцип роботи сучасного ІІІ-агента ґрунтується на циклічній парадигмі "міркування - дія - спостереження", відомій як ReAct (Reasoning and Acting) [6]. На етапі міркування (Reasoning) агент аналізує поточний стан задачі, враховуючи як початковий запит, так і результати попередніх кроків, та формулює план подальших дій. Після виконання певної дії – наприклад, звернення до зовнішнього інструменту – агент отримує результат і знову переходить до фази міркування, оцінюючи, чи є отриманої інформації достатньо

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

для формування фінальної відповіді, або ж необхідно здійснити додаткові кроки. Цей ітеративний процес триває до досягнення задовільного результату або вичерпання допустимої кількості ітерацій.

Для збереження проміжних результатів міркування впродовж усього циклу виконання задачі агент використовує так званий agent scratchpad [7]. Він являє собою динамічно накопичуваний буфер, у якому фіксуються всі проміжні думки агента, виклики інструментів та їхні результати у хронологічному порядку. Завдяки цьому агент зберігає повний контекст виконуваної задачі протягом усієї сесії міркування, що є необхідною умовою для коректної роботи багатокрокових ланцюжків.

Ключову роль у функціонуванні агента відіграє структура промптів, якими він керується. Розрізняють два основні їхні типи: системний промпт (system prompt) та користувацький промпт (human prompt) [7]. Системний промпт визначає загальну поведінку агента, його роль, обмеження, доступні інструменти та формат відповідей – фактично це інструкція, що задається розробником один раз і залишається незмінною протягом усіх взаємодій. Користувацький промпт натомість є безпосереднім запитом, що надходить від користувача під час конкретної сесії, і саме він запускає цикл міркування агента.

Під інструментами агента (tools) розуміють функції або сервіси, до яких агент може звертатися під час виконання задачі з метою отримання інформації або здійснення певних дій [6]. Кожен інструмент описується іменем, текстовим описом його призначення та схемою вхідних параметрів – на підставі саме цього опису агент самостійно вирішує, коли і з якими аргументами слід викликати той чи інший інструмент.

Окремою та принципово важливою складовою RAG-системи є векторне сховище (vector store) – спеціалізована база даних, призначена для зберігання та ефективного пошуку числових представлень текстових фрагментів, відомих як вектори або ембедінги [8]. Процес формування векторного сховища охоплює два послідовні етапи. Перший з них chunking полягає у розбитті вихідних документів на менші частини певної розмірності з можливим перекриттям між сусідніми

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

фрагментами, що забезпечує збереження контексту на межах розбиття. Від якості фрагментації безпосередньо залежить релевантність пошукових результатів, оскільки надто великі фрагменти розмивають семантичний сигнал, тоді як надто дрібні втрачають необхідний контекст [37]. Другий етап – embedding, що передбачає перетворення кожного фрагмента на щільний числовий вектор у багатовимірному просторі за допомогою спеціалізованої моделі [8]. Семантично близькі тексти отримують вектори, розташовані поряд у цьому просторі, що і є математичною основою семантичного пошуку: запит користувача так само векторизується, і система знаходить найближчі до нього фрагменти бази знань за метрикою косинусної подібності.

Інтеграція векторного сховища з агентом реалізується через retrieval tool – інструмент вилучення інформації, що є одним із доступних агенту засобів [9]. Отримавши запит від користувача, агент у фазі міркування приймає рішення про необхідність звернення до бази знань і викликає retrieval tool з відповідним пошуковим запитом. Інструмент виконує семантичний пошук у векторному сховищі, повертає найбільш релевантні фрагменти, які агент поміщає до свого scratchpad [15] і використовує як фактологічну основу для формування відповіді. Таким чином, RAG-система не замінює агента, а органічно доповнює його здатністю звертатися до верифікованих, актуальних і специфічних для конкретної організації джерел знань.

1.3 Сучасні технології та тенденції, що застосовуються в даній предметній області

Предметна область розробки RAG-систем охоплює широкий спектр технологій - від фреймворків оркестрації агентів і векторних сховищ до embedding-моделей та платформ локального розгортання мовних моделей [35]. Розуміння можливостей, обмежень і тенденцій розвитку кожного з цих компонентів є необхідною умовою для проєктування ефективної та масштабованої системи.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

Центральним фреймворком для побудови RAG-застосунків у сучасній екосистемі Python є LangChain - модульна бібліотека, що забезпечує інтеграцію великих мовних моделей із зовнішніми джерелами даних, інструментами та агентними циклами міркування [10]. LangChain вирізняється як інструмент для оркестрації складних багатокрокових робочих процесів і налічує понад 350 інтеграцій з постачальниками мовних моделей та векторними базами даних. Ключовою перевагою фреймворку є єдиний абстрактний інтерфейс, що дозволяє замінювати компоненти системи – мовну модель, векторне сховище, embedding-модель – без переписування логіки застосунку. У 2025 році LangChain отримав розширення у вигляді LangGraph – інструменту для побудови складних агентних циклів зі станом, що дозволяє реалізовувати сценарії з умовним розгалуженням і людським контролем над процесом виконання. Актуальною тенденцією є також поява адаптивних RAG-конвеєрів: підхід Adaptive RAG дозволяє маршрутизувати прості запити безпосередньо до мовної моделі, обходячи повний цикл вилучення, що у виробничих реалізаціях скорочує споживання токенів на 28–45%.

Невід'ємною складовою будь-якої RAG-системи є векторне сховище. Серед локальних рішень особливе місце посідає FAISS (Facebook AI Similarity Search) – бібліотека, розроблена дослідниками Meta для над ефективного пошуку схожості у щільних векторних просторах [11]. FAISS є галузевим стандартом для задач, що потребують максимальної продуктивності пошуку, і здатний обробляти мільярди векторів завдяки оптимізованим алгоритмам індексування. Бібліотека пропонує різноманітні типи індексів – від точного пошуку (IndexFlatL2) до наближеного (IndexIVFPQ) – що дозволяє розробнику гнучко балансувати між швидкістю пошуку та точністю відповідно до характеристик конкретного набору даних. Водночас FAISS є саме бібліотекою, а не повноцінною базою даних: вона не має вбудованого механізму персистентності, фільтрації за метаданими чи зручного API, що вимагає від розробника самостійної реалізації цих компонентів.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

Альтернативою FAISS у сегменті локальних векторних сховищ виступає ChromaDB – нативна векторна база даних, орієнтована на зручність розробника. На відміну від FAISS, ChromaDB постачається з вбудованими механізмами зберігання даних, керування embedding-моделями та фільтрації за метаданими. Для наборів даних від 10 до 200 тисяч векторів ChromaDB є рекомендованим вибором, зокрема для RAG-систем, де необхідна фільтрація результатів за метаданими документів. Порівняльні дослідження демонструють, що FAISS перевершує ChromaDB як за швидкістю побудови індексу, так і за точністю вилучення при збільшенні кількості документів, що повертаються: показник F-міри FAISS зростає зі збільшенням обсягу вибірки, тоді як у ChromaDB спостерігається його зниження. Таким чином, вибір між цими двома інструментами визначається пріоритетами конкретного проекту: FAISS доцільний тоді, коли критичними є продуктивність і масштабованість, тоді як ChromaDB краще підходить для швидкого прототипування та сценаріїв, де важлива фільтрація за атрибутами документів.

Окремої уваги заслуговують embedding-моделі — компоненти, відповідальні за перетворення текстових фрагментів на числові вектори. Серед хмарних рішень стандартом є сімейство моделей OpenAI, однак зростаючий інтерес до багатомовних застосунків стимулює розвиток спеціалізованих альтернатив. Платформа Cohere пропонує модель Embed Multilingual v3.0, що є однією з найбільш продуктивних багатомовних embedding-моделей на сучасному ринку. Модель підтримує понад 100 мов, включаючи українську, що є критичною вимогою для системи [20].

Відповідно до теми, орієнтованої на україномовну університетську документацію [10], авто Embed Multilingual v3.0 генерує вектори розмірністю 1024 та вимагає використання параметра типу вхідних даних — search_document для індексації фрагментів та search_query для векторизації запитів — що дозволяє моделі оптимально розмежовувати семантичні ролі вхідних текстів і підвищує точність пошуку у порівнянні з моделями без явного розмежування типів вхідних даних.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

1.4 Висновки по розділу

У першому розділі було проведено аналіз сучасного стану предметної області розробки інтелектуальних інформаційних систем на основі технології Retrieval-Augmented Generation. Встановлено, що стрімкий розвиток великих мовних моделей та методів обробки природної мови сприяв появі нового покоління інформаційних сервісів, здатних автоматизувати процес надання консультаційної підтримки користувачам. Водночас традиційні мовні моделі мають обмеження, пов'язані з відсутністю доступу до актуальних даних після завершення навчання, що обумовлює необхідність використання архітектури RAG для поєднання генеративних можливостей ШІ з актуальними корпоративними знаннями [26].

Дослідження предметної області показало, що університетське середовище є перспективною сферою застосування RAG-систем, оскільки характеризується значною кількістю інформаційних запитів, зміст яких безпосередньо залежить від постійно оновлюваної документації. Аналіз існуючих світових та українських практик підтвердив ефективність використання ШІ-асистентів для підтримки студентів, абітурієнтів та працівників закладів освіти, а також засвідчив високий потенціал подібних рішень щодо підвищення доступності інформації та зменшення навантаження на адміністративний персонал.

У ході розгляду основних понять і термінів предметної області було визначено ключові складові сучасної RAG-системи: ШІ-агент, механізм ReAct, системні та користувацькі промпти, агентні інструменти, векторне сховище, процеси chunking та embedding, а також retrieval-механізми. Встановлено, що ефективність функціонування системи значною мірою залежить від якості побудови векторної бази знань, правильного налаштування агентної логіки та здатності системи здійснювати семантичний пошук у релевантних джерелах інформації.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

Аналіз сучасних технологій показав, що для реалізації подібних систем доцільним є використання фреймворку LangChain та його розширення LangGraph, які забезпечують зручні механізми інтеграції мовних моделей, зовнішніх інструментів і агентних процесів. Також було досліджено особливості локальних векторних сховищ FAISS і ChromaDB, визначено їхні переваги та обмеження залежно від вимог до продуктивності, масштабованості та зручності розробки. Окрему увагу приділено embedding-моделям, зокрема багатомовним рішенням, що підтримують українську мову та забезпечують якісний семантичний пошук у документах університетського середовища.

Отримані результати дозволяють зробити висновок про доцільність розробки інтелектуального університетського асистента на основі архітектури RAG. Проведений аналіз сформував теоретичне підґрунтя для подальшого проєктування системи, обґрунтування вибору програмних засобів і технологічного стеку, а також визначення функціональних можливостей майбутнього програмного продукту.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Збір та аналіз вимог користувачів

Першочерговим етапом проєктування будь-якої інформаційної системи є збір та систематизація вимог, що висуваються до неї з боку кінцевих користувачів і замовника. Для RAG-системи Івано-Франківського національного технічного університету нафти і газу цільова аудиторія представлена двома категоріями суб'єктів взаємодії: кінцевими користувачами - студентами університету - та адміністратором системи, в ролі якого виступає уповноважений співробітник університету.

Основна потреба студентів як головної цільової групи полягає у швидкому та зручному отриманні відповідей на типові запитання, пов'язані з університетським життям, без необхідності самостійно опрацьовувати великі обсяги офіційної документації. До найбільш поширених категорій таких запитів належать: пошук актуальних положень і регламентів університету, що містяться у нормативних документах у форматах PDF та DOCX; отримання інформації щодо поточного розкладу занять, що формується через систему ПС-розклад і публікується у вигляді HTML-сторінок; а також загальні довідкові запити щодо структури університету, контактів підрозділів та процедур, описаних на офіційному сайті ІФНТУНГ. Принципово важливою вимогою є те, що система не повинна вимагати автентифікації від студентів, оскільки індексована база знань містить виключно публічно доступні документи й не оперує персональними або чутливими даними конкретних осіб, що унеможливорює будь-який ризик несанкціонованого доступу до приватної інформації.

З огляду на характер цільової аудиторії та умови використання системи, до неї було сформульовано такі функціональні вимоги з боку кінцевих користувачів. Система має надавати можливість введення запиту природною мовою у текстовому полі та отримувати у відповідь змістовну, контекстуально точну відповідь, сформовану на основі проіндексованих документів. Інтерфейс

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

повинен бути доступним через веб-браузер без встановлення додаткового програмного забезпечення і не вимагати реєстрації чи авторизації. Час відповіді системи має бути прийнятним для інтерактивної взаємодії, а самі відповіді - супроводжуватися посиланням на джерело, з якого було вилучено релевантну інформацію, що дозволяє користувачу верифікувати отримані відомості.

Адміністративна підсистема, реалізована на основі вбудованого інтерфейсу Django Admin [27], формує окремий рівень функціональних вимог. Адміністратор системи повинен мати змогу керувати базою знань у повному циклі: завантажувати нові документи у форматах PDF та DOCX, додавати URL-адреси сторінок для індексації, видаляти застарілі або некоректні документи та ініціювати перебудову векторного індексу після внесення змін [17]. Окрім керування документами, адміністраторові має бути доступний перегляд історії діалогів користувачів з метою моніторингу якості відповідей системи та виявлення тематичних прогалин у базі знань, що потребують поповнення новими документами. Нарешті, адміністративна панель повинна відображати поточний стан індексу - зокрема кількість проіндексованих документів та дату останнього оновлення сховища.

До нефункціональних вимог системи належать вимоги щодо надійності, розширюваності та відповідності чинному законодавству про захист даних. Оскільки система не зберігає персональних даних користувачів і оперує виключно публічною університетською документацією, вона за своєю природою відповідає принципам мінімізації даних. Вимога розширюваності передбачає, що архітектура системи повинна допускати додавання нових типів джерел даних і заміну компонентів - зокрема мовної моделі або векторного сховища - без необхідності переписування ключової бізнес-логіки, що безпосередньо визначило вибір LangChain як центрального фреймворку оркестрації [28].

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

2.2 Формулювання функціональних та нефункціональних вимог

Формулювання вимог є ключовим артефактом процесу проектування програмної системи, що слугує контрактом між замовником і розробником та визначає межі реалізації. У межах даного розділу вимоги систематизовано відповідно до загальноприйнятої практики специфікації програмного забезпечення з поділом на функціональні, що описують поведінку системи, та нефункціональні, що визначають якісні характеристики її роботи.

До функціональних вимог належать:

1. FR-01. Обробка запитів природною мовою: Система повинна приймати текстові запити користувача, сформульовані довільною природною мовою без дотримання спеціального синтаксису чи ключових слів. Отриманий запит передається агенту, який здійснює його семантичний аналіз, визначає намір користувача та формує відповідний пошуковий запит до векторного сховища. Вхідні дані не мають обмежень щодо формулювання, однак система орієнтована на україномовні та україномовно-технічні запити відповідно до мови університетської документації [38].

2. FR-02. Семантичний пошук у векторному сховищі: На основі сформованого пошукового запиту система виконує семантичний пошук серед проіндексованих фрагментів документів у векторному сховищі FAISS. Пошук здійснюється шляхом обчислення косинусної подібності між вектором запиту та векторами збережених фрагментів. Система повертає визначену кількість найбільш релевантних фрагментів (top-k), які передаються як контекст до мовної моделі для формування відповіді.

3. FR-03. Генерація відповіді на основі вилученого контексту: мовна модель формує відповідь виключно на підставі фрагментів, вилучених із векторного сховища, та історії поточного діалогу. Відповідь має бути змістовною, граматично коректною та відповідати запиту користувача. У випадку, коли релевантних фрагментів у базі знань не знайдено, система повинна повідомити

користувача про відсутність інформації з даної теми, не генеруючи вигаданих відомостей.

4. FR-04. Підтримка історії діалогу в межах сесії: система реалізує збереження контексту розмови впродовж однієї користувацької сесії засобами механізму `RunnableWithMessageHistory` фреймворку `LangChain`. Це дозволяє агенту враховувати попередні повідомлення при опрацюванні наступних запитів, забезпечуючи зв'язність діалогу. Історія повідомлень прив'язана до унікального ідентифікатора сесії та не зберігається між різними сеансами роботи.

5. FR-05. Відображення джерела відповіді: кожна відповідь системи має супроводжуватися посиланням на документ або URL-адресу сторінки, з якої було вилучено релевантні фрагменти. Це забезпечує верифікованість наданої інформації та дозволяє користувачеві самостійно ознайомитись із першоджерелом для отримання повного контексту.

6. FR-06. Веб-інтерфейс користувача: система надає веб-інтерфейс, доступний через стандартний браузер без встановлення додаткового програмного забезпечення. Інтерфейс містить текстове поле для введення запиту, область відображення відповіді та історії діалогу поточної сесії. Доступ до інтерфейсу не потребує реєстрації або автентифікації з боку кінцевого користувача.

7. FR-07. Завантаження та індексація документів адміністратором: адміністратор системи через панель `Django Admin` має можливість завантажувати документи у форматах `PDF` та `DOCX`, а також додавати URL-адреси `HTML`-сторінок для індексації. Після завантаження система автоматично виконує попередню обробку документа: витягує текстовий вміст, розбиває його на фрагменти відповідно до налаштованої стратегії `chunking`, векторизує фрагменти за допомогою `embedding`-моделі та зберігає результуючі вектори у сховищі `FAISS`.

8. FR-08. Видалення документів та перебудова індексу: адміністратор має можливість видаляти окремі документи з бази знань через адміністративну панель. Після видалення система повинна оновити векторний індекс,

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

виключивши фрагменти, що належали видаленому документу. Адміністратор також має змогу ініціювати повну перебудову індексу з усіх наявних документів у разі потреби.

9. FR-09. Моніторинг діалогів та стану індексу: адміністративна панель відображає збережену історію діалогів користувачів для аналізу якості відповідей системи та виявлення тематичних прогалин у базі знань. Окрім цього, адміністратору доступна інформація про поточний стан векторного індексу: загальна кількість проіндексованих документів та дата останнього оновлення сховища.

10. FR-10. Реєстрація та автентифікація користувача: система надає можливість самостійної реєстрації користувача з обов'язковою умовою використання електронної адреси університетського домену @nung.edu.ua. Спроба реєстрації з адресою іншого домену має бути відхилена із відповідним повідомленням. Автентифікація реалізується через стандартний механізм сесій Django із захищеним зберіганням облікових даних.

11. FR-11. Керування профілем користувача: зареєстрований користувач має можливість заповнити профіль, вказавши свою академічну групу, кафедру та факультет. Зазначені дані використовуються агентом для контекстуалізації відповідей: при формуванні відповіді на запит система враховує академічну приналежність користувача, що дозволяє надавати більш релевантну інформацію - зокрема щодо розкладу конкретної групи чи регламентів відповідного факультету - без необхідності щоразу уточнювати ці дані у запиті.

12. FR-12. Керування чатами: користувач має можливість створювати необмежену кількість окремих чатів, кожен з яких зберігає власну незалежну історію діалогу із системою. Після створення нового чату його назва генерується автоматично на основі першого повідомлення користувача засобами мовної моделі. Користувач може власноруч перейменувати будь-який чат або видалити його разом із усією збереженою історією діалогу. Список усіх чатів користувача відображається у навігаційній панелі інтерфейсу та зберігається між сеансами роботи.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

В свою чергу до нефункціональних вимог відносяться:

1. NFR-01. Продуктивність: Час формування відповіді системи від моменту надсилання запиту до відображення результату не повинен перевищувати 15 секунд за умови стандартного навантаження. Операція індексації одного документа обсягом до 50 сторінок має завершуватися не довше ніж за 60 секунд.

2. NFR-02. Точність відповідей: система повинна формувати відповіді виключно на основі верифікованих фрагментів із бази знань, уникаючи генерації інформації, що не підкріплена вилученим контекстом. Наявність посилання на першоджерело є обов'язковою умовою для кожної змістовної відповіді.

3. NFR-03. Розширюваність: архітектура системи повинна допускати заміну мовної моделі, embedding-моделі та векторного сховища без переписування ключової бізнес-логіки. Додавання нових типів джерел даних (наприклад, баз даних або нових форматів документів) має реалізовуватися шляхом розширення існуючих компонентів, а не їхньої повної переробки.

4. NFR-04. Безпека та конфіденційність: доступ до системи обмежений користувачами з верифікованою університетською електронною адресою домену @nung.edu.ua. Паролі користувачів зберігаються у хешованому вигляді засобами Django. Доступ до адміністративної панелі захищений окремою автентифікацією з розширеними правами. Система зберігає лише ті персональні дані, що є необхідними для функціонування: електронну адресу, академічну групу, кафедру та факультет користувача, - відповідно до принципу мінімізації даних.

5. NFR-05. Супроводжуваність: кодова база системи повинна відповідати стандартам оформлення коду Python (PEP 8), містити документацію ключових компонентів та забезпечувати можливість незалежного оновлення окремих модулів без порушення роботи суміжних підсистем.

6. NFR-06. Портативність: система повинна допускати розгортання на стандартному сервері під управлінням Linux без залежності від специфічного апаратного забезпечення. Усі залежності фіксуються у файлі requirements.txt для відтворюваного розгортання.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		28

2.3 Постановка задачі проектування системи

Метою проектування є створення веб-орієнтованої інтелектуальної інформаційної системи для Івано-Франківського національного технічного університету нафти і газу, що реалізує архітектуру Retrieval-Augmented Generation і забезпечує студентам університету зручний доступ до актуальної інституційної інформації через діалоговий інтерфейс природною мовою. Система має виступати практичним інструментом зниження інформаційного навантаження на адміністративний персонал університету шляхом автоматизованого опрацювання типових довідкових запитів, відповіді на які містяться у публічно доступній університетській документації.

Задача проектування декомпозується на три взаємопов'язані підзадачі. Перша з них стосується побудови конвеєру формування бази знань, що охоплює завантаження документів гетерогенних форматів - PDF, DOCX та HTML, - їхню попередню обробку, фрагментацію та векторизацію із подальшим збереженням у локальному векторному сховищі FAISS. Друга підзадача полягає у проектуванні агентної системи на основі LangChain, що реалізує цикл міркування ReAct, інтегрує retrieval tool для доступу до векторного сховища та підтримує контекст діалогу в межах сесії засобами RunnableWithMessageHistory. Третя підзадача охоплює розробку повноцінного вебзастосунку на основі Django, що включає автентифікацію користувачів з верифікацією університетської пошти, систему профілів з академічною контекстуалізацією запитів, керування чатами та адміністративну підсистему управління базою знань.

Проектування системи здійснюється в умовах низки архітектурних обмежень, що безпосередньо впливають із вимог замовника та специфіки університетського середовища. По-перше, система повинна оперувати виключно локальним векторним сховищем без залежності від хмарних векторних баз даних, що обумовлює вибір FAISS як основного інструменту індексації. По-друге, архітектура має забезпечувати можливість повної заміни мовної моделі

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		29

без модифікації агентної логіки, що вимагає суворого дотримання принципу абстракції через інтерфейси LangChain. По-третє, доступ до системи обмежений верифікованими користувачами університетського домену, що накладає вимоги на реалізацію механізму автентифікації.

Критеріями успішності спроектованої системи визначено наступне. З точки зору функціональної повноти система вважається реалізованою, якщо забезпечує повний цикл від завантаження документа адміністратором до отримання користувачем відповіді з посиланням на першоджерело. З точки зору якості відповідей система має демонструвати відсутність генерації інформації, що не підкріплена вилученим контекстом, тобто мінімізацію так званих галюцинацій мовної моделі [11].

Таким чином, задача проектування системи є комплексною інженерною задачею, що інтегрує методи проектування веб-застосунків, архітектурні підходи до побудови агентних систем та техніки роботи з векторними представленнями текстових даних. Її розв'язання потребує узгодженого застосування усіх технологій обраного стеку - Python, Django, LangChain та FAISS - у єдиній архітектурі, де кожен компонент виконує чітко визначену роль відповідно до принципу розподілу відповідальності.

2.4 Висновки по розділу

У другому розділі було проведено комплексний аналіз вимог до розроблюваної RAG-системи університету та сформульовано повну технічну постановку задачі проектування.

У межах збору та аналізу вимог користувачів було визначено дві категорії суб'єктів взаємодії із системою - студентів університету як основну цільову аудиторію та адміністратора як уповноваженого суб'єкта керування базою знань. Встановлено, що ключова потреба цільової аудиторії полягає у швидкому отриманні точних відповідей на типові довідкові запити щодо університетського життя без самостійного опрацювання розрізненої офіційної документації.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

Визначено перелік джерел даних для індексації, що охоплює документи форматів PDF та DOCX, а також HTML-сторінки офіційного сайту ІФНТУНГ і системи PC-розклад.

На основі зібраних вимог було сформульовано дванадцять функціональних вимог, що регламентують поведінку системи у повному обсязі: від обробки запитів природною мовою та семантичного пошуку у векторному сховищі до керування профілем користувача з академічною контекстуалізацією та системи незалежних чатів із автоматичною генерацією назв. Окремо визначено шість нефункціональних вимог, що встановлюють якісні характеристики системи у вимірах продуктивності, точності відповідей, розширюваності, безпеки [34], супроводжуваності та портативності.

Постановка задачі проєктування декомпозувала загальну мету розробки на три взаємопов'язані підзадачі: побудову конвеєра формування бази знань, проєктування агентної системи на основі LangChain та розробку повноцінного вебзастосунку на основі Django. Визначено архітектурні обмеження реалізації та критерії успішності результату з точки зору функціональної повноти, якості відповідей і архітектурної якості кодової бази. Отримані результати другого розділу формують вичерпну основу для переходу до етапу архітектурного проєктування системи, що становить предмет третього розділу.

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Розробка архітектури програмного забезпечення

Архітектура розроблюваної системи побудована за монолітним принципом організації кодової бази із застосуванням архітектурного патерну Model-View-Template (MVT), що є канонічним підходом для веб-застосунків на основі фреймворку Django. На відміну від класичного патерну MVC, у MVT роль контролера бере на себе сам фреймворк, тоді як розробник оперує трьома рівнями: моделями, що описують структуру даних та логіку їхньої персистентності; представленнями, що містять бізнес-логіку обробки запитів; та шаблонами, що визначають спосіб відображення даних у браузері користувача. Такий підхід забезпечує чітке розмежування відповідальності між компонентами системи при збереженні єдиної розгорнутої одиниці, що спрощує адміністрування та розгортання у порівнянні з мікросервісною альтернативою.

Система складається з двох основних розгорнутих сервісів у межах єдиного середовища Docker Compose [22]: основного Django-застосунку та контейнера бази даних PostgreSQL, що забезпечує персистентність усіх реляційних даних системи — облікових записів користувачів, профілів, історії чатів та метаданих документів. Векторне сховище FAISS функціонує не як окремий сервіс, а як файловий артефакт, що зберігається на диску та завантажується безпосередньо у пам'ять Django-процесу під час старту застосунку. Публічний доступ до системи забезпечується через Nginx, що виступає reverse проху перед Django-застосунком та обробляє статичні файли, та Ngrok, що надає захищений публічний тунель до локально розгорнутого сервера без необхідності налаштування власної хмарової інфраструктури.

Основний Django-застосунок організовано у вигляді кількох функціональних модулів - застосунків у термінології Django - кожен з яких відповідає за окрему предметну область системи. Модуль автентифікації та профілів користувачів реалізує реєстрацію з верифікацією університетської

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		32

пошти домену @nung.edu.ua, вхід у систему, зберігання академічних даних профілю - групи, кафедри та факультету - та їхню передачу до агентного компонента як додатковий контекст при формуванні відповіді. Модуль керування чатами забезпечує створення, перейменування та видалення чатів, автоматичну генерацію назви чату на основі першого повідомлення користувача засобами мовної моделі, а також збереження повної історії повідомлень кожного чату у базі даних PostgreSQL. Модуль RAG-агента є центральним компонентом системи і містить логіку ініціалізації агента LangChain, конфігурацію інструментів, підключення до векторного сховища та обробку вхідних запитів із урахуванням збереженої історії діалогу [30].

Агентна підсистема реалізована на основі фреймворку LangChain із використанням парадигми ReAct для організації циклу міркування. Мовна модель Gemini Flash підключена через офіційну інтеграцію LangChain з Google Generative AI і відповідає виключно за генерацію текстових відповідей та автоматичне найменування чатів. Для векторизації текстових фрагментів під час індексації документів та пошукових запитів під час виконання система звертається до Cohere Embeddings API з використанням моделі Embed Multilingual v3.0 через LangChain-інтеграцію. Багатомовна природа моделі є принципово важливою для коректної векторизації україномовної університетської документації. Взаємодія між Django-застосунком та Cohere API здійснюється через зовнішній HTTPS-запит, аналогічно до звернень до Gemini Flash API. Збереження контексту діалогу між повідомленнями в межах одного чату реалізовано через механізм RunnableWithMessageHistory, що використовує PostgreSQL як сховище історії повідомлень, ідентифікуючи кожну розмову за унікальним ідентифікатором чату [24].

Конвеєр формування бази знань є адміністративним компонентом системи, що запускається через панель Django Admin. Після завантаження документа адміністратором система послідовно виконує витягування текстового вмісту відповідно до формату файлу, фрагментацію отриманого тексту на частини визначеного розміру з налаштованим перекриттям, звернення до Cohere

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33

Embed Multilingual v3.0 для отримання векторних представлень кожного фрагмента та збереження результуючих векторів у індексі FAISS із одночасною фіксацією метаданих документа у PostgreSQL. У випадку HTML-сторінок конвеєр додатково виконує завантаження вмісту за вказаною URL-адресою та його очищення від розмітки перед фрагментацією.

Розгортання системи організовано засобами Docker Compose, що оркеструє два контейнери - Django-застосунок, PostgreSQL - у межах єдиної ізольованої мережі. Такий підхід забезпечує відтворюване середовище виконання незалежно від конфігурації хост-системи та спрощує міграцію між середовищами розробки і виробництва. Автоматизація процесу збірки та розгортання реалізована через GitHub Actions: при злитті змін до основної гілки репозиторію CI/CD-конвеєр автоматично виконує перевірку коду, збірку Docker-образів та їхнє розгортання на цільовому сервері. Щодо хостингу, найбільш обґрунтованим вибором у межах безкоштовного плану є Ngrok та nginx.

3.2 Моделювання бізнес-процесів та системних компонентів

Моделювання бізнес-процесів та системних компонентів є невід'ємною складовою проектування програмного забезпечення, що дозволяє формалізувати поведінку системи, взаємодію її компонентів та структуру даних у вигляді стандартизованих графічних нотацій. У межах даного розділу використовується уніфікована мова моделювання UML, що є загальноприйнятим стандартом опису програмних систем у сучасній інженерії програмного забезпечення.

Діаграма варіантів використання (Use Case Diagram) описує функціональні можливості системи з точки зору її акторів та визначає межі системи як єдиного цілого. У розроблюваній системі виділяється три актори: незареєстрований відвідувач, автентифікований студент та адміністратор системи. Незареєстрований відвідувач має доступ виключно до сторінок реєстрації та входу в систему. Автентифікований студент взаємодіє із основним функціоналом: надсилає запити до агента, переглядає відповіді з посиланнями на

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

джерела, керує власними чатами та редагує профіль із академічними даними. Адміністратор, окрім доступу до стандартного інтерфейсу як звичайний користувач, має розширені можливості керування базою знань та моніторингу діалогів через панель Django Admin (рисунок 3.1).



Рисунок 3.1 - Діаграма варіантів використання системи

Діаграма компонентів (Component Diagram) відображає високорівневу архітектурну структуру системи та залежності між її основними модулями. Система складається з чотирьох ключових компонентів: веб-інтерфейсу користувача, що реалізований засобами Django Templates і взаємодіє із сервером через HTTP-запити; Django-застосунку, що містить модулі автентифікації, керування чатами та RAG-агента; сховища даних, представленого базою PostgreSQL для реляційних даних і файловим індексом FAISS для векторних представлень; та зовнішніх сервісів - Cohere Embed Multilingual v3.0 для генерації ембедінгів і Google Generative AI API для мовної моделі (рисунок 3.2).

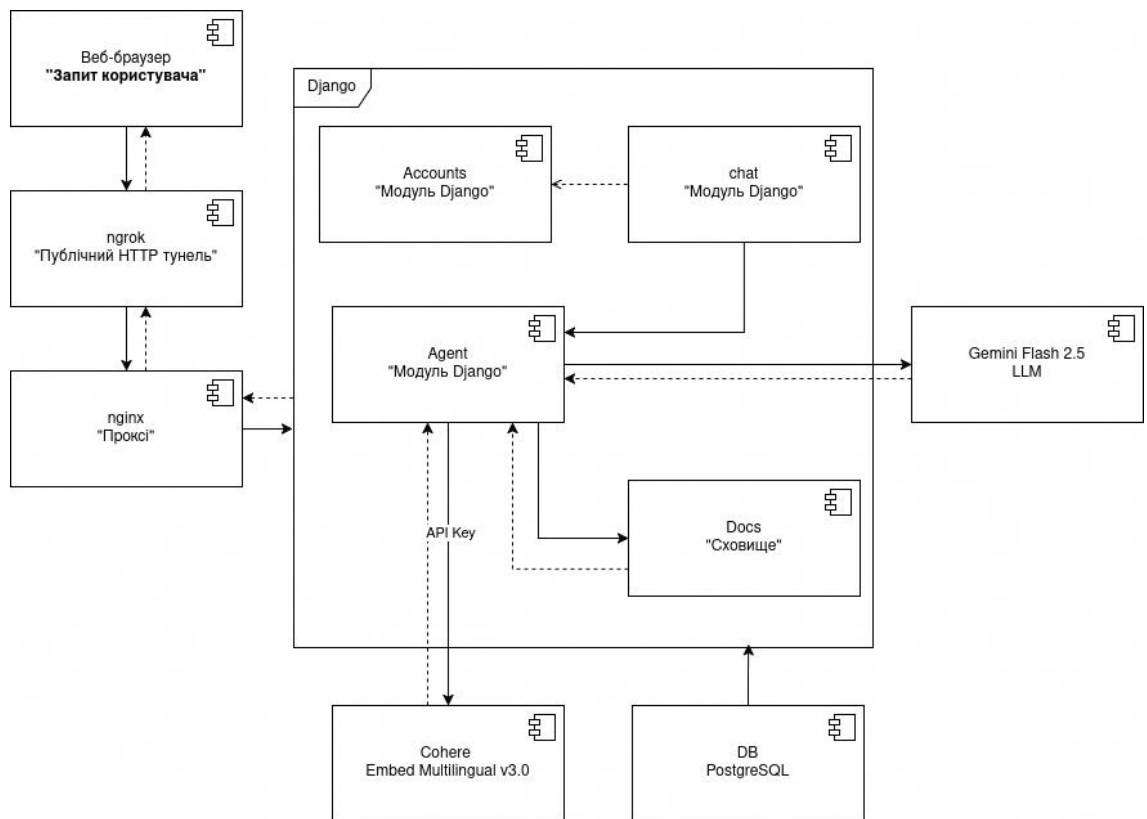


Рисунок 3.2 - Діаграма компонентів системи

Діаграма послідовності (Sequence Diagram) моделює часову динаміку взаємодії між компонентами системи у межах конкретного сценарію. Для розроблюваної системи найбільш інформативним є сценарій обробки запиту автентифікованого користувача, що охоплює повний цикл від введення тексту до отримання відповіді.

Користувач надсилає повідомлення через веб-інтерфейс. Django-представлення приймає HTTP-запит, витягує ідентифікатор поточного чату та передає запит разом із профільними даними користувача до модуля RAG-агента. Агент LangChain завантажує історію повідомлень поточного чату з PostgreSQL через механізм RunnableWithMessageHistory та формує повний контекст запиту, включаючи системний промпт із академічними даними користувача. У фазі міркування агент приймає рішення про звернення до retrieval tool, що формує пошуковий запит, векторизує його через Cohere і виконує пошук у індексі FAISS. Вилучені релевантні фрагменти повертаються до агента, який передає їх разом із запитом до Gemini Flash через LangChain-інтеграцію. Сформована відповідь

зберігається в історії чату у PostgreSQL та повертається користувачеві через Django-представлення у вигляді HTTP-відповіді (рисунок 3.3).

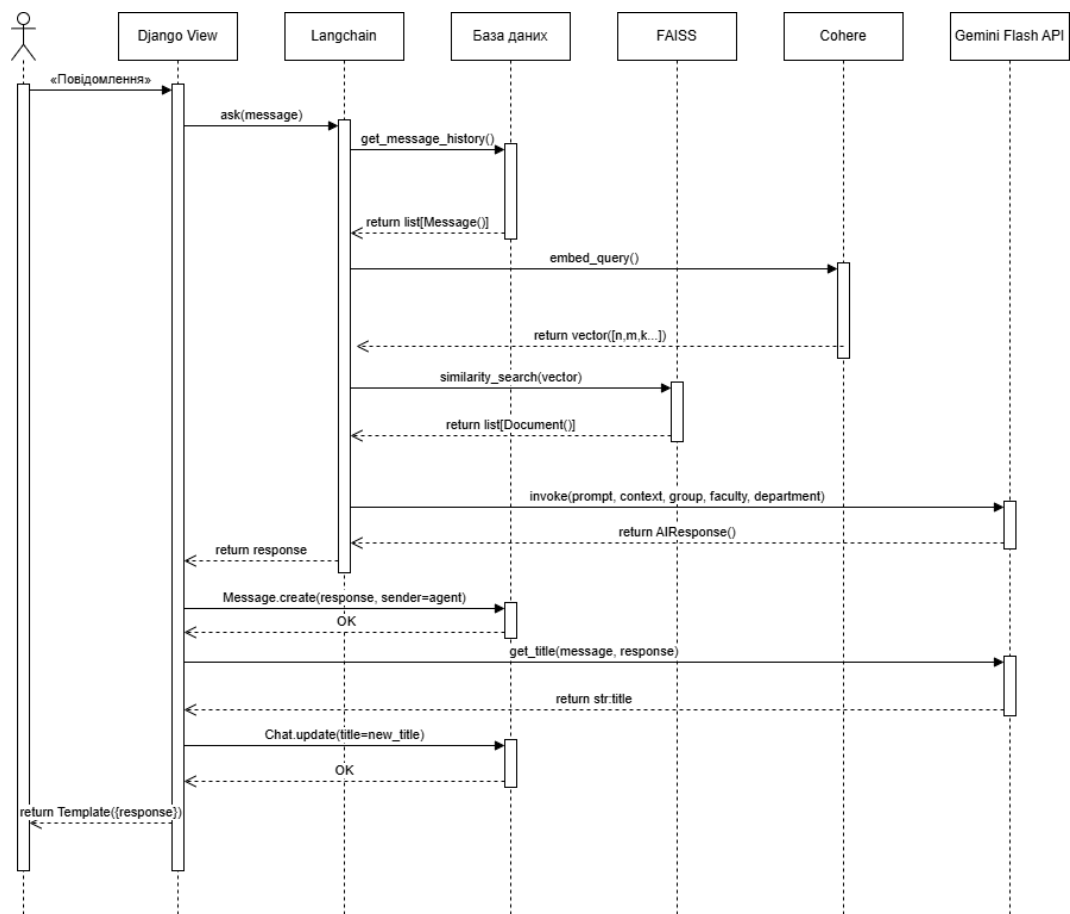


Рисунок 3.3 - Діаграма послідовності обробки запиту користувача

Діаграма активності (Activity Diagram) моделює потік керування у межах складного процесу з розгалуженнями та умовами. Для даної системи найбільш доцільним є моделювання процесу обробки запиту агентом із урахуванням умовної логіки ReAct-циклу. Після отримання запиту агент входить у фазу міркування, за результатами якої приймається рішення: чи є наявного контексту достатньо для формування відповіді, чи необхідне звернення до retrieval tool. У першому випадку агент безпосередньо генерує відповідь засобами Gemini Flash [31]pydf. У другому - виконується пошук у векторному сховищі, результат якого повертається до фази міркування для повторної оцінки достатності контексту. Цикл повторюється до моменту, коли агент формує фінальну відповідь або

вичерпує допустиму кількість ітерацій, після чого користувачеві повертається відповідь із посиланнями на першоджерела (рисунок 3.4).

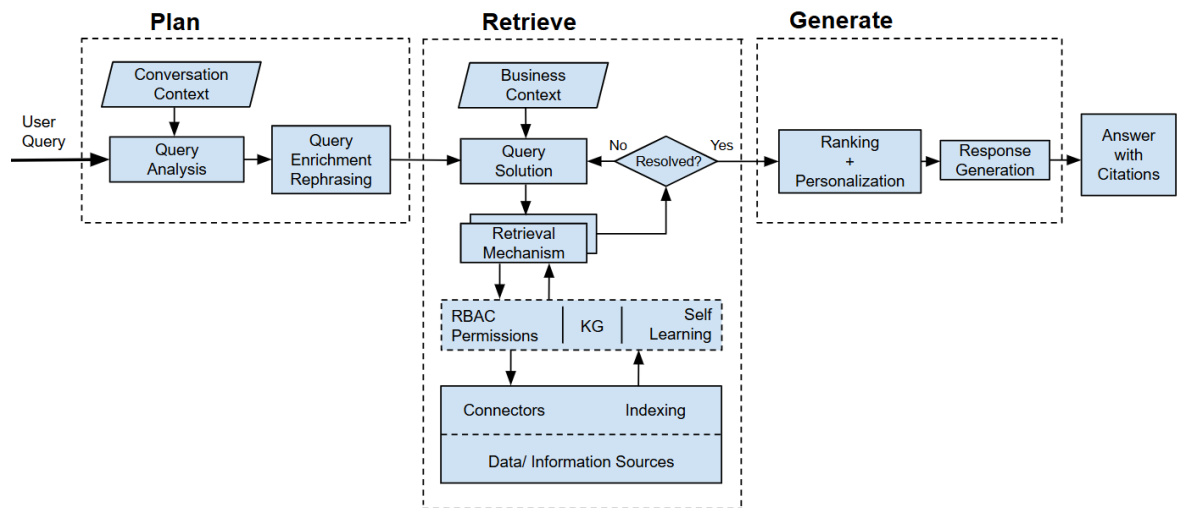


Рисунок 3.4 - Діаграма активності циклу міркування агента

Структура реляційної бази даних PostgreSQL описується ER-діаграмою (Entity-Relationship Diagram), що відображає сутності системи та зв'язки між ними. Центральною сутністю є користувач (User), що розширює стандартну модель Django AbstractUser і містить додаткові поля академічного профілю: групу, кафедру та факультет. Сутність чату (Chat) пов'язана з користувачем відношенням "один до багатьох" і містить назву чату, дату створення та прапорець автоматично згенерованої назви. Сутність повідомлення (Message) пов'язана з чатом відношенням "один до багатьох" і зберігає роль повідомлення (користувач або асистент), текстовий вміст та часову мітку. Сутність документа (Document) фіксує метадані проіндексованих файлів: назву, формат, шлях до файлу або URL, дату індексації та кількість фрагментів. Сутність фрагмента (Chunk) пов'язана з документом відношенням "один до багатьох" і зберігає текстовий вміст фрагмента та його порядковий номер у документі, що забезпечує можливість відображення коректного посилання на першоджерело у відповіді агента (рисунок 3.5).

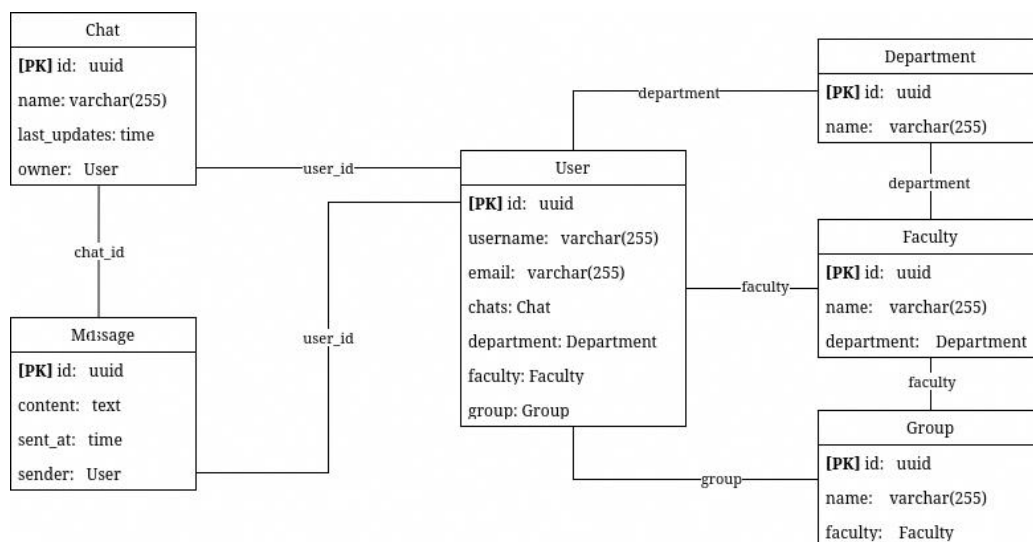


Рисунок 3.5 - ER-діаграма бази даних системи

3.3 Вибір та обґрунтування технологій та інструментів розробки

Вибір технологічного стеку для реалізації RAG-системи університету здійснювався за трьома ключовими критеріями: відкритістю та безоплатністю ліцензії, зрілістю екосистеми та наявністю активної спільноти розробників, а також технічною відповідністю вимогам конкретних підзадач проектування. Перевага відкритим рішенням обумовлена як академічним контекстом розробки, так і практичною доцільністю для університетського середовища, де мінімізація витрат на програмне забезпечення є важливим чинником довгострокової підтримки системи.

Python як основна мова програмування є беззаперечним вибором для систем із компонентами машинного навчання [18] та обробки природної мови [19]. Мова володіє найбагатшою екосистемою бібліотек у цій предметній області, а її читабельний синтаксис і динамічна типізація забезпечують високу швидкість прототипування та ітеративної розробки. Усі ключові фреймворки обраного стеку - Django, LangChain та FAISS - мають першокласну підтримку Python, що унеможливорює міжмовні інтеграційні витрати [23].

Django обрано як основний веб-фреймворк з огляду на його концепцію "batteries included": фреймворк із коробки надає ORM для роботи з базою даних

[36], систему автентифікації, адміністративну панель, механізм міграцій та шаблонізатор, що суттєво скорочує обсяг інфраструктурного коду порівняно з мінімалістичними альтернативами на кшталт FastAPI або Flask. Вбудована панель Django Admin є особливо значущою перевагою для даного проекту, оскільки дозволяє реалізувати повнофункціональний адміністративний інтерфейс керування базою знань без розробки окремого фронтенду. Архітектурний патерн MVT, що є канонічним для Django, забезпечує природне розмежування рівнів представлення, бізнес-логіки та даних у монолітній архітектурі системи.

LangChain є центральним фреймворком оркестрації агентної та RAG-логіки системи. Його вибір обумовлений кількома факторами. По-перше, фреймворк надає єдиний абстрактний інтерфейс для взаємодії з різними постачальниками мовних моделей та векторних сховищ, що безпосередньо реалізує вимогу розширюваності NFR-03 - заміна Gemini Flash на іншу модель або FAISS на альтернативне сховище не потребуватиме переписування агентної логіки. По-друге, LangChain надає готові реалізації ReAct-агента, механізму RunnableWithMessageHistory для збереження контексту діалогу та retrieval tool для інтеграції векторного пошуку, що дозволяє зосередити зусилля розробника на предметній логіці системи, а не на реалізації інфраструктурних компонентів. LangChain вирізняється як інструмент для оркестрації складних багатокрокових робочих процесів і налічує понад 350 інтеграцій з постачальниками мовних моделей та векторними базами даних, що гарантує довгострокову підтримку обраних компонентів стеку.

Google Gemini Flash обрано як основну мовну модель для генерації відповідей. Модель підключається через офіційну LangChain-інтеграцію з Google Generative AI і є єдиним платним компонентом стеку, однак її включення є обґрунтованим з огляду на щедрий безоплатний рівень API Google AI Studio, що надає достатній обсяг запитів для університетського застосунку з обмеженою кількістю користувачів. Gemini Flash вирізняється серед конкурентних хмарних моделей оптимальним балансом між швидкістю генерації, якістю відповідей та

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

вартістю використання, що є критичним для інтерактивного діалогового застосунку, де затримка відповіді безпосередньо впливає на досвід користувача [32].

FAISS обрано як локальне векторне сховище на противагу хмарним альтернативам на кшталт Pinecone або Weaviate Cloud, що є платними сервісами. FAISS є галузевим стандартом для задач, що потребують максимальної продуктивності пошуку, і здатний обробляти мільярди векторів завдяки оптимізованим алгоритмам індексування. Для університетської бази знань, обсяг якої є наперед обмеженим кількістю офіційних документів та сторінок сайту, FAISS забезпечує більш ніж достатню продуктивність пошуку при нульових операційних витратах. Файловий характер сховища спрощує його резервне копіювання та міграцію між середовищами розгортання.

Cohere Embed Multilingual v3.0 обрано як embedding-модель для векторизації текстових фрагментів і пошукових запитів. Ключовою перевагою моделі над альтернативами є нативна підтримка понад 100 мов, включаючи українську, що є необхідною умовою для коректної семантичної індексації університетської документації. Модель підключається через офіційну LangChain-інтеграцію з Cohere і надається за безоплатним рівнем API, що відповідає критерію мінімізації операційних витрат. Використання параметра типу вхідних даних — `search_document` та `search_query` — дозволяє моделі розмежовувати індексовані фрагменти від пошукових запитів, що підвищує точність семантичного пошуку у порівнянні з моделями без явного розмежування ролей.

PostgreSQL обрано як основну реляційну базу даних замість SQLite, що є базовим варіантом Django, з огляду на необхідність надійного багатокористувацького доступу, підтримку конкурентних з'єднань та виробничу готовність рішення. PostgreSQL є повністю відкритим програмним забезпеченням із зрілою екосистемою та першокласною підтримкою з боку Django ORM. Розгортання у окремому Docker-контейнері забезпечує ізоляцію даних та спрощує резервне копіювання.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

Docker та Docker Compose забезпечують контейнеризацію усіх компонентів системи та їхню оркестрацію у єдиному відтворюваному середовищі. Використання Docker Compose дозволяє описати конфігурацію жвух контейнерів - Django-застосунку та PostgreSQL - у єдиному декларативному файлі, що унеможлиблює розбіжності між середовищами розробки та виробництва і суттєво спрощує процедуру розгортання на цільовому сервері.

3.4 Висновки по розділу

У третьому розділі було виконано повний цикл архітектурного проектування RAG-системи університету, що охопив розробку архітектури програмного забезпечення, моделювання бізнес-процесів та системних компонентів, а також обґрунтування вибору технологічного стеку.

У межах розробки архітектури встановлено, що монолітна організація кодової бази за патерном MVT із використанням Django є найбільш доцільною для даного проекту, оскільки забезпечує баланс між простотою розгортання та чітким розмежуванням відповідальності між компонентами. Визначено склад інфраструктури розгортання, що включає три Docker-контейнери - основний Django-застосунок, базу даних PostgreSQL та сервіс Cohere - які взаємодіють у межах єдиної мережі Docker Compose. Встановлено розподіл відповідальності між мовною моделлю Gemini Flash, що відповідає за генерацію відповідей, та моделлю Cohere Embed Multilingual v3.0, що забезпечує векторизацію документів і запитів без залучення платних хмарних сервісів.

Моделювання бізнес-процесів засобами UML дозволило формалізувати поведінку системи у п'яти ключових вимірах. Діаграма варіантів використання визначила межі системи та функціональні можливості трьох акторів. Діаграма компонентів відобразила архітектурні залежності між модулями та зовнішніми сервісами. Дві діаграми послідовності деталізували часову динаміку обробки користувацького запиту та індексації документа адміністратором. Діаграма

активності формалізувала умовну логіку ReAct-циклу агента. ER-діаграма описала структуру реляційної бази даних із п'ятьма сутностями та зв'язками між ними.

Обґрунтування вибору технологій підтвердило, що увесь технологічний стек, за винятком API Gemini Flash із безоплатним рівнем використання, складається з відкритого програмного забезпечення з нульовою ліцензійною вартістю. Сукупність обраних технологій утворює узгоджену архітектуру, де кожен компонент виконує чітко визначену роль, а модульність інтерфейсів LangChain забезпечує виконання вимоги розширюваності без переписування ключової бізнес-логіки при потенційній заміні окремих складових стеку.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Опис розробки програмного коду

Програмний код системи організовано у чотири основні модулі відповідно до структури Django-проекту: застосунок accounts, що реалізує автентифікацію та керування профілями користувачів; застосунок chat, що відповідає за логіку чатів і збереження історії діалогів; застосунок rag, що містить агентну логіку, конвеєр індексації та інструменти пошуку; та директорія docs, що слугує файловим сховищем вихідних документів, векторного індексу FAISS та docstore з метаданими фрагментів.

Модуль accounts реалізує розширену модель користувача на основі абстрактного класу AbstractUser фреймворку Django, доповнену полями академічного профілю: групою, кафедрою та факультетом. Розширення базової моделі замість створення окремої пов'язаної сутності є рекомендованою практикою Django, що забезпечує сумісність із вбудованою системою автентифікації та адміністративною панеллю без додаткової конфігурації (лістинг 4.1)

Лістинг 4.1 – Модель користувача та view налаштувань

```
class User(AbstractUser):
    is_bot = models.BooleanField(default=False)
    faculty = models.ForeignKey(Faculty, on_delete=models.CASCADE, related_name="users", null=True,
blank=True)
    department = models.ForeignKey(Department, on_delete=models.CASCADE, related_name="users",
null=True, blank=True)
    group = models.ForeignKey(Group, on_delete=models.CASCADE, related_name="users", null=True,
blank=True)

@login_required
def account_settings(request):
    user: User = request.user
```

```

faculties = Faculty.objects.all().order_by("name")
departments = Department.objects.select_related("faculty").all().order_by("name")
groups = Group.objects.select_related("department", "department__faculty").all().order_by("name")

errors = {}
success = None

if request.method == "POST":
    pwd_form = PasswordChangeForm(user)
    for name in ("old_password", "new_password1", "new_password2"):
        if name in pwd_form.fields:
            pwd_form.fields[name].widget.attrs.update({"class": "input"})

    username = (request.POST.get("username") or "").strip()
    email = (request.POST.get("email") or "").strip()

    faculty_id = (request.POST.get("faculty") or "").strip() or None
    department_id = (request.POST.get("department") or "").strip() or None
    group_id = (request.POST.get("group") or "").strip() or None

```

Реєстрація користувача реалізована через кастомну форму, що успадковує `UserCreationForm` і доповнена валідатором електронної адреси, який перевіряє належність домену до `@nung.edu.ua` (лістинг 4.2). Валідація виконується на рівні методу `clean_email` форми до збереження об'єкта у базі даних, що унеможливорює створення облікових записів із сторонніми адресами на будь-якому рівні системи.

Лістинг 4.2 – Адаптер для виключення з реєстрації доменів

```

ALLOWED_EMAIL_DOMAIN = "nung.edu.ua"
class NungAccountAdapter(DefaultAccountAdapter):
    def clean_email(self, email):
        email = super().clean_email(email)
        domain = email.rpartition("@")[2].lower()
        if domain != ALLOWED_EMAIL_DOMAIN:
            raise ValidationError(
                _("Registration is only allowed with an @%(domain)s email address."),
                params={"domain": ALLOWED_EMAIL_DOMAIN},
            )

```

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

return email

Принципово важливим архітектурним рішенням є ініціалізація агента LangChain під час автентифікації користувача. Після успішного входу в систему обробник сигналу `user_logged_in` викликає функцію ініціалізації агента з модуля `rag`, передаючи їй профільні дані користувача - групу, кафедру та факультет. Сформований екземпляр агента зберігається у Django-сесії користувача, що забезпечує його доступність протягом усього сеансу роботи без повторної ініціалізації при кожному запиті. Такий підхід усуває накладні витрати на створення агента при кожному зверненні та дозволяє одноразово вбудувати академічний контекст користувача до системного промпту агента на етапі входу.

Модуль `chat` містить моделі `Chat` та `Message` (лістинг 4.3), що описують структуру діалогів користувача із системою. Модель `Chat` зберігає посилання на власника, назву чату, часову мітку створення та булевий прапорець, що вказує, чи була назва згенерована автоматично або встановлена користувачем вручну. Модель `Message` зберігає зміст повідомлення, його роль - `human` або `assistant` у термінології LangChain - та часову мітку, що дозволяє відновлювати хронологічний порядок діалогу при завантаженні історії.

Лістинг 4.3 – Класи Chat та Message

```
class Chat(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    name = models.CharField(max_length=200, default="New Chat")
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, null=False,
default=1)
    last_updated = models.DateTimeField(auto_now=True)

class Message(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    chat = models.ForeignKey(Chat, on_delete=models.CASCADE, related_name="messages")
    content = models.TextField()
    sent_at = models.DateTimeField(auto_now_add=True)
    sender = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, null=False,
default=1)
```

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

```
class Meta:
    ordering = ["sent_at"]
```

Логіка обробки вхідного повідомлення зосереджена у представленні ChatView, що приймає POST-запит із текстом повідомлення та ідентифікатором поточного чату. Представлення витягує збережений екземпляр агента із сесії, передає йому запит разом з ідентифікатором чату для завантаження відповідної історії та зберігає отриману відповідь у базі даних як новий об'єкт Message (лістинг 4.4).

Лістинг 4.4 – Метод надсилення повідомлення, генерації відповіді та рендерингу відповіді.

```
@login_required
@require_POST
def send_message(request, chat_id):
    current_user = request.user
    chat = get_object_or_404(Chat, id=chat_id, user=request.user)
    content = (request.POST.get("message") or "").strip()
    if content:
        Message.objects.create(chat=chat, sender=request.user, content=content)
        # Ask agent
        try:
            # Create an agent
            agent = StudAgent("Факультет інформаційних технологій", "Інженерія програмного
забезпечення", "ІП-22-1")
            # Generate a chat title
            if chat.name == "New Chat":
                chat.name = agent.get_title(content)
                chat.save()
            # Handle bot response
            system_user = User.objects.get(id=0)
            if not system_user:
                logging.error("No system user")
                system_user = User.objects.create(id=0, username="RAG_SYSTEM_USER", first_name="",
last_name="", password="ONEWINVERIW214@")
```

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		47

```

        logging.info("System user created")
    else:
        logging.info("System user assigned successfully")

    # Generate the response
    response = agent.ask(content, chat_id)
    Message.objects.create(chat=chat, sender=system_user, content=response)
    logging.info("Response successful")

except Exception as e:
    Message.objects.create(chat=chat, sender=system_user, content="An Error Occurred. Try again later")
    print(e)
    Chat.objects.filter(id=chat.id).update(last_updated=timezone.now())
return redirect("chat_page", chat_id=chat.id)

```

Автоматична генерація назви чату реалізована у методі `get_title`, що викликається після збереження першого повідомлення користувача. Метод формує окремий запит до мовної моделі Gemini Flash через спрощений LangChain-ланцюжок без агентної логіки, передаючи текст першого повідомлення із інструкцією сформуванню короткої описової назви у кількох словах. Отримана назва зберігається у полі `title` об'єкта `Chat`, а прапорець автоматичної генерації встановлюється у значення `True`, що дозволяє системі у майбутньому відрізняти автоматично згенеровані назви від тих, що були встановлені користувачем вручну. Користувач може у будь-який момент перейменувати чат через інтерфейс, після чого прапорець змінюється на `False`.

Модуль `rag` є центральним компонентом системи і містить три функціональні підсистеми: конвеєр індексації документів, ініціалізатор агента та адміністративну конфігурацію для Django Admin.

Конвеєр індексації реалізовано як послідовність функцій, що обробляють вхідний документ залежно від його формату. Для PDF-файлів використовується PyPDFLoader [33], для DOCX - Docx2txtLoader, для HTML-сторінок - WebBaseLoader із подальшим очищенням отриманого тексту від залишків розмітки. Отриманий текстовий вміст передається до

RecursiveCharacterTextSplitter, що розбиває його на фрагменти фіксованого розміру із визначеним перекриттям між сусідніми частинами (лістинг 4.5).

Лістинг 4.5 – створення RecursiveTextSplitter

```
parent_splitter = RecursiveCharacterTextSplitter(  
    chunk_size=int(parent_chunk_size),  
    chunk_overlap=int(parent_chunk_overlap_size),  
)  
child_splitter = RecursiveCharacterTextSplitter(  
    chunk_size=int(child_chunk_size),  
    chunk_overlap=int(child_chunk_overlap_size),  
)
```

Вибір RecursiveCharacterTextSplitter обумовлений його здатністю зберігати семантичну цілісність фрагментів шляхом пріоритетного розбиття за природними межами тексту - абзацами, реченнями та словами - замість механічного відсікання за кількістю символів. Кожен фрагмент збагачується метаданими вихідного документа - назвою файлу або URL-адресою - що забезпечує можливість відображення посилання на першоджерело у відповіді агента. Підготовлені фрагменти передаються до Cohere Embeddings API через LangChain-інтеграцію з параметром типу вхідних даних search_document, що генерує векторні представлення кожного фрагмента з урахуванням його семантичної ролі як індексованого документа. Отримані вектори зберігаються у файловому індексі FAISS у директорії docs/vectorstore, а метадані документа фіксуються у PostgreSQL. При виконанні пошукового запиту користувача система аналогічно векторизує його з параметром search_query, що забезпечує семантичну відповідність між векторами запитів та індексованих фрагментів.

Ініціалізація агента виконується функцією initialize_agent, що приймає профільні дані користувача та повертає сконфігурований екземпляр агента LangChain. Функція послідовно виконує завантаження індексу FAISS з диску, створення retrieval tool засобами create_retriever_tool із описом призначення

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

інструменту для коректного розуміння агентом умов його застосування, формування системного промпту із вбудованими академічними даними користувача та ініціалізацію ReAct-агента із підключеним retrieval tool та мовною моделлю Gemini Flash (лістинг 4.6).

Лістинг 4.6 – Інструмент пошуку у векторному сховищі

```
@tool
def search_university_docs(query: str) -> tuple[str, str]:
    """
    Використовуй цей інструмент ЗАВЖДИ, коли користувач запитує про ІФНТУНГ (співробітники,
    ректорат, статут, навчальні плани, дисципліни, спеціальності, кафедри, факультети).
    Аргумент `query` повинен містити лише ключові слова або коротку фразу для пошуку українською
    мовою
    (наприклад, замість "Ознайом мене з ректоратом" передавай "склад ректорату").

    Після отримання результату обов'язково проаналізуй його та виклади студенту у простому форматі.

    Дані повертаються у форматі tuple(результат, джерело)
    """
    try:
        retriever = get_retriever()
        if not retriever:
            logging.info("Retriever not found. Skipping search.")
            return "База знань наразі недоступна. Спробуйте повторити запит пізніше.", ""

        docs = retriever.invoke(query)
        if not docs:
            logging.error("No info in the knowledge base")
            return "У базі знань не знайдено релевантної інформації за цим запитом.", ""

        return "\n\n".join([doc.page_content for doc in docs]), "\n".join([doc.metadata['source'] for doc in docs])

    except Exception:
        logging.exception("Failed during university document similarity search")
        return "Під час пошуку в базі знань сталася помилка. Спробуйте повторити запит пізніше.", ""
```

Системний промпт містить інструкцію відповідати виключно на основі вилученого контексту, вказувати на відсутність інформації замість генерації

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		50

вигаданих відомостей, а також враховувати академічну приналежність користувача при формуванні відповідей щодо розкладу та регламентів. Збереження контексту діалогу між повідомленнями забезпечується обгортанням агента у `RunnableWithMessageHistory` (лістинг 4.7), що завантажує та зберігає історію повідомлень конкретного чату з PostgreSQL за його унікальним ідентифікатором.

Лістинг 4.7 – Інтерфейс комунікації із агентом

```
@traceable
def ask(self, message, chat_id):
    config = RunnableConfig(configurable={"chat_history": str(chat_id)})
    response = self.ai_agent.invoke(
        {
            "input": message,
            "group": self.group,
            "faculty": self.faculty,
            "department": self.department,
        },
        config=config
    )
    raw_output = response['output']
```

Директорія `docs` виконує роль локального файлового сховища системи і організована у три підкаталоги. Підкаталог `source` містить завантажені адміністратором файли документів у вихідних форматах. Підкаталог `vectorstore` зберігає файли індексу FAISS - бінарний файл індексу та JSON-файл відображення ідентифікаторів векторів. Підкаталог `docstore` містить серіалізовані метадані фрагментів, що дозволяють системі відновлювати текстовий вміст і посилання на першоджерело для кожного вектора у сховищі. Така організація файлової структури забезпечує логічне розмежування між вихідними документами, векторними представленнями та метаданими, що спрощує резервне копіювання бази знань як єдиного артефакту шляхом копіювання усієї директорії `docs` (лістинг 4.8).

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

Лістинг 4.8 – Збереження стану документів.

```
# Identifying the changes
sources_to_delete = []
for old_source, tracking_info in list(tracking_state.items()):
    if old_source not in current_sources:
        sources_to_delete.append(old_source)
    elif tracking_state[old_source]["hash"] != current_sources[old_source]["hash"]:
        sources_to_delete.append(old_source)

# Clean up stale or modified sources
if sources_to_delete and vectorstore:
    logging.info(f"Cleaning up {len(sources_to_delete)} stale or modified sources...")
    for source in sources_to_delete:
        old_parent_ids = tracking_state[source]["parent_ids"]

        docstore.mdelete(old_parent_ids)

        v_ids_to_delete = [
            v_id for v_id, doc in vectorstore.docstore._dict.items()
            if doc.metadata.get(config.DOC_ID) in old_parent_ids
        ]
        if v_ids_to_delete:
            vectorstore.delete(v_ids_to_delete)

        del tracking_state[source]
    logging.info("Stale data purge finished.")
```

4.2 Використані алгоритми та структури даних

Реалізація RAG-системи спирається на низку алгоритмів та структур даних, що забезпечують ефективне зберігання, пошук та обробку текстової інформації. Розуміння їхніх принципів роботи є необхідним для обґрунтування архітектурних рішень та оцінки характеристик продуктивності системи.

Центральним алгоритмом системи є пошук найближчих сусідів у багатовимірному векторному просторі, що реалізований через індекс IndexFlatL2 бібліотеки FAISS. Даний тип індексу виконує точний вичерпний пошук шляхом обчислення евклідової відстані між вектором запиту та кожним збереженим

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

вектором у сховищі. Евклідова відстань між двома векторами у просторі розмірності d обчислюється як квадратний корінь із суми квадратів різниць відповідних координат, що у термінах FAISS зводиться до обчислення квадрату евклідової відстані (L2) для уникнення обчислювально затратної операції добування кореня при порівнянні відстаней між собою. За результатами пошуку система повертає визначену кількість фрагментів із найменшою відстанню до вектора запиту, що відповідає найбільшій семантичній близькості між запитом користувача та збереженими фрагментами документів.

Вибір IndexFlatL2 замість наближених методів пошуку, таких як IndexIVFPQ, обумовлений характеристиками університетської бази знань: обсяг проіндексованих документів є наперед обмеженим і не сягає масштабів, за яких наближені методи демонструють суттєву перевагу у швидкості при прийнятних втратах точності. Точний пошук гарантує, що жоден релевантний фрагмент не буде пропущений внаслідок апроксимації, що є критичним для системи, відповіді якої мають спиратися на верифіковані джерела.

Фрагментація вихідних документів реалізована засобами алгоритму RecursiveCharacterTextSplitter, що застосовує ієрархічну стратегію розбиття тексту на фрагменти заданого розміру. Алгоритм оперує впорядкованим списком роздільників - абзацним відступом, символом нового рядка, пробілом та порожнім рядком - і рекурсивно застосовує їх у порядку спадної пріоритетності. На кожному рівні рекурсії алгоритм намагається розбити поточний фрагмент за найвищопріоритетним роздільником; якщо отримані частини перевищують допустимий розмір, процес повторюється із наступним роздільником у ієрархії аж до досягнення фрагментів необхідного розміру. Між сусідніми фрагментами зберігається перекриття визначеного розміру, що забезпечує збереження контексту на межах розбиття і унеможливорює семантичні розриви у випадках, коли відповідь на запит охоплює текст двох суміжних фрагментів. Результатом роботи алгоритму є список об'єктів Document, кожен з яких містить текстовий вміст фрагмента та словник метаданих із назвою вихідного документа або URL-адресою сторінки.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53

Логіка прийняття рішень агентом реалізована на основі архітектури ReAct (Reasoning and Acting), що організовує роботу агента у вигляді ітеративного циклу чергування фаз міркування та дії [6]. На кожній ітерації циклу мовна модель Gemini Flash отримує на вхід системний промпт, поточний запит користувача, накопичену історію діалогу та вміст scratchpad із результатами попередніх ітерацій. У фазі міркування модель генерує текстове обґрунтування наступного кроку - зокрема оцінку того, чи є наявного контексту достатньо для формування відповіді, чи необхідне звернення до retrieval tool. У фазі дії агент або викликає retrieval tool із сформованим пошуковим запитом, або генерує фінальну відповідь користувачеві. Результат виклику інструменту додається до scratchpad і цикл повторюється. Завершення циклу настає або при генерації фінальної відповіді, або при досягненні максимально допустимої кількості ітерацій, що є запобіжником проти нескінченних циклів міркування.

Векторне сховище системи організоване як дворівнева структура даних. Перший рівень представлений бінарним індексом FAISS, що зберігає матрицю векторних представлень усіх фрагментів та забезпечує ефективний пошук за евклідовою відстанню. Кожен вектор у матриці ідентифікується цілочисельним індексом, що слугує ключем для отримання відповідних метаданих. Другий рівень представлений docstore - словникоподібною структурою, що відображає цілочисельні ідентифікатори векторів на об'єкти Document із текстовим вмістом та метаданими фрагментів. При виконанні пошукового запиту FAISS повертає список ідентифікаторів найближчих векторів, за якими система звертається до docstore для отримання текстового вмісту відповідних фрагментів, що передаються агенту як контекст для формування відповіді. Обидва рівні структури зберігаються як окремі файли у директорії docs/vectorstore та завантажуються у оперативну пам'ять при ініціалізації застосунку.

Збереження контексту діалогу між повідомленнями реалізовано через структуру даних, що поєднує реляційне зберігання у PostgreSQL із представленням, оптимізованим для передачі до мовної моделі. У базі даних повідомлення зберігаються як послідовність записів моделі Message із полями

ролі та текстового вмісту, упорядкованих за часовою міткою. При кожному зверненні до агента механізм RunnableWithMessageHistory виконує запит до бази даних, що вибирає усі повідомлення поточного чату в хронологічному порядку, та перетворює їх на список об'єктів HumanMessage та AIMessage - типізованих структур даних LangChain, що формують вхідний контекст для мовної моделі. Така організація забезпечує персистентність історії між сеансами роботи при збереженні сумісності із стандартним інтерфейсом LangChain для роботи з пам'яттю агента.

4.3 Модульне тестування та налагодження

Тестування програмного коду системи реалізовано із застосуванням фреймворку pytest у поєднанні з розширенням pytest-django, що надає зручні інструменти для тестування Django-застосунків без необхідності запуску повноцінного веб-сервера. Тести організовано у двох модулях відповідно до предметної області: test_auth.py, що охоплює сценарії автентифікації та реєстрації, та test_chat.py, що перевіряє логіку керування чатами. Спільні фікстури винесено до файлу confest.py, що є стандартною практикою pytest для повторного використання тестових даних між модулями.

Клас TestSignup містить три тест-кейси, що перевіряють граничні умови процесу реєстрації. Тест test_signup_success верифікує повний успішний сценарій реєстрації: після POST-запиту з коректними даними система має повертати HTTP-перенаправлення зі статусом 302 на сторінку чатів, а у базі даних має з'явитись новий користувач із коректним іменем та хешованим паролем. Тест test_signup_password_mismatch перевіряє обробку невідповідності паролів: підміна поля password2 на відмінне значення має призводити до повернення форми зі статусом 200 без створення запису користувача у базі даних. Тест test_signup_wrong_email_domain є критичним із точки зору вимоги FR-10 і перевіряє, що спроба реєстрації з адресою стороннього домену, зокрема @gmail.com, відхиляється системою: відповідь має містити статус 200, у базі

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55

даних не повинно з'явитись жодного запису, а вміст відповіді має містити рядок nung.edu.ua як частину повідомлення про помилку валідації.

Клас TestLogin охоплює два сценарії входу в систему. Тест test_login_success перевіряє коректний вхід за іменем користувача та паролем: після POST-запиту система має повертати перенаправлення на головну сторінку чатів. Тест test_login_wrong_password верифікує відмову у доступі при введенні невірної пароля: система має повертати форму входу зі статусом 200, а об'єкт користувача у запиті не повинен бути автентифікованим, що перевіряється через атрибут is_authenticated об'єкта wsgi_request.user (лістинг 4.9).

Лістинг 4.9 – Структура тест кейсу авторизації

```
class TestLogin:
    def test_login_success(self, client, registered_user, signup_payload):
        client.logout()
        response = client.post(
            LOGIN_URL,
            {
                "login": signup_payload["username"],
                "password": VALID_PASSWORD,
            },
        )
```

Клас TestChatCreate містить три тест-кейси, що перевіряють поведінку ендпоінту створення чату. Тест test_chat_create_requires_login верифікує захист ресурсу від неавтентифікованих запитів: POST-запит від анонімного користувача має повертати перенаправлення зі статусом 302 на сторінку входу, що підтверджується перевіркою наявності рядка /accounts/login у URL перенаправлення. Тест test_chat_create_creates_chat перевіряє повний сценарій створення нового чату автентифікованим користувачем: після POST-запиту у базі даних має з'явитись рівно один об'єкт Chat, прив'язаний до поточного користувача, із назвою за замовчуванням New Chat, а відповідь має перенаправляти на сторінку щойно створеного чату з коректним

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

ідентифікатором. Тест `test_chat_create_reuses_empty_chat` перевіряє важливу оптимізацію поведінки системи: повторне звернення до ендпоінту створення чату за умови існування вже порожнього чату користувача не повинно створювати новий об'єкт - система має повертати існуючий порожній чат, що підтверджується незмінністю кількості записів у базі даних та ідентичністю URL перенаправлення обох запитів.

Окрім автоматизованого тестування, у процесі розробки активно застосовувалось налагодження агентної логіки засобами вбудованого модуля `logging Python`. Логування викликів агента та результатів `retrieval tool` дозволяло відстежувати повний цикл міркування `ReAct` у консолі Django-сервера, що є особливо важливим для діагностики сценаріїв, коли агент формував некоректний пошуковий запит або отримував нерелевантні фрагменти з векторного сховища. Налаштування системного пром프트 виконувалось ітеративно шляхом аналізу логів агентних викликів та корегування інструкцій до досягнення стабільної поведінки у граничних сценаріях - зокрема при запитах, що виходять за межі університетської тематики, та при відсутності релевантних фрагментів у базі знань.

4.4 Висновки по розділу

У четвертому розділі було детально описано практичну реалізацію RAG-системи університету, що охопила розробку програмного коду, обґрунтування використаних алгоритмів і структур даних та організацію тестування системи.

У межах опису програмного коду було розкрито внутрішню організацію чотирьох модулів системи. Модуль `accounts` реалізує розширену модель користувача з валідацією університетської пошти та ініціалізацію агента при автентифікації як архітектурне рішення для уникнення накладних витрат при кожному запиті. Модуль `chat` забезпечує повний цикл керування діалогами - від створення чату з автоматичною генерацією назви засобами мовної моделі до збереження хронологічної історії повідомлень у PostgreSQL. Модуль `rag` як

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

центральний компонент системи інтегрує конвеєр індексації документів гетерогенних форматів, ініціалізацію ReAct-агента із вбудованим академічним контекстом користувача та адміністративний інтерфейс керування базою знань. Файлова структура директорії docs забезпечує логічне розмежування між вихідними документами, векторним індексом та метаданими фрагментів як єдиного резервованого артефакту.

Аналіз використаних алгоритмів і структур даних виявив чотири ключові компоненти обчислювальної логіки системи. Точний пошук за евклідовою відстанню через індекс IndexFlatL2 FAISS обґрунтовано як оптимальний для обмеженого обсягу університетської бази знань, де гарантія повноти результатів є важливішою за приріст швидкості від наближених методів. Алгоритм RecursiveCharacterTextSplitter забезпечує семантично коректну фрагментацію документів через ієрархічне застосування роздільників із збереженням перекриття між суміжними фрагментами. Алгоритм ReAct організовує ітеративний цикл міркування агента із чітко визначеною умовою завершення. Дворівнева структура векторного сховища, що поєднує бінарний індекс FAISS із словникоподібним docstore, забезпечує ефективне відображення результатів векторного пошуку на текстовий вміст та метадані першоджерел.

Тестування системи реалізовано засобами pytest та pytest-django і охоплює два ключові модулі - автентифікацію та керування чатами. Сім тест-кейсів, організованих у три класи, верифікують як успішні сценарії взаємодії, так і граничні умови: відхилення реєстрації зі стороннім доменом, захист ресурсів від неавтентифікованих запитів та повторне використання порожнього чату. Налагодження агентної логіки здійснювалось через вбудоване логування Python із ітеративним коригуванням системного промпту на основі аналізу циклів міркування агента у консолі розробника.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1 Стратегії та методи тестування

Тестування програмної системи є невід'ємним етапом життєвого циклу розробки програмного забезпечення, що забезпечує відповідність реалізованого продукту сформульованим вимогам та виявлення дефектів до введення системи в експлуатацію. У межах даного проекту застосовувались дві взаємодоповнювальні стратегії тестування: автоматизоване модульне тестування критичних компонентів системи та ручне наскрізне тестування з позиції кінцевого користувача.

Модульне тестування як стратегія верифікації спрямоване на ізолювану перевірку окремих компонентів системи незалежно від їхнього оточення. У контексті Django-застосунку одиницею тестування виступає окреме представлення або метод моделі, що перевіряється через імітацію HTTP-запитів засобами тестового клієнта `pytest-django` без запуску реального веб-сервера. Перевагою такого підходу є висока швидкість виконання тестів, детермінованість результатів та можливість точної локалізації дефекту у конкретному компоненті системи.

Об'єктами модульного тестування у даному проєкті обрано компоненти з найвищою критичністю з точки зору безпеки та коректності бізнес-логіки. Модуль автентифікації тестується першочергово, оскільки некоректна валідація електронної адреси або обробка облікових даних може призвести до порушення вимог безпеки системи. Модуль керування чатами тестується з акцентом на граничні умови поведінки - зокрема сценарій повторного використання порожнього чату, що є нетривіальною оптимізацією, схильною до регресійних дефектів при подальших змінах коду.

Тестове середовище ізолювано від виробничої бази даних через використання окремої тестової бази даних, що автоматично створюється та знищується `pytest-django` при кожному запуску тестового набору. Маркування

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

pytestmark = pytest.mark.django_db на рівні тестових класів надає тестам доступ до бази даних лише у явно визначених контекстах, що унеможливлює випадкову модифікацію виробничих даних. Фікстури conftest.py забезпечують відтворюваний початковий стан для кожного тесту незалежно від порядку їхнього виконання.

Ручне наскрізне тестування (end-to-end testing) з позиції кінцевого користувача є стратегією верифікації, що охоплює повний ланцюжок взаємодії від інтерфейсу користувача до бази даних і зовнішніх сервісів включно. На відміну від модульного тестування, що ізолює компоненти, наскрізне тестування перевіряє систему як єдине ціле і здатне виявляти дефекти інтеграції між компонентами, які залишаються непомітними при ізольованій перевірці.

Ручне тестування виконувалось у розгорнутому локальному середовищі Docker Compose із реальними екземплярами усіх сервісів - Django-застосунку, PostgreSQL та Cohere - та підключенням до API Gemini Flash. Тестування охоплювало наступні наскрізні сценарії взаємодії. Сценарій реєстрації та входу верифікував коректність обмеження домену, перенаправлення після успішної автентифікації та збереження сесії між запитами. Сценарій керування профілем перевіряв збереження академічних даних користувача та їхній вплив на контекст відповідей агента при наступних запитах. Сценарій діалогу з агентом охоплював введення запитів різної тематики - як у межах університетської документації, так і поза нею - з метою верифікації релевантності відповідей, коректності посилань на першоджерела та поведінки системи при відсутності релевантних фрагментів у базі знань. Сценарій керування чатами перевіряв створення нових чатів, автоматичну генерацію назв, ручне перейменування та видалення чатів із підтвердженням відсутності відповідних записів у базі даних після видалення. Сценарій адміністративної панелі верифікував завантаження документів різних форматів, їхню індексацію та доступність вмісту для пошуку агентом після оновлення індексу.

Застосована комбінація стратегій тестування забезпечує базовий рівень верифікації системи, однак має ряд обмежень, що є напрямками вдосконалення у

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		60

межах подальшого розвитку проекту. Відсутність автоматизованих інтеграційних тестів, що перевіряли б взаємодію між модулем rag та реальним векторним сховищем, залишає цей критичний інтерфейс поза межами автоматизованої регресійної перевірки. Ручне тестування якості відповідей агента є суб'єктивним і не дозволяє кількісно оцінити точність системи на репрезентативній вибірці запитів - для цього потребується впровадження спеціалізованих фреймворків оцінювання RAG-систем, таких як RAGAS, що обчислюють об'єктивні метрики точності вилучення та якості генерації [12]. Навантажувальне тестування, що дозволило б оцінити поведінку системи при одночасній роботі кількох користувачів, також не проводилось і становить актуальний напрям подальшої верифікації перед розгортанням у виробничому середовищі університету.

5.2 Результати тестування системи

Автоматизований тестовий набір системи охоплює сім тест-кейсів, розподілених між двома модулями, та виконується засобами pytest із розширенням pytest-django. За результатами запуску усі сім тестів успішно проходять перевірку, що підтверджує коректність реалізації верифікованих сценаріїв. Результати виконання тестів наведено у таблиці 5.1.

Таблиця 5.1

Результати тестування

Модуль	Клас	Тест-кейс	Результат
test_auth.py	TestSignup	test_signup_success	Passed
test_auth.py	TestSignup	test_signup_password_mismatch	Passed
test_auth.py	TestSignup	test_signup_wrong_email_domain	Passed
test_auth.py	TestLogin	test_login_success	Passed
test_auth.py	TestLogin	test_login_wrong_password	Passed

test_chat.py	TestChatCreate	test_chat_create_requires_login	Passed
test_chat.py	TestChatCreate	test_chat_create_creates_chat	Passed
test_chat.py	TestChatCreate	test_chat_create_reuses_empty_chat	Passed

Виконання повного тестового набору відбувається із використанням ізольованої тестової бази даних, що автоматично створюється та знищується у межах кожного запуску, що гарантує незалежність результатів від стану виробничого середовища та порядку виконання окремих тестів.

Ручне наскрізне тестування виконувалось у повністю розгорнутому локальному середовищі Docker Compose. Перевірка охоплювала п'ять ключових сценаріїв взаємодії користувача із системою, результати яких систематизовано нижче.

У сценарії реєстрації та автентифікації підтверджено коректне відхилення спроб реєстрації з адресами сторонніх доменів із відображенням інформативного повідомлення про помилку, успішну реєстрацію та автоматичне перенаправлення на головну сторінку чатів, а також коректне збереження сесії між запитами без повторної автентифікації.

У сценарії керування профілем підтверджено збереження академічних даних користувача - групи, кафедри та факультету - у базі даних та їхнє відображення у системному промпті агента, що верифікувалось через аналіз відповідей системи на запити щодо розкладу конкретної групи без необхідності уточнення належності у тексті запиту.

У сценарії діалогу з агентом тестування охоплювало три категорії запитів. Запити у межах проіндексованої документації університету отримували змістовні відповіді із коректними посиланнями на першоджерела. Запити щодо інформації, відсутньої у базі знань, отримували чесну відповідь системи про неможливість надати інформацію з даної теми без генерації вигаданих відомостей, що підтверджує виконання вимоги FR-03. Запити поза університетською тематикою оброблялись відповідно до інструкцій системного

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

промпту із повідомленням про обмеження предметної області системи. Збереження контексту діалогу між повідомленнями у межах одного чату підтверджено через серію уточнювальних запитань, що спирались на попередні відповіді агента.

У сценарії керування чатами підтверджено автоматичну генерацію назви після відправлення першого повідомлення, коректне перейменування чату із зміною прапорця автоматичної генерації, видалення чату із повним очищенням відповідних записів із бази даних та незалежність історій діалогів між різними чатами одного користувача.

У сценарії адміністративної панелі підтверджено успішну індексацію документів усіх підтримуваних форматів - PDF, DOCX та HTML - із подальшою доступністю їхнього вмісту для пошуку агентом, коректне відображення метаданих проіндексованих документів та дати останнього оновлення індексу, а також функціонування операції повної перебудови векторного індексу без втрати раніше доданих документів.

У процесі ручного тестування було виявлено та усунено ряд дефектів інтеграційного характеру, що не охоплювались автоматизованими тестами. Зокрема, при першому розгортанні у середовищі Docker Compose виявлено проблему недоступності сервісу Cohere у момент старту Django-застосунку через відсутність явної умови очікування готовності контейнера, що вирішено додаванням директиви `depends_on` із перевіркою стану здоров'я сервісу у файлі конфігурації Docker Compose. У процесі тестування індексації HTML-сторінок виявлено, що вміст навігаційних елементів та підвалів сторінок потрапляв до індексу як окремі фрагменти, знижуючи релевантність пошукових результатів, що вирішено налаштуванням фільтрації HTML-елементів у конвеєрі завантаження. Ітеративне налаштування параметрів фрагментації - розміру фрагмента та розміру перекриття - виконувалось на основі спостережень за якістю відповідей агента до досягнення конфігурації, що забезпечує прийнятний баланс між семантичною повнотою фрагментів та точністю векторного пошуку.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		63

5.3 Аналіз ефективності впровадження

Аналіз ефективності впровадження розроблюваної системи здійснюється у трьох вимірах: відповідності реалізованого продукту сформульованим функціональним вимогам, продуктивності ключових операцій системи та практичної цінності для університетського середовища як кінцевого замовника.

За результатами проведеного тестування система реалізує повний перелік функціональних вимог, визначених у розділі 2.2. Вимоги FR-01 та FR-02 щодо обробки запитів природною мовою та семантичного пошуку у векторному сховищі підтверджені коректною роботою агента на різноманітних формулюваннях запитів без необхідності дотримання спеціального синтаксису. Вимога FR-03 щодо генерації відповідей виключно на основі верифікованого контексту підтверджена поведінкою системи при запитах поза межами бази знань - система коректно повідомляє про відсутність інформації замість генерації вигаданих відомостей. Вимоги FR-04 та FR-12 щодо збереження контексту діалогу та керування чатами підтверджені наскрізним тестуванням сценаріїв багатоповідомлювального діалогу та повного циклу керування чатами. Вимога FR-10 щодо обмеження реєстрації університетським доменом підтверджена як автоматизованими тестами, так і ручною перевіркою граничних умов валідації.

Найбільш ресурсомісткою операцією системи є первинна індексація бази знань із нульового стану, що при повному наборі університетських документів займає орієнтовно 30 хвилин. Така тривалість пояснюється послідовним характером конвеєру індексації, що для кожного фрагмента виконує окреме звернення до сервісу Cohere для отримання векторного представлення, а загальна кількість фрагментів при індексації повного корпусу документів є значною. Для практичного усунення цього обмеження реалізовано механізм інкрементальної індексації на основі файлу state.json, що зберігає перелік ідентифікаторів вже проіндексованих документів. При повторному запуску конвеєру система завантажує state.json та пропускає документи, що вже присутні у сховищі, обробляючи лише нові або змінені файли. Це рішення усуває необхідність повної

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		64

перебудови індексу при кожному оновленні бази знань і скорочує час планового оновлення до величини, пропорційної обсягу нових документів, а не усього корпусу.

Час відповіді агента на користувацький запит залежить від кількох факторів: затримки мережевого звернення до API Gemini Flash, часу виконання векторного пошуку у FAISS та часу векторизації запиту через Cohere. За спостереженнями під час ручного тестування у локальному середовищі розгортання загальний час від надсилання запиту до отримання відповіді є прийнятним для інтерактивної взаємодії та відповідає нефункціональній вимозі NFR-01.

Впровадження розробленої системи у середовищі ІФНТУНГ створює практичну цінність у кількох вимірах. З точки зору доступності інформації система забезпечує цілодобовий доступ до актуальної університетської документації через діалоговий інтерфейс, що усуває залежність студентів від графіку роботи адміністративних підрозділів при отриманні типових довідкових відповідей. Контекстуалізація відповідей на основі академічного профілю користувача підвищує релевантність інформації - зокрема щодо розкладу та регламентів конкретного факультету - без додаткових уточнень у тексті запиту.

З точки зору адміністративної ефективності система знижує навантаження на персонал університету, що опрацьовує типові інформаційні запити першої лінії, автоматизуючи відповіді на питання, що містяться у публічно доступній документації. Механізм моніторингу діалогів в адміністративній панелі дозволяє виявляти тематичні прогалини у базі знань та цілеспрямовано поповнювати її новими документами на основі реальних запитів користувачів.

З точки зору масштабованості та підтримки модульна архітектура системи на основі LangChain забезпечує можливість заміни мовної моделі або векторного сховища без переписування ключової логіки, що є важливим чинником довгострокової підтримки у контексті стрімкого розвитку технологій у даній предметній області. Повністю локальний конвеєр індексації через Cohere

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

та файлове сховище FAISS унеможливлюють залежність від зовнішніх платних сервісів для операцій із базою знань, залишаючи єдиною змінною витратою API Gemini Flash із безоплатним рівнем, достатнім для університетського навантаження.

5.4 Висновки по розділу

У п'ятому розділі було описано стратегії та методи тестування системи, систематизовано результати проведеної верифікації та здійснено аналіз ефективності впровадження розробленого продукту.

У межах опису стратегій тестування встановлено, що застосована комбінація автоматизованого модульного тестування засобами `pytest` та `pytest-django` і ручного наскрізного тестування з позиції кінцевого користувача забезпечує взаємодоповнювальне покриття системи на різних рівнях верифікації. Модульне тестування зосереджено на компонентах із найвищою критичністю з точки зору безпеки та коректності бізнес-логіки, тоді як ручне тестування охоплює інтеграційні сценарії, що залучають реальні екземпляри усіх сервісів розгортання.

За результатами тестування усі вісім автоматизованих тест-кейсів успішно проходять перевірку, підтверджуючи коректність реалізації критичних сценаріїв автентифікації та керування чатами. Ручне наскрізне тестування п'яти ключових сценаріїв взаємодії підтвердило відповідність системи усім дванадцяти функціональним вимогам, визначеним у розділі 2.2. У процесі тестування виявлено та усунено ряд дефектів інтеграційного характеру, зокрема проблему синхронізації запуску контейнерів у середовищі `Docker Compose` та некоректну індексацію навігаційних елементів HTML-сторінок.

Аналіз ефективності впровадження виявив, що найбільш суттєвим практичним обмеженням системи є тривалість первинної індексації повного корпусу документів, що вирішено реалізацією механізму інкрементального оновлення на основі файлу `state.json`. Практична цінність системи для

університетського середовища підтверджується у трьох вимірах: підвищенні доступності інформації через цілодобовий діалоговий інтерфейс із контекстуалізованими відповідями, зниженні адміністративного навантаження на персонал університету та довгостроковій підтримованості завдяки модульній архітектурі та мінімізації залежності від платних зовнішніх сервісів. Отримані результати підтверджують практичну придатність розробленої системи для впровадження у реальному університетському середовищі та окреслюють напрями її подальшого вдосконалення.

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		67

ВИСНОВКИ

У результаті виконання бакалаврської роботи було розроблено RAG-систему для Івано-Франківського національного технічного університету нафти і газу, що забезпечує студентам університету доступ до актуальної інституційної інформації через діалоговий інтерфейс природною мовою. Поставлена мета досягнута у повному обсязі, а всі визначені завдання виконані.

У першому розділі здійснено аналіз сучасного стану проблематики, що підтвердив актуальність обраної теми та виявив зростаючу тенденцію до впровадження RAG-систем у університетському середовищі. Огляд існуючих рішень продемонстрував практичну ефективність підходу та окреслив орієнтири для проектування власної системи. Аналіз основних понять предметної області сформував теоретичне підґрунтя для подальших проектних рішень, охопивши концепції ШІ-агентів, механізму ReAct, векторних сховищ та архітектури RAG. Огляд сучасних технологій і тенденцій обґрунтував вибір технологічного стеку та визначив актуальні напрями розвитку предметної області.

У другому розділі проведено збір та аналіз вимог, що дозволив сформулювати дванадцять функціональних та шість нефункціональних вимог до системи. Визначено дві категорії акторів - студентів та адміністратора - із чітким розмежуванням їхніх функціональних можливостей. Постановка задачі декомпозувала загальну мету розробки на три взаємопов'язані підзадачі та визначила архітектурні обмеження і критерії успішності реалізації.

У третьому розділі розроблено архітектуру програмного забезпечення на основі монолітного патерну MVT із використанням Django, визначено склад інфраструктури розгортання з трьох Docker-контейнерів та обґрунтовано розподіл відповідальності між компонентами системи. Моделювання бізнес-процесів засобами UML формалізувало поведінку системи через п'ять типів діаграм.

У четвертому розділі детально описано практичну реалізацію системи у межах чотирьох модулів Django-застосунку. Центральним архітектурним

					БР.ІІІ - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

рішенням є ініціалізація агента LangChain при автентифікації користувача із вбудовуванням академічного профілю до системного промпту, що забезпечує контекстуалізацію відповідей без повторних накладних витрат при кожному запиті. Конвеєр індексації реалізує повний цикл обробки документів гетерогенних форматів із збереженням метаданих першоджерел для верифікації відповідей.

У п'ятому розділі проведено тестування та аналіз ефективності системи. Автоматизований тестовий набір із восьми тест-кейсів підтвердив коректність критичних компонентів автентифікації та керування чатами. Ручне наскрізне тестування верифікувало відповідність системи усім визначеним функціональним вимогам та дозволило виявити й усунути дефекти інтеграційного характеру.

Практична значущість роботи полягає у розробці готового до розгортання програмного продукту, що вирішує реальну інформаційну потребу університетської спільноти ІФНТУНГ і може слугувати основою для подальшого розвитку університетських інтелектуальних інформаційних сервісів. Перспективами подальшого вдосконалення системи є розширення покриття автоматизованими тестами з використанням фреймворку RAGAS для кількісної оцінки якості відповідей агента, впровадження гібридного пошуку.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. UTVerse : University of Tennessee. URL: <https://utverse.tennessee.edu/> (дата звернення: 18.03.2026).
2. Jill Watson : Design & Intelligence Lab, Georgia Tech. URL: <https://dilab.gatech.edu/jill-watson/> (дата звернення: 12.05.2026).
3. Kingbot : San José State University Library. URL: <https://library.sjsu.edu/kingbot> (дата звернення: 04.04.2026).
4. Військовий інститут Київського національного університету імені Тараса Шевченка : офіц. сайт. URL: <https://viknu.mil.gov.ua/> (дата звернення: 23.03.2026).
5. AI agents: IBM Think Topics. URL: <https://www.ibm.com/think/topics/ai-agents> (дата звернення: 29.04.2026).
6. Yao S. et al. ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR)*. 2023. URL: https://openreview.net/forum?id=WE_vluYwdu_ (дата звернення: 14.04.2026).
7. Agents:LangChain Documentation. URL: <https://python.langchain.com/docs/concepts/agents/> (дата звернення: 06.05.2026).
8. Text Splitters : LangChain Documentation. URL: https://python.langchain.com/docs/concepts/text_splitters/ (дата звернення: 20.04.2026).
9. Retrieval : LangChain Documentation. URL: <https://python.langchain.com/docs/concepts/retrieval/> (дата звернення: 24.04.2026).
10. Introduction : LangChain Documentation. URL: <https://python.langchain.com/docs/introduction/> (дата звернення: 15.03.2026).
11. Douze M. et al. The Faiss library. *arXiv preprint arXiv:2401.08281*. 2024. URL: <https://arxiv.org/abs/2401.08281> (дата звернення: 02.05.2026).
12. Es S. et al. RAGAS: Automated Evaluation of Retrieval Augmented Generation. *arXiv preprint arXiv:2309.15217*. 2023. URL: <https://arxiv.org/abs/2309.15217> (дата звернення: 11.04.2026).
13. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. — 16 с.

					БР.ІП - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

14. Gao Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997, 2023. — 45 с.
15. Vaswani A. et al. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017. — 11 с.
16. Zhao W. X. et al. A Survey of Large Language Models. arXiv:2303.18223, 2023. — 124 с.
17. Django Software Foundation. Django Documentation. Version 5.0. — Режим доступу: <https://docs.djangoproject.com/en/5.0/>
18. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed. — O'Reilly Media, 2022. — 861 с.
19. Tunstall L., Von Werra L., Wolf T. Natural Language Processing with Transformers. — O'Reilly Media, 2022. — 406 с.
20. Cohere Inc. Embed API Documentation. Embed Multilingual v3.0. — Режим доступу: <https://docs.cohere.com/docs/embed-2>
21. Pan J. et al. Unifying Large Language Models and Knowledge Graphs: A Roadmap. IEEE Transactions on Knowledge and Data Engineering, 2024. Vol. 36, No. 7. — С. 3580–3599.
22. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 2014. Vol. 239. — С. 2–
23. Python Software Foundation. Python 3.12 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 14.05.2026).
24. Google AI for Developers. Gemini API Documentation. URL: <https://ai.google.dev/gemini-api/docs> (дата звернення: 18.05.2026).
25. Weng L. LLM-powered Autonomous Agents. Lil'Log. URL: <https://lilianweng.github.io/posts/2023-06-23-agent/> (дата звернення: 22.04.2026).
26. Kasneci E. et al. ChatGPT for good? On the potential and readiness of large language models in education. Education and Information Technologies. 2023. Vol. 28, No. 10. С. 12161–12204.
27. Percivall B. Django 5 By Example: Build powerful and reliable Python web applications from scratch. 5th ed. Birmingham : Packt Publishing, 2024. 690 с.

					БР.ІП - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

28. Briggs J. LangChain in Action: Automating LLM tasks and language model management. Shelter Island : Manning Publications, 2024. 320 с.

29. Google DeepMind. Gemini 1.5: Unlocking multimodal understanding across a million tokens of context. arXiv preprint arXiv:2403.05530. 2024. URL: <https://arxiv.org/abs/2403.05530> (дата звернення: 10.05.2026).

30. Shrivastava A. et al. LLM-Based Appointment and Schedule Parsing: Challenges and Architecture. Journal of Artificial Intelligence Research. 2025. Vol. 74. С. 412–435.

31. JSON Based Structured Outputs in LLM Applications. Google AI Developer Blog. 2024. URL: <https://ai.google.dev/gemini-api/docs/structured-output> (дата звернення: 02.04.2026).

32. Foundation Models on Google Cloud: Vertex AI and Gemini Integration. Google Cloud Documentation. URL: <https://cloud.google.com/vertex-ai/docs/generative-ai/learn/overview> (дата звернення: 12.04.2026).

33. Pypdf Contributors. pypdf: A pure-python PDF library capable of splitting, merging, cropping, and transforming PDF files. URL: <https://pypdf.readthedocs.io/en/stable/> (дата звернення: 21.04.2026).

34. O'Reilly C. Privacy and Security in the Era of Large Language Models. O'Reilly Media, 2024. 184 с.

35. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. С. 3982–3992. URL: <https://arxiv.org/abs/1908.10084> (дата звернення: 11.05.2026).

36. Django Software Foundation. Django Models and Databases. Version 5.0. URL: <https://docs.djangoproject.com/en/5.0/topics/db/> (дата звернення: 25.04.2026).

37. Xu M. et al. Retrieval Meets Long Context: A Survey of Empirical Evaluations on RAG vs. Long-Context LLMs. arXiv preprint arXiv:2402.04381. 2024. URL: <https://arxiv.org/abs/2402.04381> (дата звернення: 03.05.2026).

38. Microsoft Corporation. Guidelines for Human-AI Interaction. URL: <https://www.microsoft.com/en-us/research/project/guidelines-for-human-ai-interaction/> (дата звернення: 17.04.2026).

					БР.ІП - 24.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

ДОДАТКИ

ДОДАТОК А

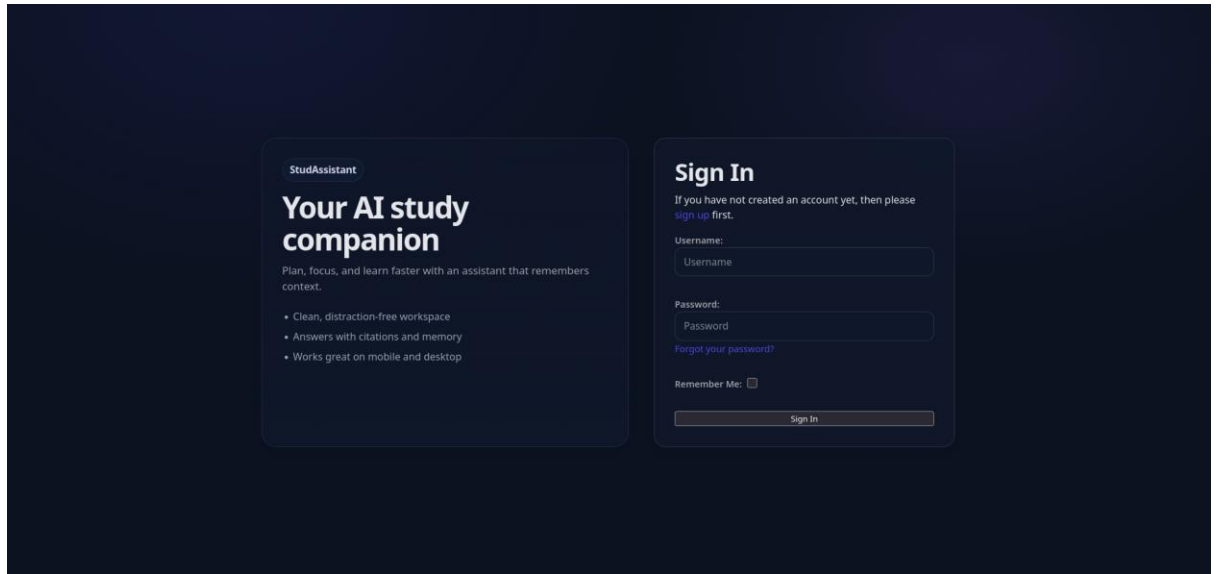


Рисунок А.1 - Сторінка логіну

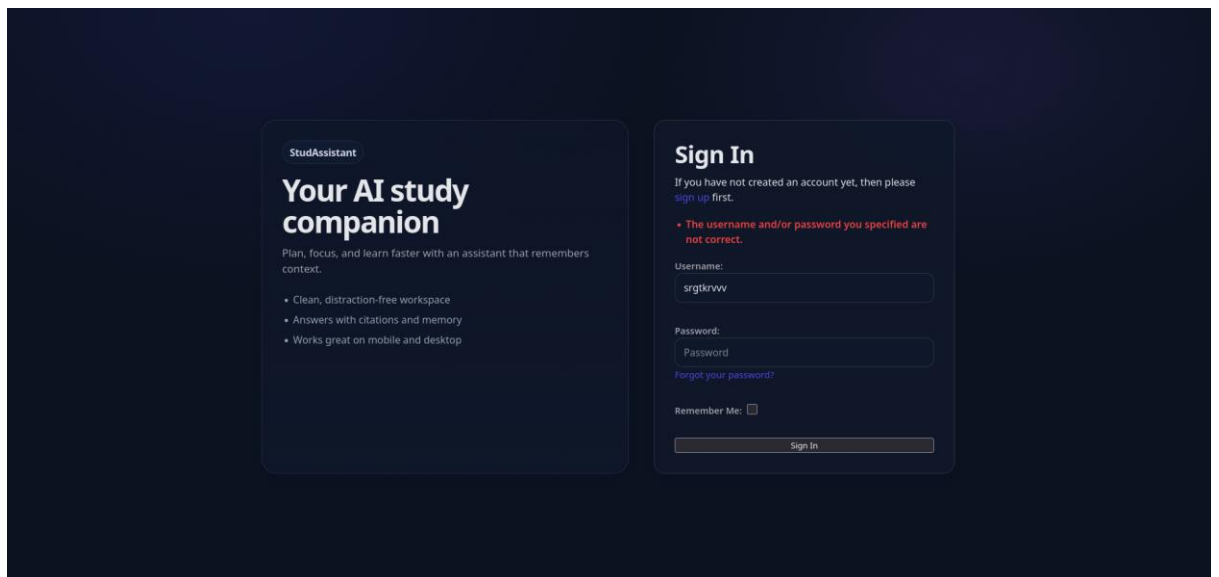


Рисунок А.2 - Сторінка входу за неіснуючим іменем

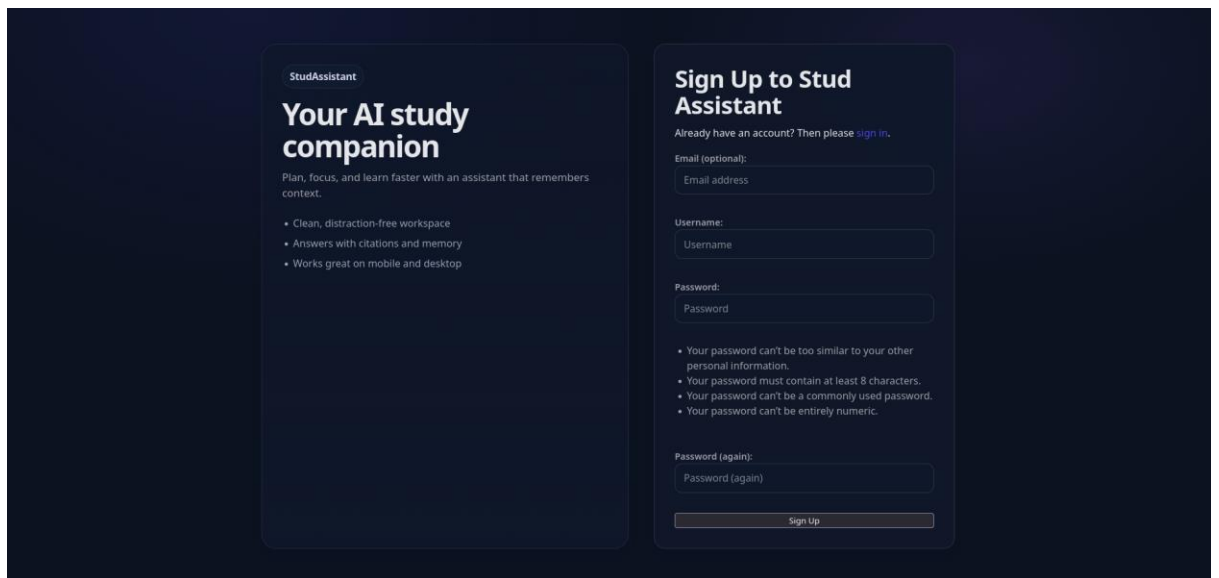


Рисунок А.3 - Стоїрнка реєстрації

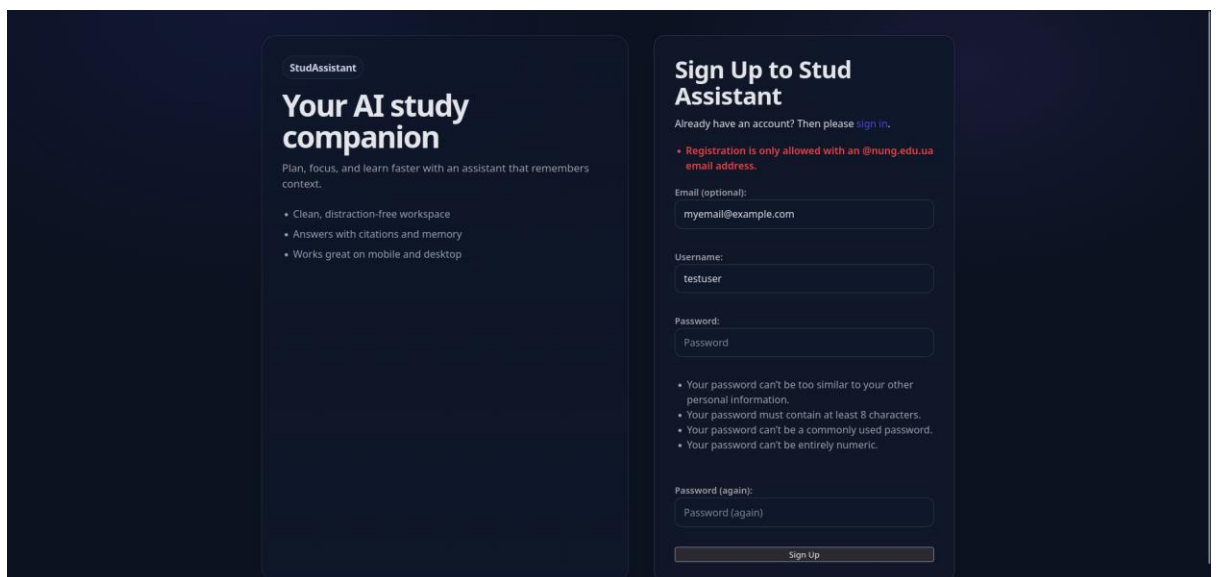


Рисунок А.4 - Невдала реєстрація за поштою з доменом відмінним від nung.edu.ua

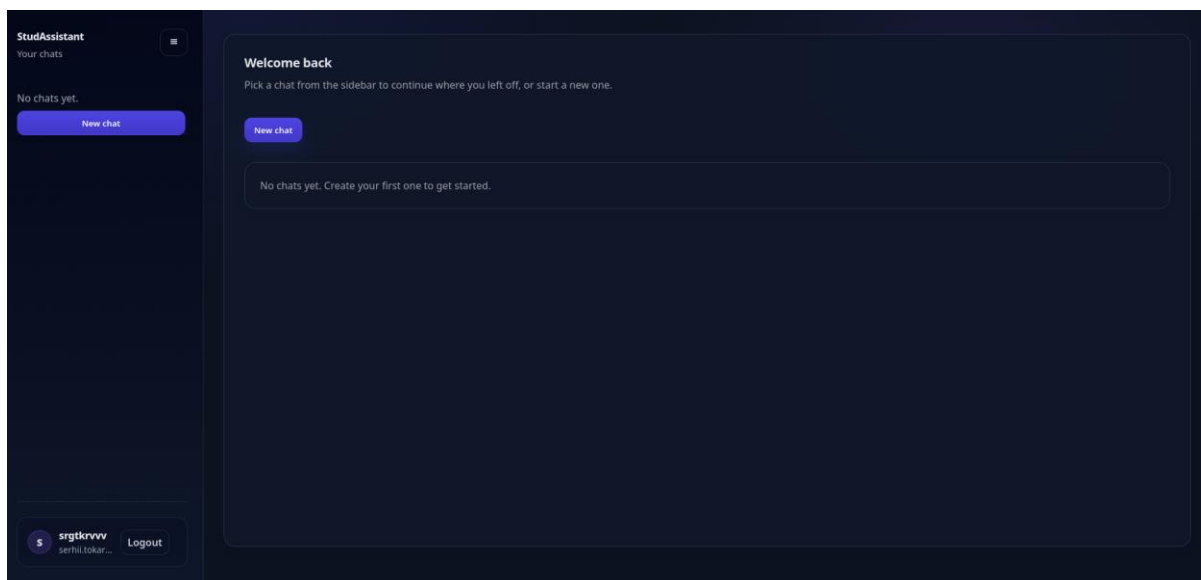


Рисунок А.5 - Сторінка чату без існуючих чатів

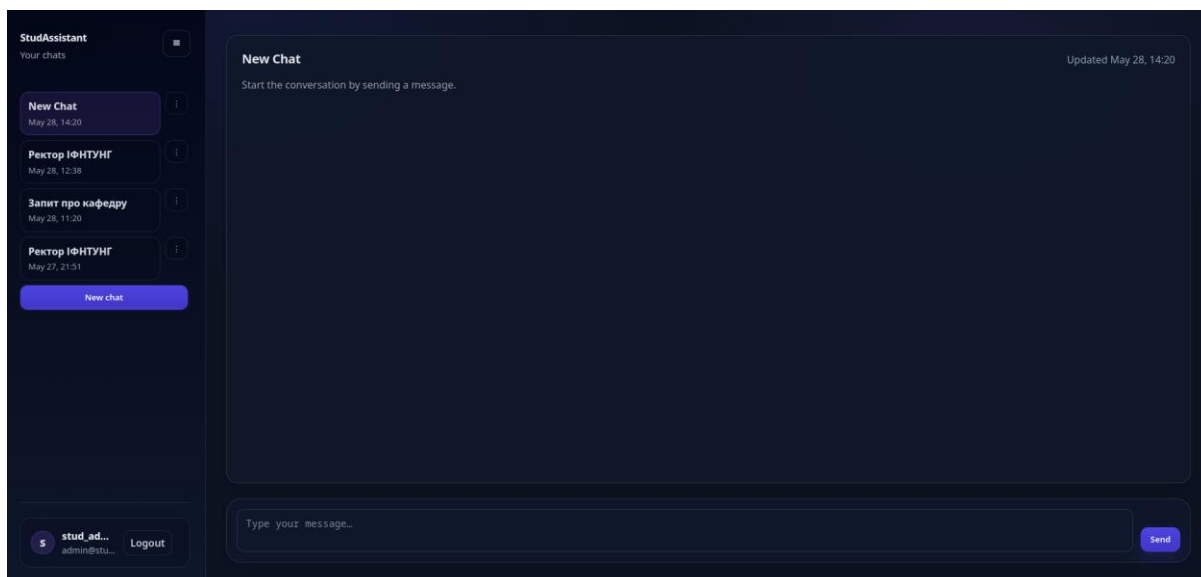


Рисунок А.6 - Головна сторінка

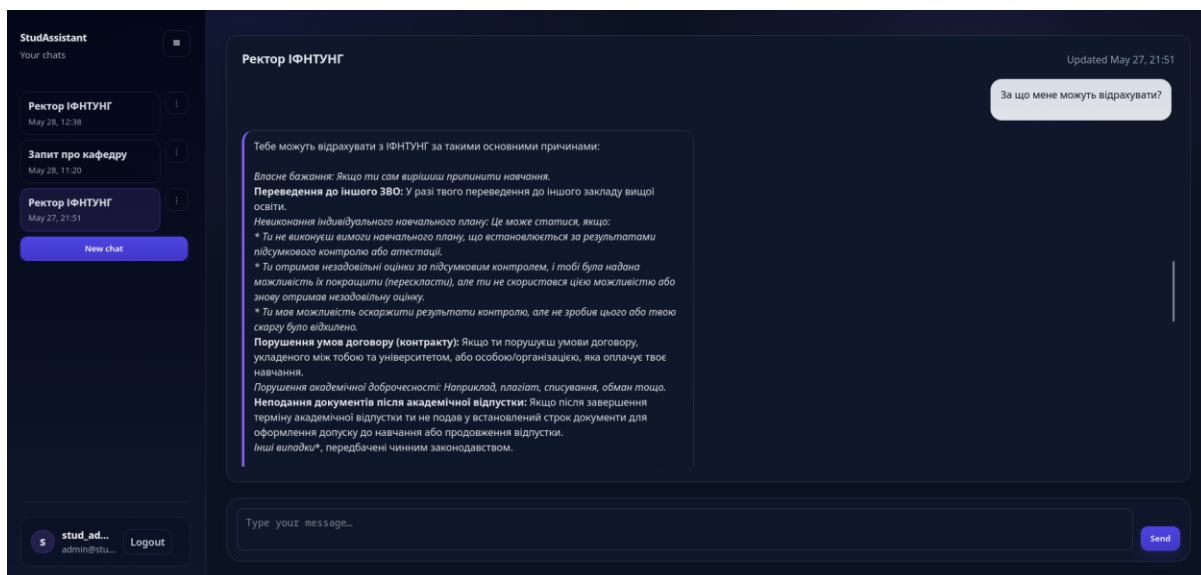


Рисунок А.7 - Сторінка чату

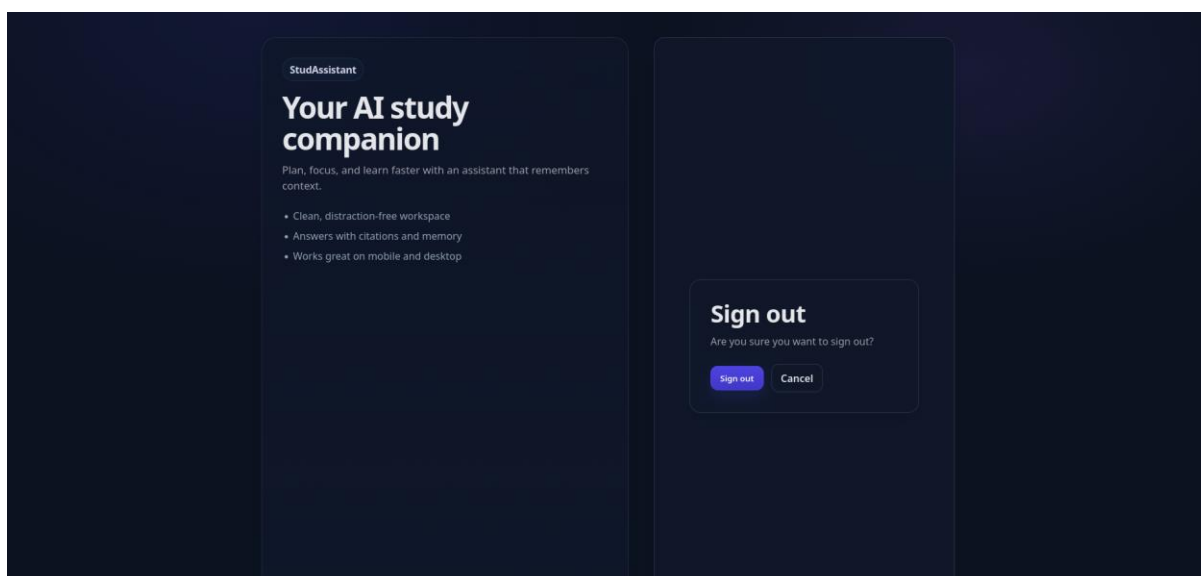


Рисунок А.8 - Сторінка підтвердження виходу з аккаунта

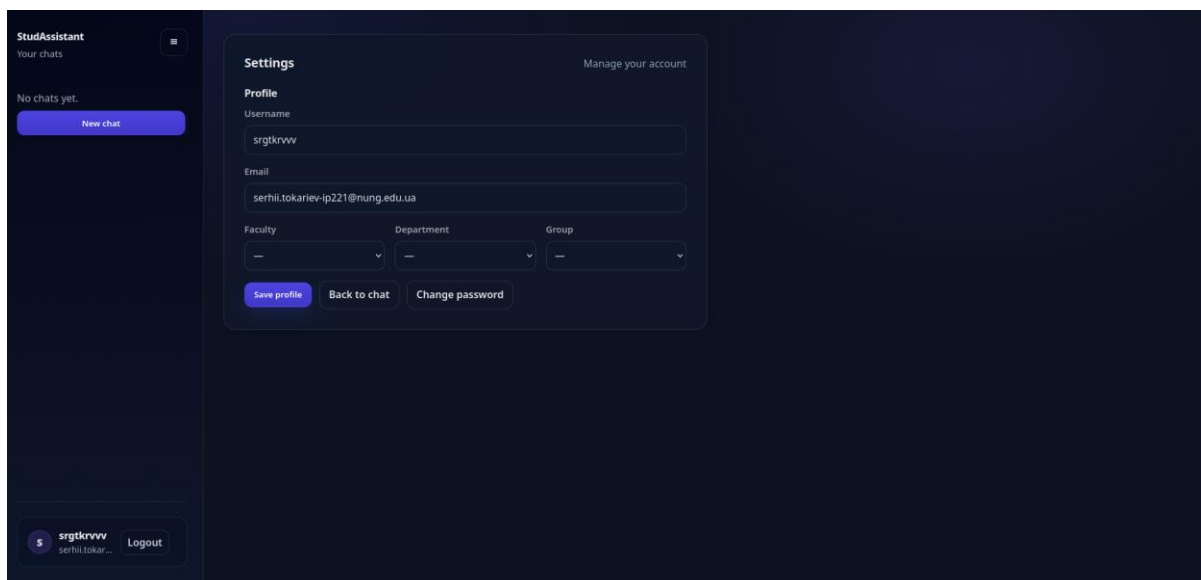


Рисунок А.9 - Сторінка налаштування профілю

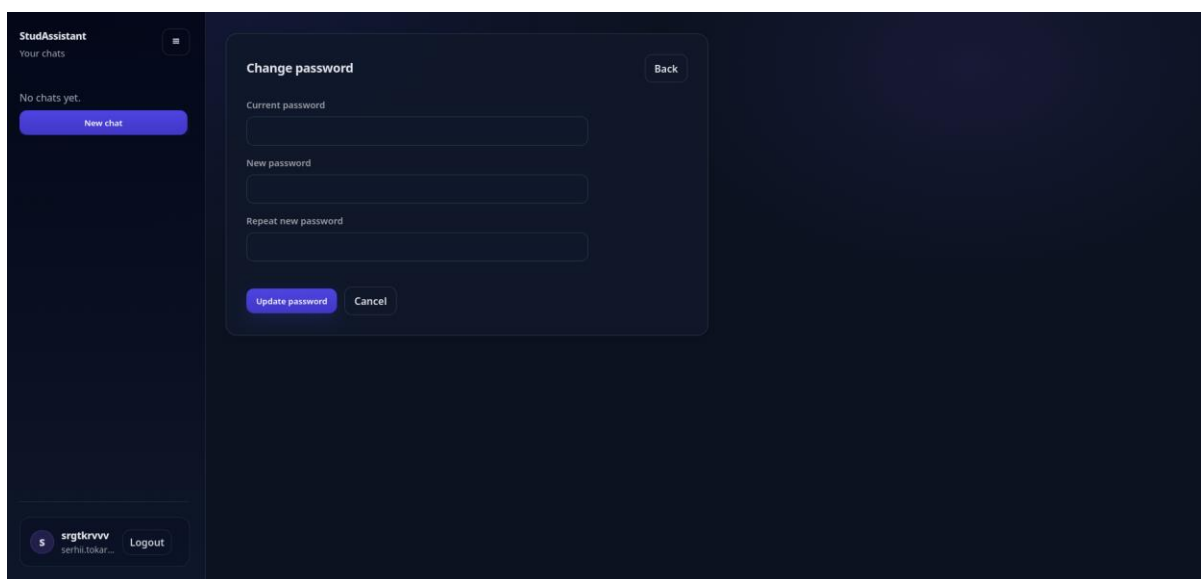


Рисунок А.10 - Сторінка зміни паролю

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “RAG-система для університету”.

Обсяг пояснювальної записки: 73 аркуша .

Дата закінчення роботи 06 червня 2026р.

ідпис: _____