Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка Telegram-бота для профорієнтації на базі генеративних мовних моделей
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-22-2
Єлизавета КОЛІСНИК
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)

Керівник ас. Микола ЧЕРНЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Колісник Єлизаветі Валеріївні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка Telegram-бота для профорієнтації на базі генеративних мовних моделей

затверджена наказом університету від 5 червня 20 26 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 7 липня 20 26 р.

3. Вихідні дані до роботи мова програмування Python, асинхронний фреймоврк aiogram для розробки Telegram-ботів, система управління базами даних SQLite, доступ до хмарного векторного сховища Pinecone, API великої мовної моделі Google Gemini (gemini-2.5-flash та gemini-embedding-2), відкриті дані реєстру Єдиної державної електронної бази з питань освіти (ЄДБО), статичні дані кадрового порталу Work.ua

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі

2) Проектування архітектури та технологічне обґрунтування системи

3) Програмна реалізація та інтеграція інтелектуальних модулів системи

4) Тестування та демонстрація роботи профорієнтаційної системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі та аналіз існуючих	09.06.2025–12.06.2026	виконано
	рішень		
3	Обґрунтування технологій та проектування	13.06.2026–16.06.2026	виконано
	архітектури		
4	Розробка алгоритмів автоматизованого збору	17.06.2026–21.06.2026	виконано
	та структурування даних		
5	Програмна реалізація клієнт-серверної частини	22.06.2026–26.06.2026	виконано
	та інтеграція ШІ-моделі		
6	Комплексне тестування розробленого	27.06.2026–29.06.2026	виконано
	програмного забезпечення		
7	Оформлення пояснювальної записки	30.06.2026–03.07.2026	виконано
8	Підготовка до захисту	04.07.2026–06.06.2026	виконано
9	Захист перед ЕК	07.07.2026	виконано

Дата видачі завдання 8 червня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Микола ЧЕРНЕНКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 100 с., 24 рис., 7 табл., 1 дод., 37 джерел.

АНАЛІЗ РИНКУ ПРАЦІ, КАР'ЄРНЕ КОНСУЛЬТУВАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, ОСВІТНІ ПРОГРАМИ, ПРОФОРІЄНТАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, GEMINI, PINECONE, RAG, RETRIEVAL-AUGMENTED GENERATION, TELEGRAM-БОТ.

Об'єкт дослідження – процес кар'єрної орієнтації та професійного самовизначення майбутніх студентів у контексті швидкого розвитку IT та ринку праці.

Предмет дослідження – алгоритми штучного інтелекту, методи обробки природної мови та технології RAG (Retrieval-Augmented Generation) для побудови інтелектуальних діалогових систем.

Мета роботи – розробка та впровадження інтелектуальної системи кар'єрної орієнтації для адаптивного вибору професій, програм вищої освіти та аналізу ринку праці.

Методи дослідження – системний аналіз, проектування архітектури програмного забезпечення, методи веб-скрейпінгу, семантичний аналіз тексту та тестування програмного забезпечення.

Кваліфікаційну роботу присвячено створенню інтелектуальної системи профорієнтації. Практичний результат – Telegram-бот для повного циклу кар'єрної консультації. Наукова новизна полягає у застосуванні RAG-архітектури (інтеграція Gemini API, векторної БД Pinecone, реєстру ЄДБО та порталу Work.ua), що забезпечує високу фактологічну точність і базує рекомендації на верифікованих даних у реальному часі. Розроблене рішення є функціональним і готовим до розгортання в закладах освіти. Практична значимість полягає у забезпеченні абітурієнтів надійною кар'єрною аналітикою для прийняття обґрунтованих рішень.

ABSTRACT

Bachelor's thesis contains: 100 pp., 24 fig., 7 tabl., 1 ann., 37 references.

ARTIFICIAL INTELLIGENCE, CAREER COUNSELING, CAREER GUIDANCE, EDUCATIONAL PROGRAMS, GEMINI, LABOR MARKET ANALYSIS, NATURAL LANGUAGE PROCESSING, PINECONE, RAG, RETRIEVAL-AUGMENTED GENERATION, TELEGRAM BOT.

Object of the research – the career guidance and professional self-determination process of prospective students in the context of rapid IT and labor market development.

Subject of the research – artificial intelligence algorithms, natural language processing methods, and Retrieval-Augmented Generation (RAG) technologies for building intelligent dialog systems.

Aim of the thesis – to design and implement an intelligent career guidance system for the adaptive selection of professions, higher education programs, and labor market analysis.

Research methods – system analysis, software architecture design, web scraping techniques, semantic text analysis, and software testing.

The thesis is devoted to the creation of an intelligent career guidance system. The practical result is a Telegram bot for a full cycle of career consultation. The scientific novelty lies in the application of a RAG architecture (integration of the Gemini API, Pinecone vector DB, EDEBO registry, and Work.ua portal), which ensures high factual accuracy and grounds recommendations in verified real-time data. The developed solution is functional and ready for deployment in educational institutions. The practical significance lies in providing applicants with reliable career analytics for informed decision-making.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі 11
1.1 Аналіз профорієнтації абітурієнтів у сучасних умовах 11
1.2 Огляд існуючих рішень 13
1.2.1 Портал «вступ.освіта.ua» 13
1.2.2 Платформа «epam campus» 15
1.2.3 Огляд платформи «дія.освіта» 16
1.2.4 Аналіз зарубіжного досвіду: платформа «careerexplorer» 18
1.2.5 Огляд офіційного ресурсу «my next move» 21
1.2.6 Комплексний аналіз та виявлення недоліків існуючих рішень 24
1.3 Аналіз сучасних технологій та методів побудови інтелектуальних систем оброки природної мови 26
1.3.1 Великі мовні моделі (llm) 26
1.3.2 Обмеження генеративних моделей та методи їх подолання 27
1.3.3 Технологія retrieval-augmented generation (rag) 28
1.3.4 Методи організації та збереження даних у ші-системах 28
1.4 Постановка задачі 29
2 Проєктування архітектури та технологічне обґрунтування системи 31
2.1 Деталізація функціональних та технічних вимог до системи 31
2.2 Аналіз методів та архітектурних підходів до реалізації компонентів системи 33
2.2.1 Аналіз платформ для реалізації користувацького інтерфейсу .. 34
2.2.2 Аналіз великих мовних моделей як когнітивного ядра системи 37
2.2.3 Аналіз методів та інструментів збереження даних 39
2.2.4 Аналіз мов програмування та інструментарію реалізації 42
2.3 Обґрунтування вибору технологічного набору 44

3 Програмна реалізація та інтеграція інтелектуальних модулів системи ... 48
3.1 Структура програмного проекту та обґрунтування технологічного стеку ... 48
3.2 Алгоритм автоматизованого збору даних та парсингу зовнішніх платформ ... 52
3.2.1 Автоматизований збір освітніх програм із використанням технології імітації браузера ... 52
3.2.2 Парсинг та структурування інформації про заробітні плати ... 57
3.3 Інтеграція підсистеми штучного інтелекту та реалізація rag-архітектури ... 61
3.3.1 Генерація векторних подань та ініціалізація індексів pinecone 61
3.3.2 Семантичний пошук а генерація персоналізованих рекомендацій ... 64
3.4 Програмна реалізація діалогового інтерфейсу telegram-бота та підсистеми управління сесіями ... 72
3.4.1 Організація локального збереження даних та дедуплікація користувацьких сесій ... 72
3.4.2 Керування станами користувача за допомогою finite stare machine (fsm) та резервне збереження даних ... 76
3.4.3 Розробка динамічних елементів інтерфейсу та механізму каруселної навігації ... 80
4 Тестування та демонстрація роботи профорієнтаційної системи ... 85
4.1 Демонстрація базового сценарію використання та інтелектуальної генерації рекомендацій ... 85
4.2 Перевірка відмовостійкості системи та обробка виняткових ситуацій ... 90
Висновки ... 94
Перелік джерел посилання ... 96
Додаток А Відомість кваліфікаційної роботи ... 100

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЄДБО – Єдина державна електронна база з питань освіти;

ЗВО – заклад вищої освіти;

СУБД – система управління базами даних;

ШІ – штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

BPMN – Business Process Model and Notation – модель і нотація бізнес-процесів;

CSV – Comma-Separated Values – формат файлу з розділовими комами;

DOM – Document Object Model – об’єктна модель документа;

FSM – Finite State Machine – скінченний автомат (машина станів);

HTML – Hypertext Markup Language – мова гіпертекстової розмітки;

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту;

IT – Information Technology – інформаційні технологій;

JSON – JavaScript Object Notation – текстовий формат обміну мови;

LLM – Large Language Model – велика мовна модель;

NLP – Natural Language Processing – обробка природної мови;

RAG – Retrieval-Augmented Generation – генерація, доповнена пошуком;

SDK – Software Development Kit – набір інструментів для розробки програмного забезпечення;

SQL – Structured Query Language – мова структурованих запитів;

UI – User Interface – користувацький інтерфейс;

UX – User Experience – користувацький досвід.

ВСТУП

Процес професійного самовизначення є одним із найбільш критичним етапів у житті старшокласників та абітурієнтів, оскільки свідомий вибір майбутнього фаху безпосередньо визначає їхню подальшу освітню траєкторію, соціальну адаптацію та успішність кар'єрної самореалізації. Проте значна частина молоді стискається із серйозними труднощами під час прийняття цього рішення. За даними сучасних соціологічних досліджень, зокрема комплексного аналізу Фундації Олени Зеленської «Індекс майбутнього», лише близько 30% підлітків від 13 до 16 років здатні впевнено обрати вектор майбутньої професійної діяльності. Як наслідок, цей перехідний етап супроводжується значним психологічним навантаженням і вимагає створення якісно нових інструментів індивідуальної інформаційно-консультаційної підтримки.

Оцінка сучасного стану проблеми показує, що вона суттєво ускладняється стрімким розвитком інформаційних технологій та автоматизацією виробничих процесів. У результаті таких змін ринок праці стає надзвичайно динамічним: з'являється нові міждисциплінарні спеціальності, трансформуються вимоги до кваліфікацій, а частина традиційних професій поступово втрачає свою актуальність. На цьому тлі класичні методи профорієнтації, що практикуються в закладах освіти, а також більшість комерційних онлайн-платформ не забезпечують гнучкого та персоналізованого підходу. Вони переважно базуються на статичних тестах та лінійних опитувальниках, які не враховують унікальну динаміку інтересів кожного окремого користувача.

Актуальність кваліфікаційної роботи полягає у необхідності переходу від застарілих статичних опитувальників до інтелектуальних діалогів систем. Застосування сучасних генеративних моделей штучного інтелекту у поєднанні з технологією Retrieval-Augmented Generation (RAG) дозволяє створити систему, здатну взаємодіяти з користувачем у формі природної

розмови, аналізувати приховані закономірності у відповідях та оперувати актуальними даними з державних реєстрів і кадрових порталів без ризику генерації хибної інформації («галюцинацій»).

Метою кваліфікаційної роботи є розробка, архітектурне проектування та програмна реалізація інтелектуальної систем профорієнтаційного консультування (у вигляді Telegram-бота) для автоматизованого підбору професій, релевантних освітніх програм закладів вищої освіти та надання аналітики ринку праці.

Отримані в ході виконання роботи результати мають високу практичні цінність, оскільки створений програмний комплекс є повністю функціональним та відмовостійким рішенням. Сфера застосування розробленої систем охоплюють: заклади загальної середньої освіти (як допоміжний інструмент для старшокласників), приймальні комісії та відділи профорієнтації університетів (для автоматизації консультацій та залучення нових вступників), а також інтеграцію в існуючі освітні онлайн-платформ для розширення їхнього функціоналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз профорієнтації абітурієнтів у сучасних умовах

В житті кожної людини настає дуже важливий етап – вибір професії. Цей етап припадає на учнів старших класів. Обрання професійного напрямку впливає на подальше формування особистості та її психічний стан. Зокрема, постають численні питання, як-от: чи зможе підліток матеріально забезпечувати себе в майбутньому, чи отримуватиме він задоволення від професійної діяльності, чи може зіткнутися з професійною незадоволеністю через помилковий вибір.

Проблема профорієнтації у молоді є актуальною темою. За результатами дослідження Фундації Олени Зеленської «Індекс майбутнього» (2025), лише 30% підлітків віком від 13 до 16 років можуть впевнено визначити майбутню професійну діяльність або фах [1]. Також із дослідження спостерігається, що 50% лише замислювались про майбутню професію, а решта 20% – зовсім не замислюються про вибір професійної діяльності.

Традиційно процес професійного самовизначення вважається складним етапом життєвого шляху особистості, що пояснюється як недостатньою психологічною зрілістю, так і обмеженим життєвим досвідом учнів старших класів (8–9 класів). Проте в сучасних умова ця проблема набуває це більшої актуальності у зв’язку з глобальними трансформаціями ринку праці, спричиненні активним прогресом та інтеграцією технологій штучного інтелекту.

Якщо раніше абітурієнти мали змогу спиратися на відносно стабільні шляхи розвитку кар’єри, то сьогодні спостерігається швидка трансформація традиційних професій та поява нових спеціальностей, пов’язаних із цифровими технологіями. Масова автоматизація рутинних операцій, прогрес інтелектуальних рішень та трансформація вимог роботодавців

накладають додаткові перешкоди на молодь під час визначення подальшого професійного вектору.

У зв’язку з цим виникають виклики:

– швидке зникнення окремих спеціальностей та поява нових професій у сфері ІТ-технологій, ускладнюють тривале кар’єрне планування на основі традиційних методів профорієнтації;

– поширення технологій штучного інтелекту викликає занепокоєння серед молоді щодо потенційного витіснення окремих видів розумової діяльності автоматизованими системами;

– недостатня наявність або брак додаткових заходів з боку освітніх закладів для поглибленого вивчення теми професійного розвитку;

– складність інтерфейсів вебресурсів закладів вищої освіти та перевантаженість інформацією;

– вплив соціального оточення, зокрема батьків, чиї поради щодо вибору «затребуваної» професії часто не беруть до уваги інтереси самої дитини;

– низька ефективність статичних тестів із великою кількістю запитань для профорієнтації, які вказують лише загальний вектор діяльності, не даючи конкретики.

Варто зазначити, що всі ці чинники спричиняють виникнення стресового стану. Тиск оточення може спричинити формування хибної професійної траєкторії. Окрім того, університети надають інформації про факультети та спеціальності, використовуючи складну термінологію, яка є незрозумілою для молоді. На сьогодні методи тестування є застарілими: вони є одноманітними та знижують мотивацію для подальшого самовизначення. Як наслідок, традиційні методи втрачають здатність мотивувати молодь до самовизначення. Дані дослідженнь свідчать, що лише 1% підлітків звертаються до шкільних психологів, тоді як 42% шукають відповіді в інтернеті [1].

Беручи до уваги запит аудиторії на цифрові інструменти, виникає потреба в інтелектуальному програмному засобі, здатному до динамічної комунікації. Таке рішення повинно не лише пропонувати окремі назви професій, а й допомагати координувати процес вибору, враховуючи індивідуальні інтереси користувача та розвиток ринку, що трансформується під впливом ШІ (штучний інтелект).

1.2 Огляд існуючих рішень

На сьогодні існують чимало онлайн-ресурсів, які призначенні для допомоги молоді у виборі майбутній професії. Проведений аналіз дозволяє класифікувати ці рішення на три основні групи: системи збору інформації, платформи приватних компаній та державні сервіси профорієнтаційного спрямування.

1.2.1 Портал «Вступ.Освіта.UA»

Портал «Вступ.Освіта.UA» є одним із найпоширеніших інформаційних джерел в Україні, що надає деталізовану інформацію щодо прохідних балів, вартість навчання за різними освітніми програмами, переліку спеціальностей та умови вступу до закладів вищої освіти (ЗВО) [2].

Основними функціями порталу є:

– надання актуальної статистики вступів минулих років;
– пошук ЗВО за регіонами та галузями знань;
– розрахунок конкурсного балу абітурієнта.

Основним призначенням порталу є порівняння різних закладів вищої освіти та аналіз даних щодо вступу. Проте цей ресурс має значний недолік: він орієнтований на молодь, яка вже визначилась з професійним напрямом. Як демонструє інтерфейс на рисунку 1.1, пошук здійснюється за допомогою жорстких фільтрів, а сторінка з переліком спеціальностей перевантажений

значною кількістю табличних даних (рисунок 1.2), що ускладнює сприйняття інформації підлітками. Крім того, відсутність інтелектуального помічника не дозволяє отримати персоналізовану пораду щодо вибору професійного напрямку.

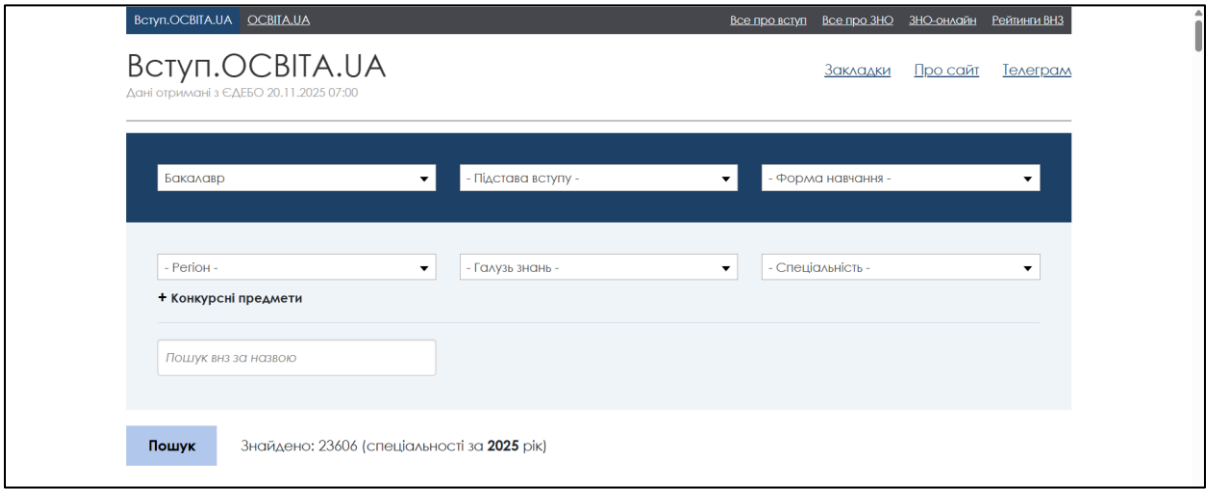


Рисунок 1.1 – Інтерфейс вибору спеціальності на порталі «Вступ.ОСВІТА.UA»

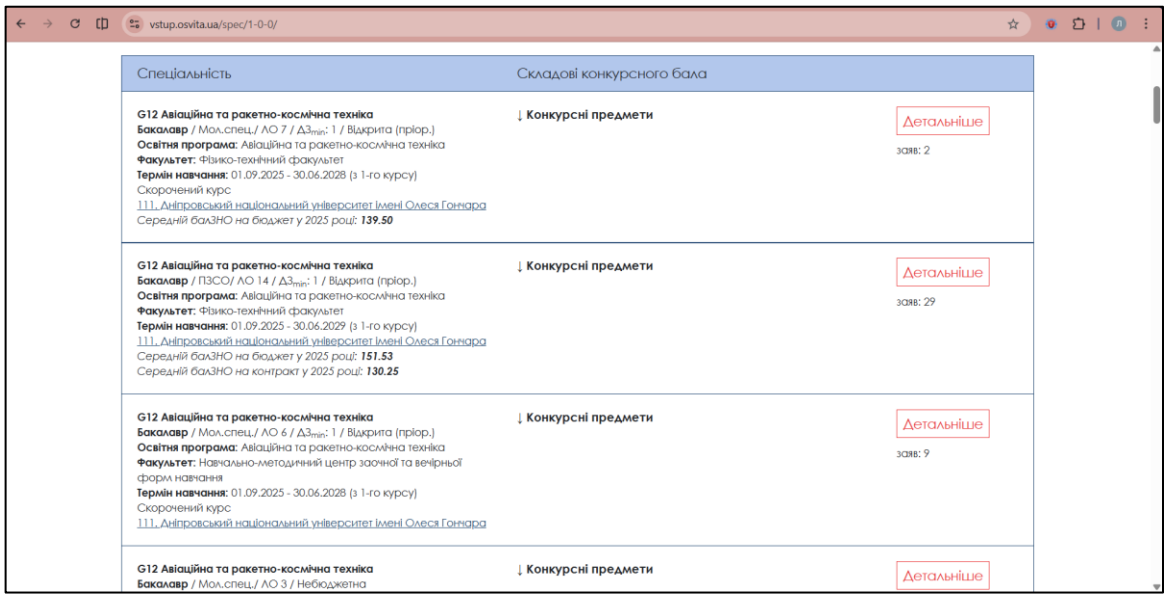


Рисунок 1.2 – Інтерфейс перелік спеціальностей на порталі «Вступ.ОСВІТА.UA»

1.2.2 Платформа «EPAM Campus»

Платформа «EPAM Campus» є прикладом корпоративного освітнього ресурсу створеного IT-компанією задля поширення інформації про професії в галузі інформаційних технологій та заохочення потенційних учнів до навчальних курсів [3]. Подібні платформи нерідко пропонують пройти профорієнтаційне тестування, яке допомагає визначити схильності особистості до певних напрямів діяльності.

Зазвичай подібні освітні ресурси використовують статичні тести, висновки яких можуть бути надаватися у вигляді загальних рекомендацій або повідомлятися користувачу пізніше через представників компанії. Відміною рисою платформи «EPAM Campus» виступає те, що після завершення тесту особа миттєво отримує результат у вигляді конкретної професії, а також детальний опис професійних обов'язків і необхідного набору знань, куди входять як технічні вміння та так звані «soft skills» (рисунок 1.3).

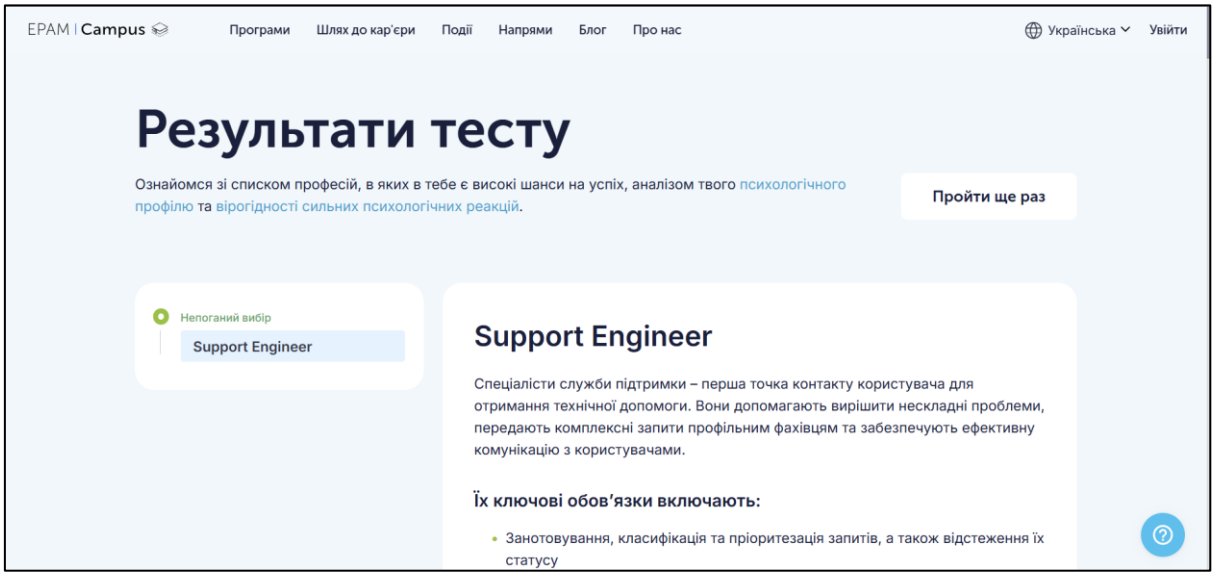


Рисунок 1.3 – Інтерфейс компанії «EPAM Campus», після проходження тесту

Такий підхід може допомогти особам глибше зрозуміти вимоги до певної професії. Платформа спрямовує користувача на певний кар’єрний шлях, що сприяє формування уявлення про майбутню кар’єру.

Однак дане рішення має низьку обмежень. По-перше, платформа фокусується переважно на IT-напрямку, що значно звужує діапазон можливих професій. По-друге, результати тестування мають статичний характер і не передбачають подальшого уточнювального діалогу з користувачем. Крім того, у результатах відсутній зв’язок між запропонованою професією та офіційними державними кодами спеціальностей, що використовуються у системі вищої освіти.

1.2.3 Огляд платформи «Дія.Освіта»

Державний освітній ресурс «Дія.Освіта», спрямований на розвиток цифрових навичок та знайомство молоді з різноманітністю професій [4]. Цей сервіс надає великий набір інструментів для профорієнтації, включаючи освітні серіали, вебінари, подкасти, гайди, симулятори та тестування (рисунок 1.4).

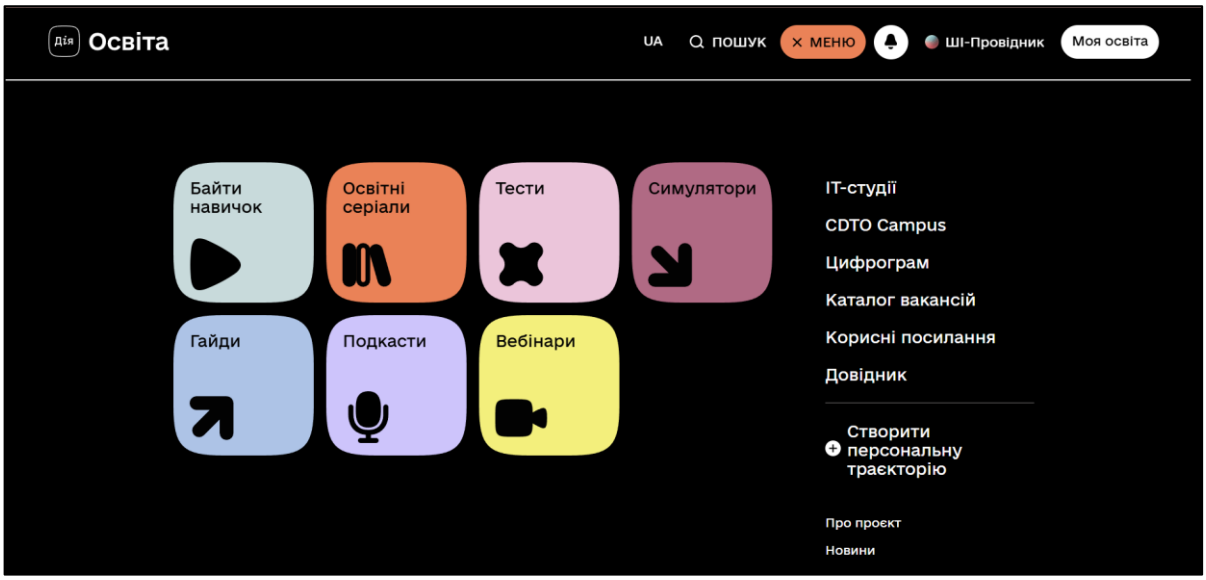


Рисунок 1.4 – Інтерфейс платформи «Дія.Освіта»

Подібні матеріали дозволяють особам, які користуються платформою, розібратися з особливостями різних професій, зрозуміти ключові обов'язки та вимоги до фахівців у певній галузі. Завдяки використанню інтерактивних методів, цей портал сприяє формуванню загального розуміння про поточний стан ринку праці.

Для більш детального аналізу професійних нахилів, цей ресурс також пропонує пройти тест з профорієнтації. Після завершення тестування користувач отримує декілька варіантів професій, а також перелік актуальних вакансій, доступних на ринку (рисунок 1.5). Такий підхід дає можливість користувачам отримати певні орієнтири щодо можливих напрямів професійного розвитку.

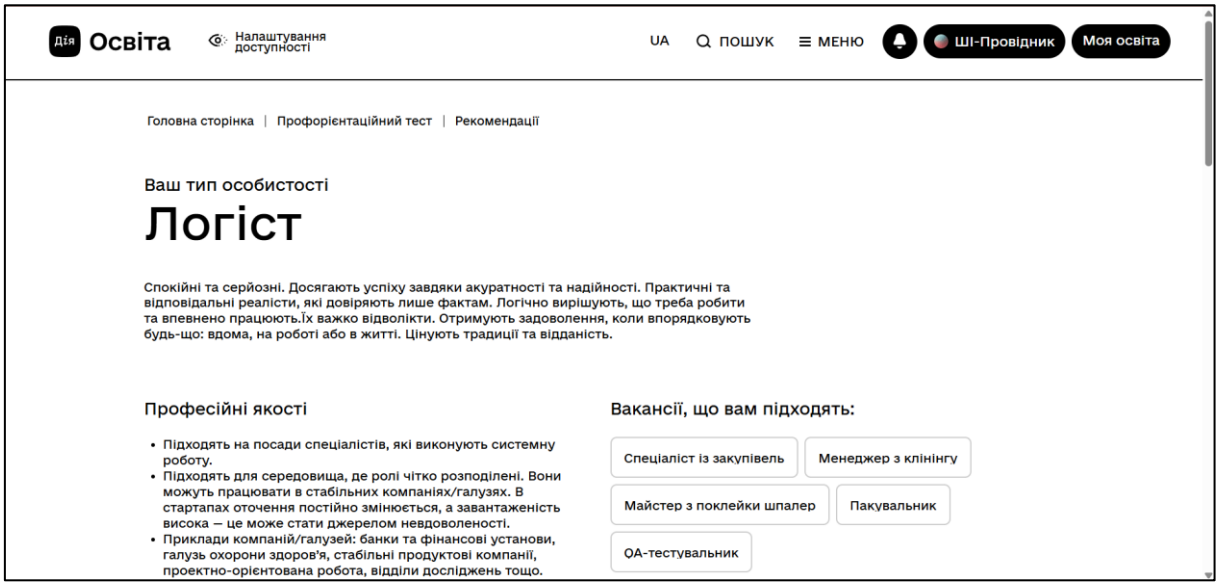


Рисунок 1.5 – Інтерфейс платформи «Дія.Освіта», після проходження тесту

Проте, такий метод має й певні обмеження. Профорієнтаційний тест містить значну кількість незмінних запитань (близько 94), що призвести до втрати зосередженості під час проходження. Крім того, запитання тесту не адаптуються до попередніх відповідей користувача, тобто система не застосовує індивідуалізованого підходу до формування кінцевих результатів.

Іншим недоліком є те, що після проходження тесту користувач отримує лише перелік рекомендованих професій та пропозиції роботи, але система не надає інформації про відповідні коди спеціальностей чи освітні програми у вищих навчальних закладах. У результаті абітурієнту доводиться самостійно шукати інформацію про те, яку саме спеціальність слід обрати для подальшого навчання за рекомендованим професійним напрямом.

Таким чином, попри наявність значного обсягу навчальних матеріалів та інструментів профорієнтації, цей портал не забезпечує повного супроводу користувача, починаючи від моменту виявлення професійних інтересів до вибору конкретної спеціальності для вступу.

1.2.4 Аналіз зарубіжного досвіду: платформа «CareerExplorer»

Для отримання цілісного уявлення про світові стандарти автоматизованої профорієнтації було проаналізовано американську платформу «CareerExplorer», розроблену компанією Sokanu. На відміну від більшості статичних інформаційних ресурсів, цей сервіс функціонує як розгалужена аналітична мережа, яка базується на обробці понад 140 мільйонів первинних даних з метою формування індивідуальних зразків професійної відповідності користувачів [5].

Хід взаємодії з цією системою структуровано як проходження низки послідовних етапів: від початкового формування психологічного образу користувача аж до детального оцінювання його кар’єрних нахилів та шляхів навчання. Загальний час, необхідний для повного тестування, складає приблизно тридцять хвилин, що дозволяє системі зібрати достатній обсяг даних для здійснення глибокого аналізу (рисунок 1.6).

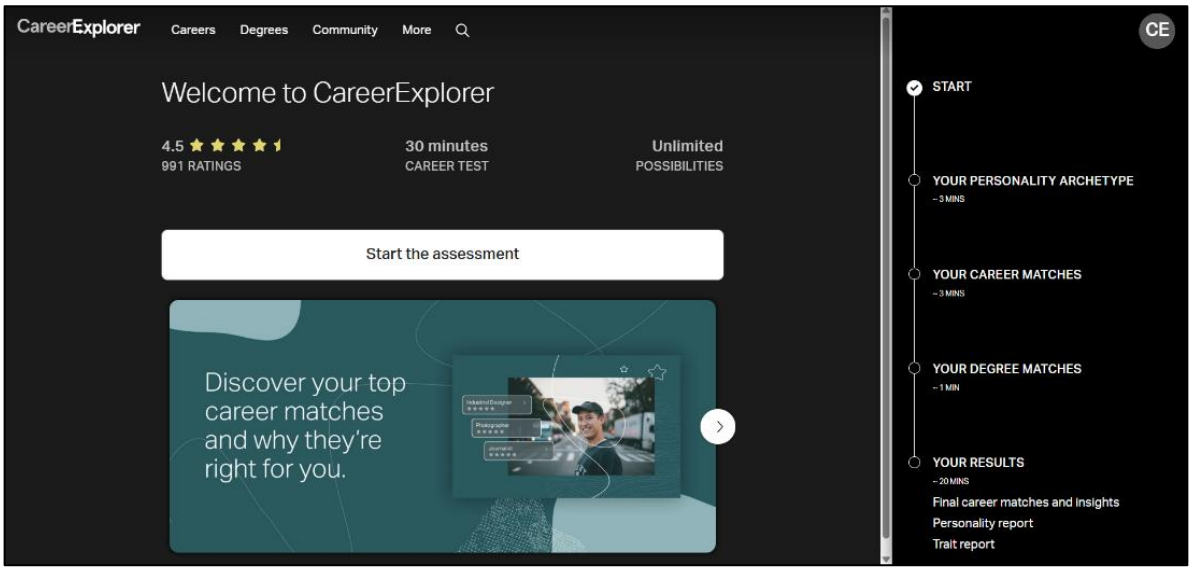


Рисунок 1.6 – Етапи та часова тривалість проходження тестування на платформі «CareerExplorer»

Під час практичного дослідження можливостей сервісу було встановлено, що «CareerExplorer» використовує складні психометричні моделі. Система не обмежується лише видачею переліку професій, а також визначає індивідуальний архетип особистості. Як приклад, після проходження тестування може бути виявлено архетип «Maverick» – інтуїтивний, впливовий та експресивний тип особистості, який притаманний приблизно 4,7% тих, хто скористався платформою. Також система оцінює рівень схильності до різних видів діяльності, зокрема до заснування власної справи (рисунок 1.7).

Огляд цього рішення свідчить, що реалізоване розробниками глибинне психологічне профілювання дозволяє оцінювати не лише професійні навички, а й психологічну відповідність користувача певному робочому середовищу. Такий підхід допомагає іноземному сервісу спрямовувати абітурієнтів до більш усвідомленого вибору як самої спеціальності, так і формату майбутньої зайнятості.

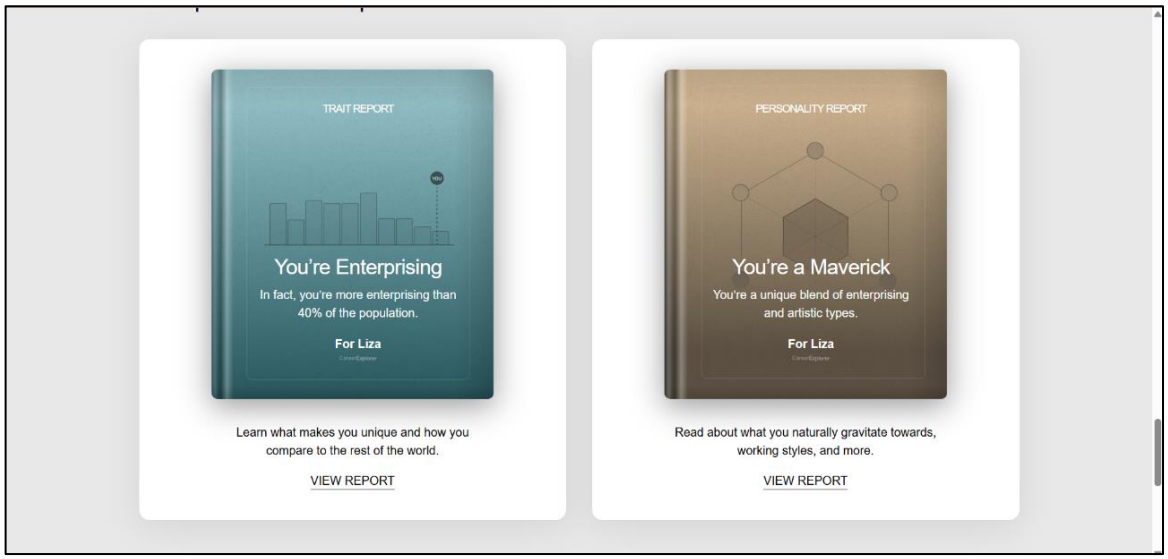


Рисунок 1.7 – Архетип особистості «Maverick» та рівень підприємливості

Кінцевий результат роботи платформи вражає своєю деталізацією. Користувач отримує інформацію про середній рівень заробітної плати, динаміки розвитку галузі та рівень задоволеності фахівців у відповідних сферах діяльності. Система також пропонує перелік освітніх ступенів (Degrees), які можуть забезпечити підготовку для обраної професії, а також надає інформацію про потенційні прибутки та витрати, пов'язані із здобуттям освіти (рисунок 1.8).

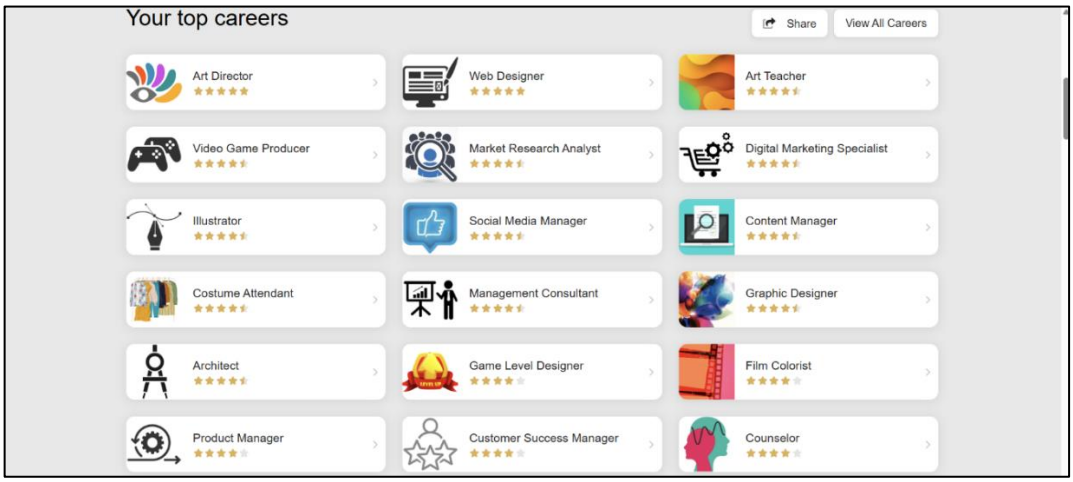


Рисунок 1.8 – Рекомендовані професії та освітні ступені

Незважаючи на високий технологічний рівень платформи, досвід її практичного застосування виявив і деякі недоліки. Основною перешкодою є значне розумове напруження для користувача, адже у процесі тестування вимагається оцінювати велику кількість подібних за характером тверджень та професій, використовуючи п’ятибальну шкалу. Така одноманітність може викликати швидке виснаження та втрату зацікавлення в подальшому тестуванні. Крім того, система залишається односторонньою: вона не передбачає можливість поставити уточнююче запитання або отримати додаткове пояснення результатів, що є важливим для абітурієнтів у стані професійної невизначеності (рисунок 1.9).

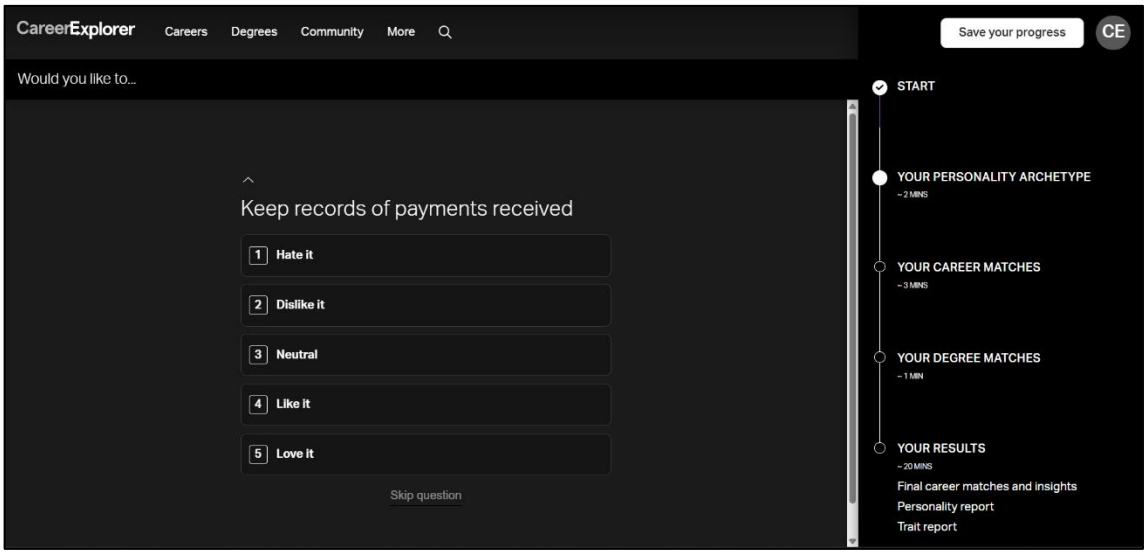


Рисунок 1.9 – Приклад монотонного опитування на основі статичних оцінок

1.2.5 Огляд офіційного ресурсу «My Next Move»

Ще одним прикладом для профорієнтації є платформа «My Next Move», розроблена зусиллями Національного центру «O*NET» за підтримки Міністерства праці Сполучених Штатів Америки. Ця система являє собою значну базу даних про професій [6]. Ключовим елементом

системи є інструмент «O*NET Interest Profiler», який фокусується на швидкій діагностиці професійних інтересів. На відміну від «CareerExplorer», тест складається з 30 запитань, що значно зменшує час проходження та знижує психологічне навантаження на користувача (рисунок 1.10).

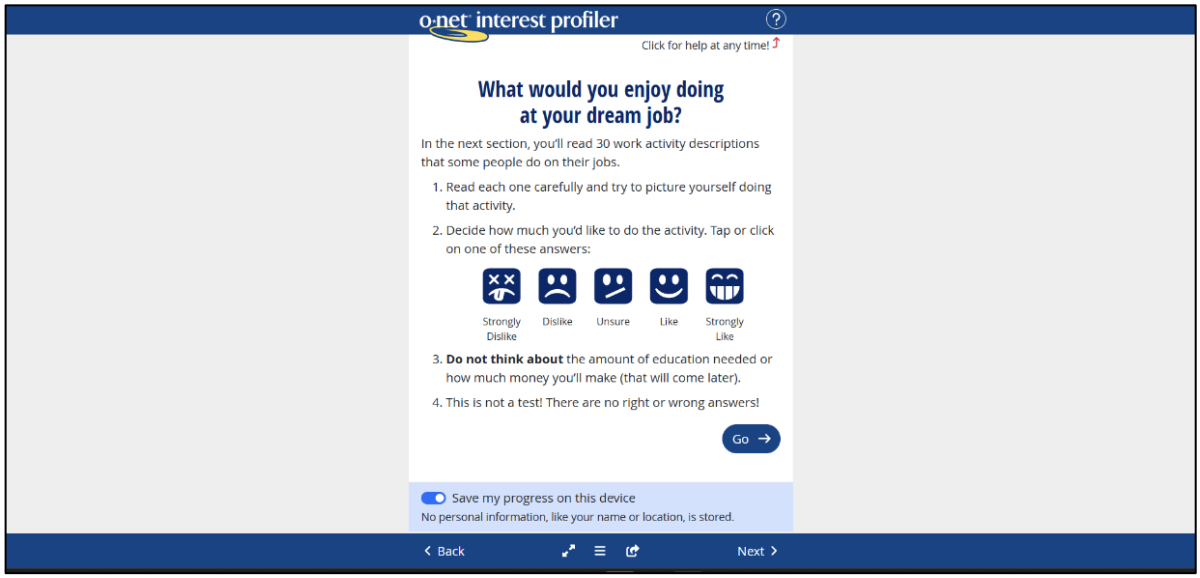


Рисунок 1.10 – Інтерфейс тестування «O*NET Interest Profiler» на 30 запитань

Алгоритм тесту базується на класичній моделі типів особистості RIASEC, що включає шість категорій інтересів: реалістичний, дослідницький, артистичний, соціальний, підприємницький та конвенційний. Після завершення опитування система генерує графічний профіль інтересів користувача, що дає можливість візуально оцінити його схильність та визначити найбільш відповідні сфери діяльності, що видно на рисунок 1.11.

Детальне вивчення цього інструменту свідчить, що розробники ресурсу використовують таку візуалізацію для подолання проблеми складності психологічних профілів. Оскільки в реальності найчастіше фіксуються змішані типи інтересів за моделлю RIASEC, іноземна платформа свідомо виводить комбінований результат. Оцінка подібного

інтерфейсного рішення підтверджує: наочне структурування даних об'єктивно допомагає користувачеві в самостійній інтерпретації підсумків та слугує зручним механізмом для відсіювання нерелевантних кар'єрних напрямків.



Рисунок 1.11 – Графічна візуалізація профілю інтересів за моделлю RIASEC

На підставі отриманих результатів сервіс пропонує список рекомендованих професій. Для кожної з них надається інформація про рівень заробітної плати та перспективи розвитку галузі. Професії з високим попитом на ринку праці позначається спеціальним маркером Bright Outlook, що демонструє про сприятливі перспективи працевлаштування, як показано на рисунку 1.12.

Наявність такого прогностичного індикатора є критично важливою для формування реалістичних кар’єрних очікувань. Інтеграція актуальних аналітичних даних про стан ринку праці дозволяє користувачам не лише орієнтуватися на власні психологічні схильності, але й враховувати економічну доцільність вибору. Такий прагматичний підхід допомагає

мінізувати ризики майбутнього безробіття та гарантує, що обраний напрям підготовки залишиться затребуваним після завершення навчання.



Рисунок 1.12 – Результати підбору професій з урахуванням ринкових перспектив

Аналіз ресурсу підтверджує, що ефективна профорієнтація можлива за умови поєднання лаконічного опитування та потужної бази знань. Проте система залишається статичним веб-сайтом, який лише видає готові списки. Таким чином, у процесі професійного самовизначення виникає розрив між отриманими рекомендаціями та подальшими кроками абітурієнта.

1.2.6 Комплексний аналіз та виявлення недоліків існуючих рішень

Проведений комплексний аналіз існуючих в Україні та зарубіжних систем для профорієнтації дає змогу дійти висновку, що попри значне різноманіття інструментів, у цій сфері простежується значний технологічний розрив. Більшість розглянутих систем функціонують як статичні бази даних або лінійні опитувальники, що не здатні забезпечити індивідуальний та гнучкий підхід до кожного абітурієнта.

У результаті дослідження було виокремлено низку характерних недоліків профорієнтаційних веб-ресурсів.

Насамперед, це статичність та прямолінійність процесу взаємодії. Більшість доступних тестових систем використовують заздалегідь сформований набір запитань, який не змінюється залежно від відповіді користувача. Такий метод не враховує індивідуальні особливості абітурієнта та не дозволяє адаптувати процес тестування під його інтереси.

По-друге, надмірне розумове навантаження. У багатьох системах кількість запитань у тестах сягає від 30 до 90 і більше, що призводить до зниження концентрації уваги користувача та втрати мотивації завершувати опитування.

По-третє, надмірність інформації в інтерфейсах. Значна частина українських платформ орієнтована на користувачів, які вже приблизно усвідомлюють бажаний професійний напрямок. Як наслідок, вони пропонують складні таблиці або каталоги спеціальностей, які не допомагають у процесі первинного вибору.

По-четверте, відсутність інтерактивного супроводу користувача. Жодне з досліджених рішень не забезпечує безперервний сценарій взаємодії типу « визначення інтересів – формування рекомендацій щодо професій – підбір закладів вищої освіти» у форматі діалогової взаємодії. Користувач змушений самостійно аналізувати отримані результати та приймати остаточне рішення.

Сукупність цих недоліків вказує на наявність певної технологічної прогалини у сфері автоматизованої профорієнтації. Це зумовлює необхідність створення інтелектуального програмного засобу, здатного забезпечити гнучку та адаптивну взаємодію з користувачем.

Найбільш перспективним вектором для подолання цієї проблеми є створення системи, побудованої на базі технологій обробки природної мови та сучасних моделях штучного інтелекту. Такий підхід дозволить перейти від статичних тестів до інтерактивного діалогу з користувачем, у межах

якого система може аналізувати запити, уточнювати інформацію та формувати індивідуальні рекомендації.

Саме з цієї причини виникає необхідність у аналізу сучасних методів обробки природної мови (NLP) та архітектури рішень, на яких базуються сучасні мовні моделі.

1.3 Аналіз сучасних технологій та методів побудови інтелектуальних систем оброки природної мови

Сучасний розвиток систем штучного інтелекту у сфері обробки природної мови (NLP) базується на використанні архітектури Transformer [7]. Дана архітектура стала фундаментом для розробки сучасних інтелектуальних систем, здатних генерувати текст, аналізувати вміст запитів та працювати із змістовими зв’язками між словами.

Ключовим елементом функціонування Transformer є механізм самої уваги (self-attention), який дозволяє моделі аналізувати всі слова у реченні одночасно та визначати їх взаємозв’язки незалежно від порядку розташування. Завдяки цьому система здатна більш точно розуміти зміст тексту та контекст запиту користувача. Саме на цій архітектурі було збудовано великі мовні моделі.

1.3.1 Великі мовні моделі (LLM)

Великі мовні моделі (LLM) становлять основу сучасних систем у сфері опрацювання людської мови. Ці системи навчаються на великих масивах текстових даних і здатні вирішувати широкий спектор завдань: генерація тексту, аналіз запитів користувача, логічне узагальнення інформації та формування пропозицій.

На сучасному ринку виокремлюються дві домінуючи лінійки моделей, що спираються на архітектурне рішення Transformer.

Моделі сімейства GPT функціонують за механізмом послідовного формування тексту, де система поступово передбачає наступний слово, спираючись на вже наявний контекст. Дані моделі демонструють виняткову ефективність у задачах, що вимагають логічного висновку, генерації тексту та програмного коду. Однак їх застосування у комерційних проектах може бути обмежене закритою архітектурою та високою вартістю обробки запитів через API (Application Programming Interface) [8].

Іншою важливою лінійкою є моделі Gemini, розроблені компанією Google. Їхніми відмінними рисами є мультимодальний підхід, можливість опрацьовувати великі обсяги контексту та оптимізацією під різні обчислювальні ресурси. Моделі з підсерії «Flash» вирізняються низькою часовою затримкою відповіді (low latency), що робить їх ефективними для використання у чат-системах реального часу [9].

1.3.2 Обмеження генеративних моделей та методи їх подолання

Незважаючи на потужний потенціал генеративних мовних моделей, їх інтеграція у системи консультування стискається з рядом обмежень. Ключовим недоліком вважається феномен, відомий як «галюцинації», коли система видає інформацію, що є неправдивою або неіснуючою. Це пояснюється ймовірнісною природою роботи LLM, чий принцип полягає у передбаченні наступного слова виходячи із вивчених закономірностей у навчальних даних.

Іншим значним недоліком є обмеженість знань моделі. Кожна мовна модель має певний поріг навчання (knowledge cutoff), що означає повну відсутність інформації про події чи зміни, які відбулися після завершення її навчання [10].

Для подолання цих обмежень у сучасних системах штучного інтелекту застосовується технологія RAG [10].

1.3.3 Технологія Retrieval-Augmented Generation (RAG)

Технологія RAG є архітектурним підходом, який поєднує генеративні мовні моделі з зовнішніми базами знань. На відміну від традиційного підходу, де модель покладається виключно на внутрішні знання, у RAG-системах спершу відбувається пошук необхідної інформації у спеціальній базі знань.

Ключовим елементом цього процесу пошуку є використання методу щільного пошуку (Dense Retrieval). Головною метою полягає перетворення текстового вмісту на числові векторні образи, відомі як ембедінги. Пошук потрібної відповіді реалізується через ідентифікацію векторів, розташованих найближче один до одного у багатовимірному просторі, дозволяючи системі знаходити відповідні дані на основі розуміння сенсу запиту, навіть якщо не спостерігається збіг слів [7].

1.3.4 Методи організації та збереження даних у ШІ-системах

Для досягнення високої продуктивності інтелектуальної системи використовується поєднання декількох типів баз даних. Це дає змогу якісно обробляти як структуровану, так і неструктуровану інформацію.

Векторні бази даних, як-от ChromaBD та FAISS, призначенні для збереження неструктурованих текстових даних, наприклад, детальний опис професій, освітніх програм і університетів. Бібліотека FAISS вирізняється високою швидкістю індексації великих обсягів даних, забезпечуючи швидкий пошук за подібністю серед масивів векторних ознак [11]. Натомість ChromaBD пропонує розширений функціонал для роботи з метаданими та легко інтегрується з сучасними системами пошуку інформації, що критично важливо для реалізації архітектури RAG [12].

Для збереження структурованої інформації, а саме дані про користувачів та історію взаємодії із системою, тут доцільно

використовувати SQL-орієнтовані реляційні бази. Такий підхід дозволяє ефективно структурувати логіку діалогу, спираючись на логіку моделі скінченного автомата станів (FSM). Застосування FSM гарантує передбачуваність поведінки системи та чіткий контроль переходів між різними етапами взаємодії [13]. Як наслідок, цей інтелектуальний помічник може надійно фіксувати поточну стадію проходження тестування та коректно продовжувати діалог із користувачем, підвищуючи надійність і передбачуваність роботи системи.

1.4 Постановка задачі

На основі проведеного аналізу предметної галузі та вивчення недоліків існуючих цифрових сервісів стає очевидним, що сучасна профорієнтація потребує переходу від статичних методів до створення гнучких інтелектуальних рішень.

Метою даної роботи є розробка програмного засобу, що функціонуватиме як персоналізований цифровий консультант, здатний забезпечити комплексний супровід абітурієнта у процесі вибору професійного напрямку.

Наступним кроком є формування переліку ключових завдань, які має вирішувати майбутня система для забезпечення якісного рівня консультування.

Насамперед, потрібно налагодити персоналізоване середовище для взаємодії. Програма повинна вміти ідентифікувати користувача та підтримувати безперервність контексту спілкування. Це дозволить уникати повторенню запитань, формуючи відчуття єдиної, послідовної консультації, де кожна наступна відповідь базується на вже сформованому профілі інтересів особистості.

Наступним важливим аспектом є реалізація адаптивного діалогового опитування. Процес вивчення вподобань абітурієнта має відбуватися у

формі комбінованої взаємодії. Спочатку система повинна зібрати базовий масив даних про інтереси та академічні успіхи через структуровані запитання, а потім – динамічно заглибитися у найбільш перспективні напрями за допомогою уточнювальних запитань. Такий підхід дозволить отримати детальне розуміння здібностей особистості, уникаючи при цьому перевтоми користувача, яка була виявлена під час аналізу існуючих аналогів.

Фундаментальним завданням є впровадження методів інтелектуального пошуку інформації. Система повинна оперувати актуальною базою знань про освітні програми та спеціальності, проте її головною перевагою має стати здатність розуміти зміст і контекст запитів. Пошук відповідних рекомендацій має здійснюватися не за простим збігом ключових слів, а на основі глибокого семантичного пошуку у базі знань про освітні програми і спеціальності.

Кінцевим результатом роботи системи має стати генерація обґрунтованих персоналізованих рекомендацій. Цифровий консультант повинен не лише пропонувати перелік спеціальностей, а й пояснювати логіку цих рекомендацій, допомагаючи абітурієнту усвідомити зв’язок між його навичками та обраним фахом. При цьому всі результати взаємодії мають архівуватися, щоб користувач мав можливість повернутися до отриманих рекомендацій у будь-який зручний час.

Завершальним етапом розробки є створення зручного інтерфейсу, що відповідатиме звичкам і потребам сучасної молоді. Взаємодія має бути простою та динамічною, що сприятиме зменшенню розумового навантаження та перетворенню складного процесу вибору професії на зрозумілий і продуктивний шлях.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ТЕХНОЛОГІЧНЕ ОБҐРУНТУВАННЯ СИСТЕМИ

2.1 Деталізація функціональних та технічних вимог до системи

Відповідно до завдань, що були окреслені у розділі 1.4, є конкретизація функціональних та технічних вимог, які висуваються до майбутнього програмного забезпечення. На цій стадії загальне бачення інтелектуального помічника перетворюється на чітко визначений набір характеристик, які визначатимуть поведінку продукту та критерії, за якими оцінюватиметься його ефективність. Такий підхід є необхідним для переходу від теоретичного обґрунтування проблеми до безпосереднього інженерного проєктування архітектури системи.

Функціональні вимоги орієнтовані на створенні безперервного та адаптивного середовища для взаємодії з абітурієнтом. Для забезпечення керованості процесом консультування та актуалізації даних передбачено розподіл доступу між двома ролями: абітурієнт та адміністратор. Першочерговим завданням є забезпечення персоналізації через механізм ідентифікації. Це передбачає, що система має мати змогу автоматично розпізнавати користувача за його унікальним системним ідентифікатором. Це запобігає втраті контексту та дозволить зберігати історію попередніх консультаційних сесій, завдяки чому кожна нова взаємодія відбувається з урахуванням накопиченої інформації, що значно покращує якість подальших порад.

Центральною вимогою до функціоналу є запровадження гібридної моделі ведення діалогу, яка поєднує структурований збір даних із механізмами інтелектуальної адаптації. На початковому етапі користувач має пройти низку базових запитань для формування первинного профілю, що фіксуватиме його академічні інтереси, схильності та загальні вподобання. На відміну від звичайних фіксованих тестів, проєктована

система має передбачати перехід до динамічного етапу уточнення. На основі аналізу щойно наданих відповідей, програмний модуль генерує додаткові, більш цілеспрямовані запитання, що мають на меті поглибити розуміння професійних нахилів користувача. Цей метод сприяє підвищенню точності наданих рекомендацій.

Окрім діалогової логіки, важливою вимогою є здатність системи здійснювати інтелектуальний аналіз та зіставлення даних. Програма мусить оперувати даними про освітні програми, спеціальності та навчальні заклади, здійснюючи пошук відповідних варіантів не лише за ключовими словами, а на основі змістової відповідності інтересів користувача. Підсумком такого аналізу має бути формування розгорнутих порад із поясненням логіки обраного варіанту. Додатковою цінністю системи є надання економічного контексту для кожної рекомендованої професії. Система має забезпечувати користувачеві можливість отримати актуальну інформацію про середній рівень заробітної плати в Україні для обраного фаху. Це дозволить абітурієнту оцінити не лише академічну відповідність професії, а й її фінансову перспективність на ринку праці. Також система повинна надавати користувачеві можливість зберігати результати всієї консультації для їх подальшого перегляду.

З технічної точки зору система повинна відповідати ряду параметрів, що забезпечують її надійність функціонування у реальних умовах. Ключовою характеристикою тут є продуктивність, яка визначає швидкість обробки запитів від користувачів та формування релевантної відповіді. Система повинна забезпечувати мінімальні часові затримки, що дозволить зберегти природність діалогу.

Іншою суттєвою вимогою є висока доступність сервісу. Користувач повинен мати змогу отримати консультацію у будь-який час, не потребуючи встановлення якогось додаткового програмного забезпечення. Крім того, система має гарантувати надійність збережених даних, а також забезпечити

стабільність функціонування системи у випадках одночасної роботи з великою кількістю осіб.

Для наочного представлення виконання вищезазначених вимог доцільно використати моделювання бізнес-процесу за стандартом BPMN (рисунок 2.1).

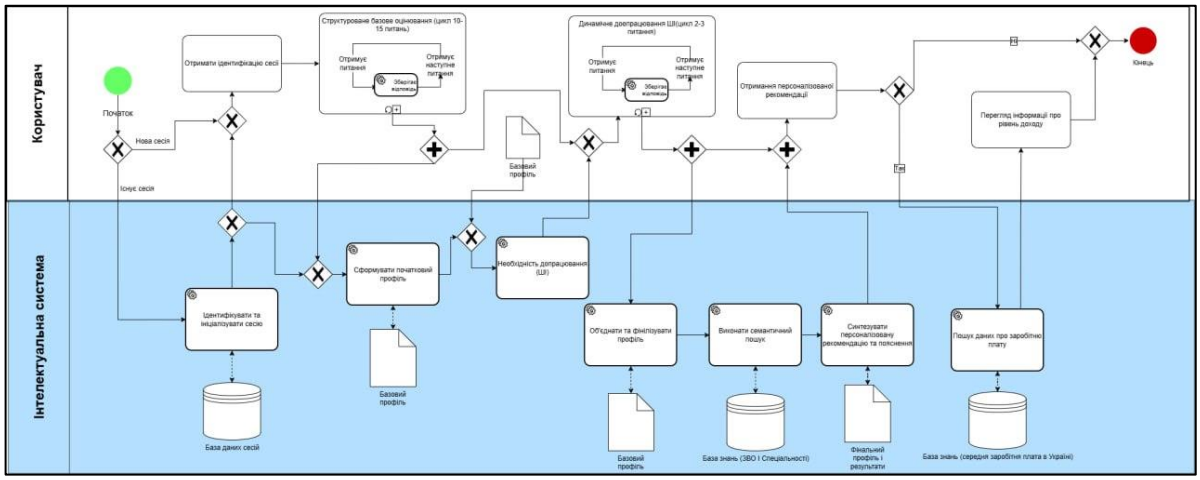


Рисунок 2.1 – Модель бізнес-процесів інтелектуальної системи профорієнтаційного консультування в нотації BPMN 2.0

Така діаграма дає змогу візуалізувати покроковий ланцюжок дій системи: від ініціювання контакту та проходження етапів тестування до моменту формування інтелектуальної поради, отримання фінансової аналітики та завершення консультаційної сесії. Графічне зображення процесу чітко визначає моменти ухвалення рішень, а також дозволяє простежити траєкторію руху даних між окремими елементами системи [14].

2.2 Аналіз методів та архітектурних підходів до реалізації компонентів системи

Спроєктована у попередньому підрозділі модель бізнес-процесів у нотації BPMN 2.0 заклала функціональний фундамент системи, визначивши

ключові етапи взаємодії абітурієнтів з інтелектуальними компонентами. Однак, ефективне впровадження цих процесів залежить від обґрунтованого вибору технологічного стеку та архітектури, що реалізується. На даному етапі постає завдання перетворити логічну структуру у конкретне інженерне рішення, що гарантуватиме баланс між складністю розробки, продуктивністю та зручністю для користувача. Нижченаведений розгляд має на меті глибоке дослідження наявних методів створення клієнт-серверних структур, способів роботи з людською мовою та підходів до структурування сховищ знань, що дозволить обрати найбільш ефективний інструментарій для досягнення поставлених цілей.

2.2.1 Аналіз платформ для реалізації користувацького інтерфейсу

Проєктування архітектури інтелектуальної системи розпочинається з аналізу платформ для реалізації користувацького інтерфейсу (UI), оскільки саме вибір технологічного середовища визначатиме зручність сервісу та можливі трудовитрати на розробку. Сучасна інженерія програмного забезпечення пропонує три основні вектори розвитку інтерфейсів, кожен з яких базується на специфічних бібліотеках та фреймворках. Для об’єктивної оцінки наявних рішень визначено ключові критерії (таблиця 2.1.)

Першим і найбільш розповсюдженим підходом є розробка вебзастосунків, де домінуюче положення займають React.js, Vue.js та Angular. Бібліотека React.js, розробленні компанією Meta, базується на концепції компонентного підходу та використанні віртуального дерева елементів (Virtual DOM). Такий підхід дає змогу створювати інтерфейси, що моментально відгукуються на дії користувача, усуваючи необхідність повного оновлення сторінки, що є незмінним для інтерактивних систем оцінювання [15].

Таблиця 2.1 – Порівняльний аналіз архітектурних підходів до реалізації UI

Критерії порівняння	Вебфреймворки (React, Vue)	Мобільні платформи (Kotlin, Flutter)	Чат-платформи (Telegram API)
Бар’єр входу для кінцевого користувача	Низький (перехід за посиланням)	Високий (потребує встановлення)	Мінімальний (месенджер)
Складність розробки (трудовитрати)	Середня (SPA архітектура)	Висока (UI + логіка)	Низька (готовий UI)
Гнучкість візуалізації	Максимальна (Custom UI)	Висока (Native/Widgets)	Стандартна (UI компоненти)
Швидкість розгортання	Середня	Низька	Висока
Кросплатформінсть	Повна (через браузер)	Часткова/Висока (Flutter)	Повна (всі ОС)

На відміну від нього, Vue.js позиціонується як фреймворк із прогресивним типом, що вирізняється низьким порогом входу та надзвичайною адаптивністю при впровадженні у вже існуючі проєкти. Завдяки своїй модульності, Vue.js дозволяє поступово нарощувати функціонал інтерфейсу, забезпечуючи високу швидкість роботи завдяки ефективному механізму реактивності [16]. У тих випадках, коли потреба проєкту полягає у наявності суворої архітектурної схеми та багатого функціоналу, найдоречнішим є застосування Angular. Цей фреймоврк від Google пропонує потужні засоби для суворої типізації даних та двостороннього зв’язку, що робить його оптимальним вибором для великомасштабних рішень на рівні підприємства, хоча він вимагає суттєво більших витрат на розробку та подальшу підтримку [17].

Іншим підходом є розробка мобільних програм, де вибір технологічного стеку визначається потребою охопити різноманітні операційні системи. Для забезпечення найвищої продуктивності на платформі Android стандартом виступає мова Kotlin. Вона пропонує сучасний синтаксис, безпеку щодо нульових посилань (null-safety) та повну сумісність із існуючим кодом на Java, що робить її незміною для нативної розробки [18]. Проте, з метою оптимізації витрат на розробку, часто

обирають кросплатформні рішення, лідером серед яких є фреймворк Flutter. Використовуючи мову програмування Dart, Flutter дозволяє створювати інтерфейси з високим рівнем деталізації, де кожен графічний елемент малюється власним рушієм рендерингу. Це гарантує ідентичний вигляд та поведінку додатка як на iOS, так і на Android, значно скорочуючи час входу продукту на ринок [19]. Незважаючи на технологічні переваги, мобільні додатки створюють бар'єр для користувача через необхідність завантаження з маркетів, що часто є недоцільним для сервісів одноразового консультування.

З огляду на специфіку діалогової взаємодії, доцільно розглянути використання розмовних інтерфейсів на базі месенджерів. Платформа Telegram Bot API надає розробникам потужний функціонал для побудови діалогових сценаріїв. Для реалізації такої взаємодії використовуються спеціалізовані фреймворки, такі як Aiogram, що пропонують високорівневі абстракції для управління станами діалогів, обробки медіафайлів та швидкої інтеграції з API [20]. Telegram забезпечує користувачам миттєвий доступ до сервісу без необхідності встановлення додаткового програмного засобу, що кардинально знижує поріг входу.

Виходячи з результатів проведеного порівняльного аналізу, можна зазначити: хоча вебзастосунки на базі React або Vue пропонують розширені візуальні здібності, а нативні мобільні рішення на Kotlin або Flutter демонструють високу швидкість роботи, найбільш раціональним варіантом для реалізації системи інтелектуальної профорієнтації є платформа Telegram разом із необхідним набором інструментів (фреймоврку Aiogram). Таким чином, обраний підхід дозволяє досягти оптимального балансу між швидкістю розробки, відсутністю витрат на підтримку серверної частини та максимальною доступністю для цільової аудиторії, що дає змогу зосередити основні інженерні зусилля на розробці інтелектуальних компонентів системи.

2.2.2 Аналіз великих мовних моделей як когнітивного ядра системи

Після визначення платформи для реалізації інтерфейсу наступним етапом – формування архітектури серверної частини (back-end), ключовим елементом якої є когнітивне ядро на базі великої мовної моделі (LLM). Саме цей компонент відповідає за розуміння запитів користувача, аналіз результатів тестування та формуванням персоналізованих рекомендацій. З огляду на актуальний стан розвитку технологій штучного інтелекту, доцільно провести порівняльний аналіз трьох найбільш потужних архітектур, що базуються на моделі Transformer та надають доступ до своїх функціональних можливостей через API. Технічні метрики моделей, що ґрунтуються на офіційних специфікаціях розробників [21], [22], [23], наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз характеристик великих мовних моделей (станом на квітень 2026)

Критерії порівняння	GPT-4o (OpenAI)	Claude 3.5 Sonnet	Gemini 2.5 Flash
1	2	3	4
Архітектурна основа	Transformer (Dense)	Transformer (MoE)	Transformer (MoE)
Вікно контексту (метрика)	128 000 токенів	200 000 токенів	1 000 000 токенів
Швидкість відповіді	Середня	Середня	Висока
Робота з українською мовою	Висока	Висока	Висока

Першим ваговим інструментом є моделі серії GPT від компанії OpenAI, зокрема модель GPT-4o. Ця модель загальновизнана як еталон у сфері міркувань та логічних висновків, демонструючи вражаючу здатність до точного виконання комплексних, багатоетапних інструкцій. Завдяки широкій підтримці спільноти та наявності розвиненої документації, інтеграція GPT у програмні рішення є стандартизованим процесом [21]. Однак, попри неабиякий інтелектуальний потенціал, застосування GPT-4o у

проєктах з великою кількістю даних може бути ускладнене високою вартістю обробки токенів та обмеженим обсягом контексту, що може призводити до втрати частини інформації при завантаженні довгих описів.

Іншим перспективним рішенням є моделі сімейства Claude від Anthropic (зокрема версія 3.5 Sonnet). Ці моделі мають незаперечну перевагу у форматуванні текстового контенту, який за манерою викладу та побудовою речень практично не відрізняється від людського мовлення. Claude відзначається високою ефективністю у роботі з такими операціями, як стислий переказ матеріалу та детальний розбір документації [22]. Однак слід зважати на певні територіальні обмеження у доступі до API, а також наявність суворіших механізмів безпеки, які час від часу спричиняють надміру відмову у надані відповідей на запити користувачів, які потребують високої технічної деталізації.

Спираючись на метрики, наведені у звіті компанії Google [23], варто приділити увагу моделі Gemini 2.4 Flash. Її головною перевогаю є вікно контексту обсягом 1 000 000 токенів. Це дозволяє опирувати детальним описами професій без ризику втрати контексту, що пітверджується результатами тестів на вилучення інформації «Needle In A Haystack» [23]. Версія Flash спеціально оптимізована для забезпечення мінімальної затримки при генерації відповіді, що є життєво необхідним для збереження плавності комунікації у месенджері. На додачу, Gemini демонструє високу адаптивність у роботі з українською мовою.

Порівняльний аналіз демонструє, що кожна з розглянутих моделей має високий потенціал для реалізації інтелектуальних функцій. Проте, враховуючи специфіку завдань профорієнтаційного консультування – а саме необхідність одночасної обробки значних обсягів інформації про університети та ринок праці при збереженні високої швидкості відповіді – найбільш обґрунтованим є використання моделі Gemini 2.5 Flash. Її архітектурні переваги дозволять створити стабільне та інтелектуально потужне ядро для майбутньої системи.

2.2.3 Аналіз методів та інструментів збереження даних

Визначення когнітивного ядра системи у попередньому підрозділі ставить наступне завдання: забезпечити моделі Gemini актуальними та достовірними даними. Головною проблемою сучасних LLM є схильність до «галюцинацій» – генерація неперервної інформації. Для побудови надійного профорієнтаційного консультанта необхідно впровадити архітектурний метод, який дозволить моделі звертатися до зовнішніх перевірених джерел інформації в реальному часі. Порівняння підходів, що базується на фундаментальних дослідженнях у галузі NLP [24], наведено у таблиці 2.3.

Таблиця 2.3 – Порівняльний аналіз методів актуалізації знань LLM

Критерій порівняння	Донавчання моделі (Fine-tuning)	Доповнене пошуком генерування (RAG)
Актуальність даних	Тільки на момент навчання	Реальний час (динаміка)
Ресурсні витрати	Високі (GPU)	Низькі (API)
Прозорість (цитування)	Відсутні	Наявна (посилання)

Виходячи з результатів порівняння, раціональним підходом даної задачі є метод RAG. Ідея методу полягає в тому, що перед формуванням відповіді система виконує пошук відповідних фрагментів інформації у зовнішній базі знань і додає їх до запиту користувача як контекст. Виходячи з цього, використання RAG гарантує підвищену коректність, можливість оновлення знань без перенавчання моделі та верифікацію відповідей [24].

Реалізація цього методу неможлива без використання векторних баз знань, які зберігають інформацію у вигляді числових векторів. Для вибору оптимального інструменту розглянуто два рішення (таблиця 2.4). Серед існуючих рішень виділяється ChromaDB – векторна СУБД з відкритим кодом, оптимізована для локальної роботи з великими мовними моделями. Вона дає змогу зберігати вектори та якнайшвидше проводити їхнє

порівняння на основі змістової близькості, що гарантує виявлення відповідних професій навіть тоді, коли у запиті немає жодного прямого текстового збігу [25].

Таблиця 2.4 – Порівняння інструментів векторного пошуку

Критерій	ChromaDB (Локальна)	Pinecone (Хмарна)
1	2	3
Локалізація даних	Локально / На сервері	Віддалений сервер
Залежність від мережі	Мінімальна	Повна
Вартість підтримки	Безкоштовно	Безкоштовно базовий рівень
Швидкість	Висока (локальний доступ)	Висока (оптимізована хмарна інфраструктура)

Проте розгортання локальної векторної бази створює додаткове навантаження на оперативну пам’ять та обчислювальні ресурсу хостингу. Альтернативою є хмарне сховище Pinecone, яке пропонує повністю керовану безсерверну інфраструктуру для високошвидкісної обробки багатовимірних векторів у хмарі [26]. Хоча Pinecone потребує стабільного доступу до мережі для взаємодії через API, ця СУБД знімає навантаження з сервера чат-бота, надійно підтримує ізольовані простори для різних наборів даних та надає повноцінний безкоштовний рівень, його цілком достатньо для базового функціонування та подальшого масштабування системи консультування.

Окрім інтелектуального пошуку знань, архітектура системи потребує управління структурованими даними, до яких належать ідентифікатори користувачів, історія відповідей та поточні стани сесій. Класичним інструментом для опрацювання подібних потреб вважаються реляційні бази даних (SQL). Зокрема, SQLite є дієвим варіантом для систем, яким потрібне локальне збереження інформації, усуваючи потребу у запуску окремого серверного процесу. Ця СУБД зберігає всю інформацію. В одному файлі, що суттєво спрощує архітектуру та забезпечує високу швидкість операцій читання та запису при помірних навантаження [27]. Однак, коли йдеться про

системи, яким необхідна значна здатність до масштабованості та одночасної роботи великої кількості користувачів з розгалуженими правами доступу, доцільно розглянути PostgreSQL. Це потужна об’єктно-реляційна система, що підтримує складні типи даних та гарантує високу цілісність інформації в розподілених середовищах [28]. Незважаючи на потужність PostgreSQL, для початкових етапів розробки інтелектуального асистента переваги SQLite у простоті налаштування та швидкості отримання даних є більш ваговим аргументом.

Додатковим аспектом є збереження статичної ринкової інформації, такої як звіти про рівень заробітних плат. Для таких завдань використання промислових баз даних може бути зайвим. Найкращим вибором у цій ситуації стають формати JSON (JavaScript Object Notation) та CSV (Comma-Separated Vaulues). JSON чудово підходить для відображення структур даних, оскільки він легко зчитується програмами та забезпечує компактність інформації [29]. Водночас, CSV досі є еталоном для зберігання даних у вигляді таблиць та можливості легкої інтеграції з будь-якими аналітичними інструментами [30]. Використання таких форматів дозволяє швидко оновлювати економічний контекст системи без необхідності зміни схеми основної бази даних.

Таким чином, проведений аналіз демонструє, що для забезпечення достовірності консультацій та ефективної реалізації методу RAG найбільш обґрунтованим є вибір хмарної векторної СУБД Pinecone. Це дозволить системі виконувати точний семантичний пошук за багатовимірними векторами ознак без перевантаження ресурсів сервера. Для управління сесіями користувачів найбільш раціональним є вибір локальної СУБД SQLite, а для збору та оновлення допоміжних економічних даних – формати JSON та CSV. Така гібридна модель збереження та пошук даних забезпечить необхідну точність персоналізованих рекомендацій та стабільність роботи сервісу.

2.2.4 Аналіз мов програмування та інструментарію реалізації

Після обрання Telegram як основної платформи для взаємодії з абітурієнтами (п. 2.2.1) та визначення методів обробки знань (п. 2.2.3), необхідно провести аналіз мов програмування для розробки серверної частини. Саме мова програмування має забезпечити стабільну інтеграцію між інтерфейсом месенджера, базою даних та інтелектуальним ядром. Для об’єктивного вибору порівняємо три поширені інструменти за ключовими для проєкту критеріями (таблиця 2.5).

Таблиця 2.5 – Порівняльний аналіз мов програмування для розробки сервісу

Критерій порівняння	Python	Node.js (JavaScript)	Java
Екосистема для ШІ	Найвища	Середня	Помірна
Швидкість розробки	Висока	Висока	Низька
Обробка асинхронності запитів	Висока (asyncio)	Максимальна (Event Loop)	Висока (Threads)
Трудовитрати на підтримку	Низькі	Середні	Високі

Згідно з індексом TIOBE та опитування Stack Overflow [31], посідає провідні позиції серед інструментів для створення інтелектуальних мережевих програм. Ключова перевага полягає у високій адаптивності до побудови складних, але при цьому зручних в обслуговуванні систем, що базуються на принципах об’єктно-орієнтовного програмування. Завдяки чіткій структурі класів та ефективним методам роботи з інформацією, застосування Python забезпечує створення стійких застосунків і бібліотек, що є важливим для стабільної роботи інтелектуальних систем [32]. Python на сьогодні є основним інструментом для роботи з AI-сервісами, оскільки він пропонує найбільш розвиненні механізми інтеграції з API великих мовних моделей, бібліотеками для обробки природньої мови та векторних сховищ. Висока читабельність коду та велика кількість готових модулів дозволять значно скоротити час на розробку бізнес-логіки чат-бота.

Іншим потужним напрямом є використання JavaScript, зокрема серверного середовища Node.js. Аналіз архітектурних рішень показує, що ключову перевагу цієї платформи складають її неблокуючий механізм вводу/виводу та принцип роботи через цикл подій [33]. Така архітектура дозволяє ефективно обробляти тисячі одночасних запитів від користувачів месенджера без значних витрат серверних ресурсів, що забезпечить високу масштабованість системи. Крім того, використання JavaScript як універсальної мови спрощує структурування коду та дозволяє безперешкодно переходити від розробки клієнтських вебінтерфейсів до реалізації складної серверної логіки [34]. Це робить Node.js чудовим вибором для систем, де пріоритетом є швидкість реакції та робота в режимі реального часу.

Аналіз мови Java демонструє її ключове значення у сфері розробки програмного забезпечення для бізнесу. Властивість кросплатформності, яку забезпечує віртуальні машина Java (JVM), є однією з її сильних сторін, гарантуючи при цьому стійкість функціонування в реальних виробничих середовищах [35]. Сувора типізація та розвиненні механізми управління пам’яттю дозволяють уникати багатьох помилок на етапі компіляції, що робить Java надійним інструментом для розробки великих систем із жорсткими вимогами до безпеки. Проте, попри архітектурну перевагу, для оперативного внесення змін у логіку діалогових потоків Java вимагає значно більшої кількості програмного коду у порівнянні з мовами з динамічною типізацією, що може уповільнити процес створення робочих прототипів.

Загальний порівняльний огляд сучасних мов програмування підтверджує, що вибір часто залежить від балансу між продуктивністю виконання та швидкістю розробки [36]. Для проєктів, де ключовим фактором є інтеграція з ШІ-технологіями та гнучкість налаштування логіки, доцільно обирати мови з низьким порогом входу та потужною підтримкою сучасних хмарних сервісів [37]. Вибір між продуктивністю серверної

частини та швидкістю ітерації розробки є визначальним етапом проєктування інтелектуального помічника.

Таким чином, проведений аналіз дозволяє зробити висновок, що для реалізації Telegram-бота з елементом штучного інтелекту найбільш раціональним є використання мови Python. Вона забезпечує найкращу підтримку асинхронності для чат-ботів та має найбільш розвинену екосистему для роботи з неструктурованими даними та зовнішніми API.

2.3 Обґрунтування вибору технологічного набору

На основі проведеного аналізу сучасних методів і технологій, було сформовано підсумковий технологічний стек для розробки інтелектуального сервісу. Для наочного представлення обраних компонентів та їх зон відповідальності розроблено таблицю 2.6.

Таблиця 2.6 – Підсумковий перелік технологічного стеку системи

Зона відповідальності	Технологія/Інструмент	Основне обґрунтування/Метрика
1	2	3
Back-end (мова)	Python	Найвища адаптивність до AI-сервісів та асинхронності
Користувацький UI	Telegram Bot API	Мінімальний бар’єр входу та асинхронні обробки
Когнітивне ядро	Gemini 2.5 Flah	Вікно контексту 1 000 000 токенів
Векторна СУБД	Pinecone	Хмарний семантичний пошук для архітектури RAG
Реляційна СУБД	SQLite	Простота архітектури
Формат даних	JSON / CSV	Легкість оновлення статичної інформації

Виходячи з результатів аналізу, мова програмування Python визначена як основний інструмент для розробки серверної частини. Вибір зумовлений її домінуючим становищем у сфері штучного інтелекту та наявності офіційних SDK для роботи з великими мовними моделями. Екосистема Python дозволяє легко інтегрувати логіку чат-боті з інструментами обробки

природної мови та векторними сховищами, що значно пришвидшує етапи розробки та подальшого тестування системи.

Для реалізації користувацького інтерфейсу в месенджері Telegram обрано фреймворк Aiogram. Його головна перевага полягає у застосуванні парадигми асинхронного програмування, що базується на asyncio. Це важливо для інтелектуальних систем, оскільки дозволяє серверу обробляти запити сотень користувачів одночасно, не блокуючи основний потік під час очікування відповіді від віддалених API-сервісів штучного інтелекту.

Когнітивним ядром системи обрано модель Gemini 2.5 Flash, розроблену компанією Google. Ця модель поєднує в собі швидкість генерації та здатність оперувати надзвичайно великим вікном контексту. Останній фактор є вирішальним для реалізації архітектури RAG, оскільки дозволяє моделі аналізувати розлогі описи освітніх програм та професій без втрати деталей, забезпечуючи високу точність персоналізованих рекомендацій.

Для збереження неструктурованих даних, які стосуються закладів вищої освіти, освітніх програм, спеціальностей, а також актуальної статистики ринку праці, було обрано хмарну векторну СУБД Pinecone. Вона дозволяє виконувати семантичний пошук, порівнюючи векторні представлення запитів користувача із наявними документами. Делегування обчислювального навантаження на надійну хмарну інфраструктуру, ефективна робота з розширеними метаданими та можливість гнучкого адміністрування ізольованих індексів роблять Pinecone найбільш раціональним вибором для забезпечення швидкодії та масштабованості системи в межах архітектури RAG.

Для збереження структурованих даних користувачів, результатів профорієнтації та контролю за статусом діалогових сесій, обрано СУБД SQLite. Ця система керування базами даних (СУБД) є простою у використанні та не вимагає ініціалізації окремого сервера бази даних, адже функціонує як локальний файл.

Злагоджене застосування обраних технологічних рішень надасть змогу створити надійну, продуктивну та масштабовану систему профорієнтаційного консультування, здатну до ефективної комунікації з користувачами та обробки великого обсягу інформації.

Для цілісного представлення взаємодії обраних технологічних рішень та візуалізації архітектури сервісу розроблено діаграму контейнерів за моделлю C4 (рівень 2). Схема демонструє внутрішню структуру системи, де центральний застосунок на базі Python виступає основним компонентом логіки. Він забезпечує асинхронний обмін даними між зовнішнім інтерфейсом месенджера Telegram, когнітивним сервісом Google Gemini та гібридним сховищем, що складається з хмарної векторної бази Pinecone та реляційної бази SQLite (рисунок 2.2). Такий підхід дозволяє чітко розмежувати відповідальність кожного компонента та забезпечити стабільність системи. Використання побудованої контейнерної архітектури не лише фіксує поточний стан інтелектуального сервісу, а й закладає надійний інженерний фундамент для його масштабування.

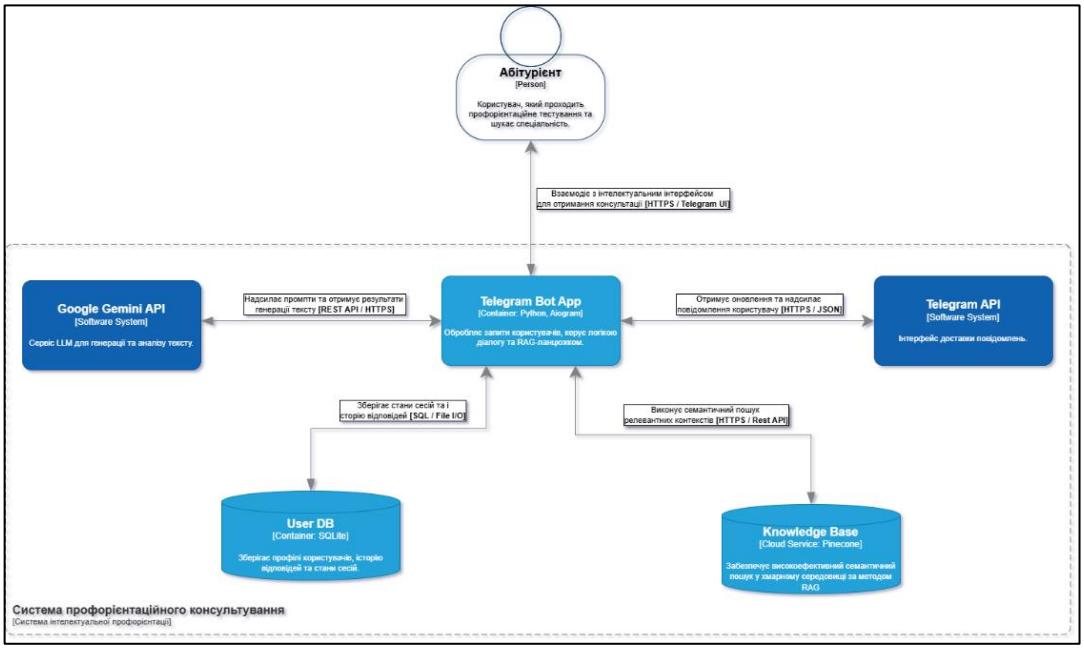


Рисунок 2.2 – Діаграма контейнерів системи інтелектуальної профорієнтації (C4 Level2)

У другому розділі було проведено детальний аналіз методів зберігання інформації та технологій штучного інтелекту, що дозволило визначити оптимальну стратегію побудови системи. На основі аналізу функціональних вимог сформовано архітектурну модель, яка відображає взаємодію ключових компонентів: інтерфейсу Telegram, когнітивного ядра на базі моделі Gemini 2.5 Flash та гібридної підсистеми зберігання даних Pinecone та SQLite. Побудована діаграма контейнерів C4 деталізувала структуру системи, чітко окресливши вузли обробки семантичних запитів та механізми забезпечення цілісності даних.

Таким чином, розроблені архітектурні принципи та обґрунтований вибір технологічного стеку створюють надійний інженерний фундамент для практичної реалізації проєкту. Систематизація логіки роботи та архітектурне проєктування дозволить мінімізувати ризики виникнення програмних помилок, що дає змогу перейти до безпосередньої програмної реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ІНТЕГРАЦІЯ ІНТЕЛЕКТУАЛЬНИХ МОДУЛІВ СИСТЕМИ

3.1 Структура програмного проекту та обґрунтування технологічного стеку

Практична реалізація розробленої профорієнтаційної системи безпосередньо базується на архітектурних рішеннях, обґрунтованих у попередньому розділі, зокрема на контейнерному рівні моделі C4 та концепції гібридного збереження інформації. Для втілення цієї архітектури було обрано мову програмування Python (версія 3.11.9), що зумовлено її розвиненою інфраструктурою асинхронного програмування, наявністю потужних інструментів для аналізу даних та офіційних оптимізованих SDK для роботи з хмарними векторними базами й моделями генеративного штучного інтелекту.

Для забезпечення модульності, зручності тестування та дотримання принципу єдиної відповідальності програмний проект було організовано у вигляді чіткої ієрархії директорій. Такий підхід дозволив повністю ізолювати логіку користувацького інтерфейсу від модулів скрейпінгу сайтів та сервісів взаємодії з ШІ.

Вибір такого методу організації коду обумовлений потребою у гнучкому управлінні життєвим циклом інтелектуального сервісу. Оскільки система поєднує різні за своєю природою завдання – фоновий парсинг даних, обробку запитів через месенджер, виконання семантичного пошуку та взаємодію з хмарними нейромережами – монолітна структура коду суттєво ускладнила б підтримку та розширення функціоналу. Розподіл компонентів за функціональними директоріями реалізує патерн розділення відповідальності. Це дозволяє незалежно оновлювати алгоритми збору знань, коригувати логіку взаємодії з мовною моделлю або вносити зміни до структури бази даних, не створюючи при цьому ризиків для стабільності

всього проекту. Такий структурний підхід також мінімізує витрати часу на налагодження та спрощує інтеграцію нових модулів у майбутньому. Загальну структуру папок та ключових файлів реалізованого проекту наведено на рисунку 3.1.

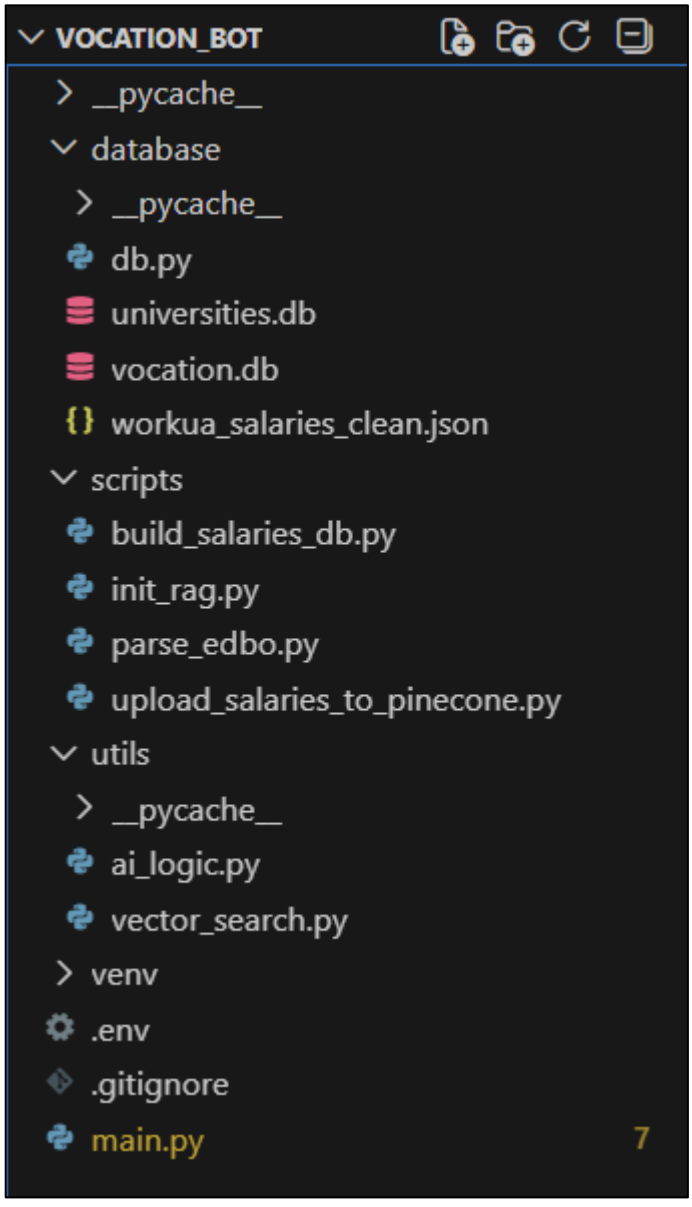


Рисунок 3.1 – Структура проекту

Як показано на рисунку 3.1, корінь проекту містить головний скрипт запуску main.py, файл конфігурації оточення .env, де безпечно зберігаються ключі доступу до API, та файлів залежності requirements.txt. Інші

компоненти системи структуровані відповідно до їх функціонального призначення.

Директорія database/ містить реляційну базу даних vocation.db та модуль db.py для управління локальними таблицями користувачів. Директорія scripts/ ізолює автономні утілити для збору даних (parse_edbo.py, build_salaries_db.py) та скрипти первинного наповнення хмарних індексів (init_rag.py, upload_salaries_to_pinecone.py). Директорія utils/ містить сервісні модулі: ai_logic.py (робота з генерацією контенту через Gemini) та vector_search.py (виконання семантичного пошуку в Pinecone).

Стабільна взаємодія компонентів системи забезпечується використанням спеціалізованих бібліотек та фреймворків, кожен із яких був обраний відповідно до поставлених інженерних завдань.

Бібліотека aiogram використовується як асинхронний фреймворк для роботи з Telegram API. Її застосування дозволило реалізувати неблокуючу обробку запитів від кількох користувачів одночасно. Додатково вбудований механізм Finite State Machine (FSMContext) забезпечує контроль етапів проходження тестування та відстеження поточного стану користувача під час діалогу.

Для інтеграції з мовними моделями використовується бібліотека google-genai – офіційний SDK компанії Google для доступу до сервісів штучного інтелекту. Через неї реалізовано взаємодію з моделлю gemini-2.5-flash для аналізу тексту та генерації відповідей, а також з моделлю gemini-embedding-2 для створення векторних представлень текстових даних.

Бібліотека pinecone забезпечує взаємодію з хмарною векторною базою даних Pinecone. Вона використовується для виконання семантичного пошуку та обчислення подібності між векторними представленнями професії та даними про освітні програми в межах RAG-підходу.

Для збору та обробки даних із вебресурсів застосовано бібліотеки selenium і beautifulsoup4. Selenium використовується для керування браузером у фоновому режимі для виконання JavaScript на динамічних

вебсторінок, тоді як beautifulsoup4 забезпечує вилучення необхідної текстової інформації з HTML-коду сторінок.

Для локального збереження даних використовується вбудований модуль sqlite3, який забезпечує роботу з реляційною базою даних SQLite. Його використання є доцільним завдяки невисоким вимогам до ресурсів та достатній продуктивності для збереження даних користувачів і результатів тестування.

Логіка розробки та запуску проекту будувалась за принципом «спочатку дані – потім інтерактивна взаємодія». Оскільки функціонування системи на основі RAG(Retrieval-Augmented Generation) потребує наявності попередньо підготовленої бази знань, процес реалізації було поділено на два основні етапи.

Перший етап – підготовчий (офлайн). На цьому етапі здійснювалось формування бази знань системи. Спочатку були запущені автономні парсери parse_edbo.py та build_salaries_db.py, які зібрали сиру інформацію з сайтів у локальні файли. Далі виконувалася процедура векторизації за допомогою скриптів init_rag.py та upload_salaries_to_pinecone.py, у межах якої текстові дані були перетворені на векторні представлення та завантаженні у хмарні індекси Pinecone (universities-v2 та salaries-index).

Другий етап – операційний (онлайн). Після формування та перевірки векторної бази знань було запущено основного маршрутизатора main.py. У режимі роботи користувача Telegram-бот не виконує зовнішній збір даних у реальному часі, а звертається до вже підготовлених локальних баз SQLite та векторних індексів Pinecone. Це забезпечує швидке формування відповідей та мінімальний час очікування під час проходження профорієнтаційного тестування.

Однак, успішне функціонування цього операційного етапу та висока точність інтелектуальних рекомендацій безпосередньо залежить від якості первинної бази знань. Тому критично важливим кроком стало реалізація

ефективних механізмів для вилучення та структурування сирих масивів інформації, логіку роботи яких необхідно розглянути детальніше.

3.2 Алгоритм автоматизованого збору даних та парсингу зовнішніх платформ

Для забезпечення інтелектуального модуля системи (RAG-архітектури) актуальною фактологічною інформацією було розроблено підсистему веб-скрейпінгу, яка функціонує на етапі офлайн-підготовки. Її основне завдання – автоматизований збір, очищення та первинне структурування даних із двох незалежних джерел. Оскільки архітектура цільових веб-ресурсів суттєво відрізняється(один використовує клієнтський рендеринг, інший – статичний HTML), для кожного було розроблено та реалізовано окремий алгоритмічний підхід.

3.2.1 Автоматизований збір освітніх програм із використанням технології імітації браузера

З метою наповнення бази даних вихідною інформацією для підсистеми семантичного пошуку та реалізації RAG-архітектури виникла потреба завантажити необхідні масиви даних про спеціальності, освітні ступені та географічне розташування закладів вищої освіти (ЗВО). Основним джерелом такої інформації було обрано офіційний реєстр Єдиної державної електронної бази з питань освіти (ЄДБО).

Процес збору, очищення та надійного збереження цих даних у межах файлу parse_edbo.py логічно поділяється на три ключові етапи: налаштування фонового режиму роботи браузера, обхід асинхронного завантаження з вилучення метаданих та багаторівневу фільтрацію структури з обробки станів.

Головною технічною перешкодою під час роботи с реєстром ЄДБО стала наявність клієнтського рендерингу. Цільові таблиці зі списками спеціальностей не містяться у первинному HTML-коді сторінки, а завантажуються динамічно за допомогою JavaScript-запитів вже після відкриття сторінки. Оскільки статичні GET-запити повертали порожні структури, в архітектурі модуля для імітації дій реального користувача було застосовано інструмент Selenium Webdriver.

Ключовим інженерним рішенням на етапі ініціалізації середовища всередині керуючої функції main() є оптимізація споживання оперативної пам’яті сервера та забезпечення автономності у фоновому headless-режимі. Для запобігання системи збоям примусово вимикається апаратне прискорення та фіксується віртуальна роздільна здатність екрана для коректного відображення елементів адаптивної верстки. Цей етап конфігурації наведено у лістингу 3.1.

Лістинг 3.1 – Налаштування headless-режиму та ініціалізація Webdriver

```
    def main():
        db_conn = init_db()
        print("Запускаю браузер (це займе кілька секунд)...")
        chrome_options = Options()
        chrome_options.add_argument("--headless")
        chrome_options.add_argument("--disable-gpu")
        chrome_options.add_argument("--window-size=1920,1080")
        chrome_options.add_argument("--log-level=3")
        driver =
webdriver.Chrome(service=Service(ChromeDriverManager().install
()), options=chrome_options)
        try:
            for url in URLS:
                parse_page(driver, url, db_conn)
        finally:
            driver.quit()
            db_conn.close()
```

Після запуску циклічного обходу масиву URLS, курування передається функції parse_page(). Головним рішенням для подолання проблем асинхронного підвантаження даних став запуск механізму явних очікувань WebdriverWait у поєднанні з класом очікуваних умов expected_conditions. Програма блокує потік поєднання до моменту появи елементів таблиці з CSS-селектером tbody tr.

Після стабілізації DOM-дерева сирий код зчитується через властивість driver.page_source і передається об’єкту BeautifulSoup. На першому рівні аналізу скрипт динамічно вилучає назву ЗВО з тегу <h1>. Для точного визначення географічного розташування застосовується семантичний пошук регулярним виразом r"м\.\s*([А-ЯІЇЄ][а-яіїє\w-]+)", який сканує весь текст сторінки. Ізолюючи назву населеного пункту від префікса, що демонструє лістинг 3.2.

Лістинг 3.2 – Механізм явного очікування та вилучення глобальних метаданих

```
    def parse_page(driver, url, conn):
        print(f"Отримую дані: {url}")
        try:
            driver.get(url)
            WebDriverWait(driver, 15).until(
                EC.presence_of_element_located((By.CSS_SELECTO
R, "tbody tr"))
            )
            time.sleep(1.5)
            html = driver.page_source
            soup = BeautifulSoup(html, "html.parser")
            univ_name = "Невідомий ЗВО"
            h1 = soup.find("h1")
            if h1: univ_name = h1.text.strip()
            city = "Україна"
            city_search = re.search(r"м\.\s*([А-ЯІЇЄ][а-
яіїє\w-]+)", soup.get_text())
            if city_search: city = city_search.group(1)
```

На другому рівні аналізу програма ітерує по всіх рядках <tr>. Оскільки в структурі реєстру ЄДБО чергується інформаційні рядки заголовків рівнів освіти та безпосередньо списки спеціальностей, було впроваджено алгоритм відстеження поточного стану. Логічне відсікання маркерів забезпечується умовою len(row.find_all("td")) < 3 (рядки заголовків займають менше трьох комірок). При виявленні ключових слів скрипт перемикає прапорець стану current_degree на «Бакалавр» або «Магістр», що дозволяє коректно категорізувати всі наступні записи.

Вилучення числового шифру спеціальності та відокремлення його від текстової назви реалізовано через регулярний вираз r"\((\d+)\)", який ізолює цифри у круглих дужках за допомогою методу group(1). Очищенні змінні передаються в SQL-інструкцію, а метод conn.commit() викликається суворо після повної обробки сторінки конкретного університету, що забезпечує збереження даних без втрат і збоїв (лістинг 3.3).

Лістинг 3.3 – Алгоритм відстеження стану освітнього ступення та SQL-імпорт

```
rows = soup.find_all("tr")
cur = conn.cursor()
current_degree = None
count = 0
for row in rows:
row_text = row.get_text().strip().upper()
if "БАКАЛАВР" in row_text and len(row.find_all("td")) < 3:
current_degree = "Бакалавр"
continue
elif "МАГІСТР" in row_text and len(row.find_all("td")) <
3:
current_degree = "Магістр"
continue
if current_degree == "Бакалавр":
cols = row.find_all("td")
if len(cols) >= 2:
raw_spec = cols[0].get_text(strip=True)
```

Продовження лістингу 3.3

```
                        code_match = re.search(r"\((\d+)\)",
raw_spec)
                        if code_match:
                            spec_code = code_match.group(1)
                            spec_name = raw_spec.split(')')[-
1].strip()
                            cur.execute("""
                            INSERT INTO study_programs
(university_name, specialty_code, specialty_name, degree,
city, source_url)
                            VALUES (?, ?, ?, ?, ?, ?)
                            """, (univ_name, spec_code,
spec_name, current_degree, city, url))
                            count += 1
            conn.commit()
```

Кінцевим результатом виконання розробленого модуля є автоматичне наповнення локальної бази даних SQLite (database/universities.db) в таблицю study_programs. Використання реляційного сховища на цьому етапі є найбільш раціональним інженерним рішенням, оскільки воно гарантує строгу типізацію та структуру цілісність даних перед їхнім подальшим перетворення у багатовимірні вектори.

Кожна транзакція фіксує суворий набір атрибутів: (первинний ключ), (назва ЗВО), (числовий шифр галузі), (текстове розшифрування спеціальності), (освітній ступінь), (географічна прив’язка) та (пряме посилання для верифікації), повністю закриває потребу системи в освітньому фактологічному контексті.

Однак, комплексна профорієнтація вимагає не лише підбору академічних шляхів, але й оцінок реальних перспектив працевлаштування. Тому, після успішного формування бази закладів вищої освіти, наступним кроком стала розробка механізму для вилучення фінансово-аналітичних даних з ринку праці.

3.2.2 Парсинг та структурування інформації про заробітні плати

Паралельно з накопиченням інформації про заклади вищої освіти виникла необхідність підготовці додаткового масиву даних – актуальних даних про рівень заробітних плат та ринкового попиту на фахівців в Україні. Основним джерелом для моніторингу фінансових показників було обрано кадровий портал Work.ua. Це дозволяє системі пов’язувати вибір майбутньої професії з реальними можливостями заробітку.

Процес збору, фільтрації та локального збереження фінансових маркерів у межах файлу build_salaries_db.py логічно поділяється на три ключові етапи: налаштування параметрів зв’язку з перевіркою відповідей сервера, цільовий розбір структури сторінки для пошуку вакансій та очищення отриманого списку від повторюваних записів.

На відміну від реєстру ЄДБО, архітектура цільових сторінок кадрового порталу Work.ua (розділ /salary-/) базується на статичному рендерингу на стороні сервера. Це дозволило відмовитися від ресурсомісткого інструменту Selenium на користь зв’язки бібліотеки requests та BeautifulSoup. Для запобігання блокуванню запитів з боку захисних систем порталу, в архітектурі модуля зафіксовано масив констант CATEGORY_URLS, а в HTTP-запит примусово інтегрується заголовок User-Agent.

Ключовим рішенням на етапі запуску сесії є обов’язкова перевірка успішності HTTP-відповіді від сервера перед початком обробки сторінки. Якщо сервер повертає код, відмінний від 200 (успіх), алгоритм негайно перериває поточну ітерацію за допомогою оператора continue, запобігаючи передачі порожнього або пошкодженого контенту в синтаксичний аналізатор. Цей фрагмент керуючої логіки наведено у лістингу 3.4.

Після успішного завантаження сторінки об’єкт BeautifulSoup будує локальне дерево елементів для аналізу. Головним інженерним рішенням у процесі парсингу є точкова фільтрація тегів <a>. Замість повного

сканування текстового масиву сторінки, програма шукає лише ті посилання, які мають специфічний CSS-клас chart-horizontal та містять атрибут href.

Лістинг 3.4 – Конфігурація запиту та обробка HTTP-статусів

```
      headers = {
             "User-Agent": "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36"
         }
         clean_salaries_list = []
         seen_titles = set()
         for url in CATEGORY_URLS:
             print(f"Опрацювання: {url}...")
             try:
                 response = requests.get(url, headers=headers,
timeout=10)
                 if response.status_code != 200:
                     continue
                 soup = BeautifulSoup(response.text,
'html.parser')
```

Для безпосереднього вилучення корисного навантаження всередині знайденого контейнера виконується вкладений пошук за унікальним маркером верстки порталу. Назва професії ізолюється через елемент div із класом chart-category, а числове значення середнього доходу – через клас chart-data-digits. Будь-які аномалії верстки або порожні блоки обробляються захисними конструкціями, що ілюструє лістинг 3.5.

Лістинг 3.5 – Ізоляція цільових сутностей у DOM-структурі сторінки

```
     for a_tag in soup.find_all('a', class_='chart-horizontal',
href=True):
                     href = a_tag['href']
                     if '/salary-' in href and not
href.endswith('/stat/') and not 'page=' in href:
                         title_div = a_tag.find('div',
class_='chart-category')
                         if not title_div:
           continue
```

Продовження лістингу 3.5

```
                        job_title = title_div.text.strip()
                        salary_div = a_tag.find('div',
class_='chart-data-digits')
                        if not salary_div:
                            continue
                        clean_salary = salary_div.text.strip()
```

Головним технічним викликом під час збору фінансової статистики став статичний дубляж ідентичних посад у різних категоріях та на послідовних сторінках списку на вебпорталі. Для забезпечення чистоти даних та оптимізації оперативної пам’яті в алгоритм інтегровано структуру даних хеш-множин (set), ініціалізовану як змінна seen_titles.

Перед додаванням об’єкта до фінального масиву скрипт виконує умовну перевірку наявності ключа у множині. Завдяки математичним властивостям хеш-таблиць, складність цієї операції становить O(1), що гарантує високу швидкість обробки незалежно від об’єму вибірки. У разі успішного проходження фільтра, дані додаються до списку clean_salaries_list та експортується у плоский JSON-файл (лістинг 3.6).

Лістинг 3.6 – Реалізація алгоритму дедуплікації та збереження очищених профілів у JSON

```
    if job_title and clean_salary and job_title not in
seen_titles:
                        full_job_url =
f"https://www.work.ua{href}" if href.startswith('/') else href
                        seen_titles.add(job_title)

                        clean_salaries_list.append({
                            "title": job_title,
                            "salary": clean_salary,
                            "url": full_job_url
                        })
    with open("database/workua_salaries_clean.json", "w",
encoding="utf-8") as f:
            json.dump(clean_salaries_list, f,
ensure_ascii=False, indent=2)
```

Кінцевим результатом виконання розробленого автоматизованого алгоритму збору даних є повне формування локального файлу database/workua_salaries_clean.json. Оскільки дані про рівні заробітних плат є лінійним і не мають складних ієрархічних зв’язків, вибір формату JSON є набільш раціональним рішенням, яке забезпечує швидке перетворення структурованих даних у програмні об’єкти та високу швидкість обробки інформації в операційному режимі чат-бота. Кожен об’єкт містить канонічний набір полів (title, salary, url), що робить його повністю сумісним із подальшим модулем векторного кодування.

Для наочної демонстрації кінцевої структури зібраних даних, на рисунку 3.2 наведено графічну схему згенерованого масиву JSON. Ця схема ілюструє принцип збереження лінійного списку вакансій, де кожен елемент містить чітку структуровані атрибути посади, її фінансової оцінки та посилання на джерело.

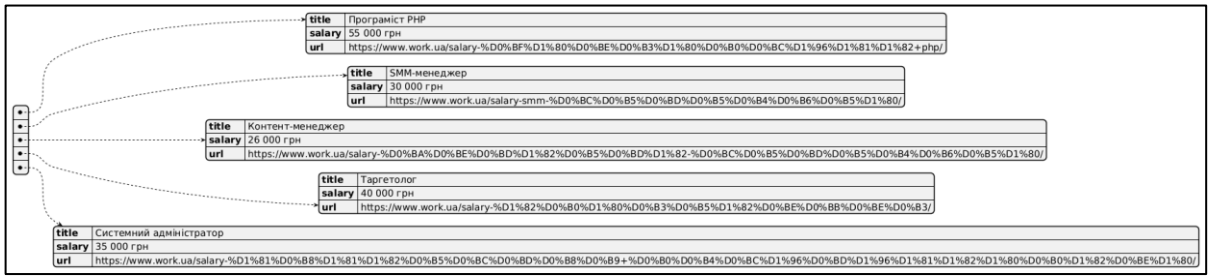


Рисунок 3.2 – Структура JSON-файлу з даними про заробітні плати

Таким чином, обидва ключові джерела інформації – бази освітніх програм та фінансової аналітики – були успішно зібрані, очищені та локально збережені. Це завершує етап класичної підготовки даних і створює необхідне підґрунтя для переходу до наступного етапу – семантичної трансформації, яка передбачає векторизацію зібраних текстових даних за допомогою мовних моделей та їх подальше завантаження до векторної бази

Pinecone для забезпечення ефективного семантичного пошуку в межах RAG-пайплайну.

3.3 Інтеграція підсистеми штучного інтелекту та реалізація RAG-архітектури

3.3.1 Генерація векторних подань та ініціалізація індексів Pinecone

Для забезпечення можливості виконання швидкого семантичного пошуку системою штучного інтелекту виникла потреба в перетворені зібраних текстових даних у багатовимірні числові вектори (embeddings). Цей етап офлайн-підготовки реалізовано у двох автономних скриптах: init_rag.py (векторизація освітніх програм) та upload_salaries_to_pinecone.py (векторизація заробітних плат). Процес інтеграції даних у хмарне векторне сховище включає підготовку інфраструктури, структурування метаданих та безпечну генерацію векторів через API.

Перед початком безпосереднього завантаження даних система має автоматично впевнитися у наявності цільових індексів у хмарному середовищі. У файлі upload_salaries_to_pinecone.py реалізовано логіку динамічної перевірки інфраструктури: програма звертається до методу pc.list_index(), і якщо цільовий індекс відсутній, ініціює його створення. Ключовим інженерним рішенням на цьому етапі є явне визначення розмірності векторного простору (dimension=3072), що строго відповідає вихідному формату моделі gemini-embedding-2, та вибір метрики косинусної подібності (metric="cosine"). Фрагмент коду, що відповідає за розгортання безсерверної архітектури у регіоні us-east-1 та блокування потоку до повної готовності сервера, наведено у лістингу 3.7.

Лістинг 3.7 – Динамічна ініціалізація індексів Pinecone

```
    existing_indexes = [index_info["name"] for index_info in
pc.list_indexes()]
        if SALARY_INDEX_NAME not in existing_indexes:
            print(f"Створюю новий індекс '{SALARY_INDEX_NAME}'
(Dimensions: {DIMENSION})...")
            pc.create_index(
                name=SALARY_INDEX_NAME,
                dimension=DIMENSION,
                metric="cosine",
                spec=ServerlessSpec(cloud="aws", region="us-
east-1")
            )
            print("Очікування ініціалізації індексу...")
            while not
pc.describe_index(SALARY_INDEX_NAME).status["ready"]:
                time.sleep(2)
```

Після ініціалізації хмарної інфраструктури починається етап формування семантичного контексту. Оскільки модель ембедінгів потребує цілісного тексту для коректного позиціонування об’єкта, у скрипті init_rag.py дані зчитуються з локальної бази SQLite та розбиваються на пакети (батчі). Для кожного запису генерується не лише єдиний текстовий рядок за допомогою функції build_text(), а й розширений словник metas. Збереження вихідних параметрів, таких як назва університету, місто та код спеціальності, безпосередньо у векторній базі як корисного навантаження (метаданих) дозволяє під час семантичного пошуку миттєво повертати готову інформацію користувачеві без повторного звернення до реляційної СУБД. Процес пакетної підготовки об’єктів проілюстровано у лістингу 3.8.

Лістинг 3.8 – Пакетна обробка та формування об’єктів метаданих

```
    for i in range(0, total, BATCH_SIZE):
            batch = programs[i:i + BATCH_SIZE]
            texts = [build_text(r) for r in batch]
            ids = [str(r[0]) for r in batch]
```

Продовження лістингу 3.8

```
        metas = [
            {
                "university": r[1],
                "specialty_code": r[2],
                "specialty_name": r[3],
                "degree": r[4],
                "city": r[5],
                "url": r[6],
                "text_content": build_text(r)
            }
            for r in batch
        ]
```

Головною технічною проблемою під час масової векторизації сформованих пакетів є жорсткі ліміти на частоту запитів з боку API Google Gemini. Для подолання цього обмеження та забезпечення високої відмовостійкості системи інтегровано механізм повторних спроб із застосуванням алгоритму експоненціального відкату, логіку якого наведено у лістингу 3.9.

Лістинг 3.9 – Трансформація тексту у вектори з інтеграцією алгоритму експоненціального відкату

```
    for attempt in range(MAX_RETRIES):
                    try:
                        resp =
gemini_client.models.embed_content(
                            model="models/gemini-embedding-2",
                            contents=title
                        )
                        vectors_to_upsert.append({
                            "id": safe_vector_id,
                            "values":
resp.embeddings[0].values,
                            "metadata": {"title": title,
"salary": salary}
                        })
                        time.sleep(0.8)
```

Продовження лістингу 3.9

```
                    break
                except Exception as e:
                    if "429" in str(e).lower():
                        time.sleep(45)
                    else:
                        time.sleep(2 ** attempt)
```

Аналіз коду з лістингу 3.9 показує, що блок відправки даних до мовної моделі загорнуто у цикл із максимальним лімітом MAX_RETRIES. Якщо віддалений сервер повертає HTTP-помилку перевантаження (код 439), виконання скрипту призупиняється на фіксовані 45 секунд, що дозволяє оновити вікну квот API. При виникненні інших системних збоїв затримка розраховується динамічно як математичний ступінь поточної спроби (2 ** attempt). У разі успішного отримання відповіді сформований масив багатовимірних векторів завантажується у хмару через метод index.upsert(). Створення та наповнення цих індексів дозволило повністю завершити етап офлайн-підготовки та перевести систему в операційний режим.

Однак, саме по собі векторне сховище є лише інфраструктурним фундаментом. Щоб система могла ефективно використовувати ці масиви даних для взаємодії з абітурієнтом у режимі реального часу, необхідно було розробити програмну логіку семантичного пошуку та підключити генеративну мовну модель, що і стало основою операційного етапу.

3.3.2 Семантичний пошук генерація персоналізованих рекомендацій

Після успішного заповнення векторних індексів даними про навчальні заклади та заробітні плати на офлайн-етапі, наступним кроком стала розробка онлайн-компонента системи. Головним завданням цього етапу стала реалізація алгоритму, здатного аналізувати відповіді абітурієнта та генерувати точні персоналізовані рекомендації.

Використання класичних алгоритмів на основі жорсткого дерева рішень є неефективним для сучасної профорієнтації, оскільки інтереси підлітків часто перетинаються на межі різних сфер. З огляду на це, для обробки відповідей було інтегровано генеративний штучний інтелект, модель gemini-2.5-flash. Вона здатна гнучко оцінювати сукупність усіх відповідей, знаходити приховані закономірності та забезпечувати високий рівень адаптивності діалогу.

Програмну взаємодію з мовною моделлю реалізовано у модулі ai_logic.py за допомогою методів інженерної підказок (Promt Engineering). Цей процес логічно розділено на два етапи. На першому етапі функція get_ai_questions збирає 10 відповідей із базового тесту та формує контекст для генерації трьох фінальних уточнюючих запитанньо. Для забезпечення коректного відображення цих запитань, до системного промту додано суворі обмеження: ШІ повинен діяти як професійний кар'єрний консультант, запитувати про конкретні дії або вибір між двома суміжними сферами, уникати вступних фраз, а кожне нове запитання повинно починатися з нового рядка. Це дозволяє системі легко розділити отриманий текст за допомогою синтаксичного аналізу. На другому етапі, після отримання відповідей на уточнюючі запитання, викликається функція get_final_recommendation, яка формує кінцеву професійну рекомендацію.

У лістингу 3.10 наведено фрагмент програмної реалізації функції генерації фінальної професії, де продемонстровано структурування системного запиту, цільове форматування виводу та обробку можливих винятків при взаємодії з API.

Лістинг 3.10 – Функція генерації фінальної професії (фрагмент ai_logic.py)

```
    async def get_final_recommendation(user_data: dict):
        full_context = f"""
        Базовий профіль: {user_data.get('answer1')},
{user_data.get('answer2')}, {user_data.get('answer10')}
```

Продовження лістингу 3.10

```
        Уточнення: {user_data.get('ai_answer1')},
{user_data.get('ai_answer2')}, {user_data.get('ai_answer3')}
        """
        prompt = f"""
        Ти — кар'єрний коуч. На основі відповідей абітурієнта:
        {full_context}
        Сформулюй вердикт у форматі:
        🎯 **Твоя ідеальна професія:** [Назва]
        □ **Чому це тобі підходить:** [3-4 речення]
        Пиши українською мовою.
        """
        try:
            response = client.models.generate_content(
                model=MODEL,
                contents=prompt
            )
            return response.text
        except Exception as e:
            return f"Сервери ШІ тимчасово перевантажені
(Помилка: {e})."
```

Запропонований алгоритм агрегує ключові точки базового профілю користувача (відповіді 1, 2 та 10) разом із результатами трьох етапів інтерактивного ШІ-уточнення. Інструкція спонукає модель виступати в ролі кар’єрного коуча та повертати структурований вердикт із чітким виділенням назви професії та детального обґрунтування її вибору обсягом у 3–4 речення.

Для практичного підтвердження адекватності роботи мовної моделі та доведення логічної цілісності алгоритму генерації було проведено вибірку реальних сесій із локальної бази даних користувачів. Метою цього аналізу була перевірка здатності ШІ-моделі диференціювати абітурієнтів за їхніми схильностями та не допускати видачі однотипних або нерелевантних результатів. У таблиці 3.1 наведено результат обробки відповідей для трьох кардинально різних профілів користувачів: з нахилом до інформаційних технологій, творчих індустрій та управлінського-соціальної сфери.

Таблиця 3.1 – Підтвердження адекватності генерації професій на основі реальних даних користувачів

Сфера	Відповіді статичного тесту	Згенерований вердикт ШІ, після 3 уточнюючих запитань
1	2	3
Інформаційні технології та аналітика	Вивчати нові програми чи грати в ігри; Математика або інформатика; Аналізую факти та шукаю логічне рішення; Тихий офіс або віддалена робота за комп'ютером; Високий дохід та стабілність;	Аналітичний інженер або інженер з обробки даних
	Виконавець, який робить все якісно; Люблю розбиратися, як працюють технології із всередині; Картинка, текст або завершений арт-проект; На потужний комп'ютер або технічне обладнення; Можливість постійно вчитись та розвиватись.	
Творчість та мистецтво	Малювати, писати текст або щось створювати; Фізкультура або праця; Намагаюсь вигадати щось нестандартне; Творча студія або сцена; Високий дохід та стабільність; Лідер, який бере відповідальність на себе; Користуватись для ним для творчості та розваг; Картинка, текст або завершений арт-проект; На навчання мистецтво або подорожі; Свобода самовираження та гнучкий графік.	Незалежний митець/Художник
Бізнес та управління	Спілкування з друзями та допомагати іншим; Фізкультура або праця; Раджусь з кимось та шукаю підтримки; Творча студія або сцена; Високий дохід та стабільність; Лідер, який бере відповідальність на себе; Користуватись для ним для творчості та розваг; Виконаний план або зароблений прибуток; На запуск власної справи або стартапу; Свобода самовираження та гнучкий графік.	Бізнес коуч/Стратегічний консультант

Як видно з наведеної таблиці, система генерує не випадкові результати, а чітко простежує семантичний зв'язок між первинними відповідями користувача та кінцевим вердиктом. Штучний інтелект коректно розрізняє технічний, творчий та управлінський напрямки,

генеруючи вузькопрофільні й логічно обґрунтовані професії під кожен унікальний запит. Це повністю доводить, що застосована архітектура та налаштування моделі Gemini успішно запобігають генеруванню неадекватного або хибного контенту, забезпечуючи високу точність профорієнтації.

Паралельно з цим, реалізовано блок перехоплення винятків except Exception as e забезпечує стійкість системи: у разі виникнення мережевих затримок або перевищення лімітів до API користувач отримує інформативне повідомлення замість критичного аварійного завершення роботи бота.

Пряме опитування великих мовних моделей щодо специфічних даних є ризикованим через їхню схильність до «галюцинацій», що може призвести до генерації неіснуючих освітніх програм або надання застарілих відомостей про заробітну плату. Для розв’язання цієї проблеми розроблено архітектуру RAG (Retrieval-Augmented Generetion). У такий парадигмі ШІ відповідає виключно за логічний висновок та текстове обґрунтування вибору професії, а достовірні дані про університети та фінансові показники динамічно вилучаються з локальних векторних індексів Pinecone.

Семантичний пошук забезпечується модулем vector_search.py. Назва обраної професії перетворюється на числовий вектор за допомогою моделі models/gemini-embedding-2. Функція search_relevant_programs здійснює пошук векторних збігів в індексі universities-v2 та повертає список об’єктів. Логіку формування векторного запиту та вилучення метаданих ЗВО наведено у лістингу 3.11.

Лістинг 3.11 – Функція семантичного пошуку освітніх програм (фрагмент vector_search.py)

```
    def search_relevant_programs(profession_name: str, top_k:
int = 5) -> list[dict]:
        try:
            response = gemini_client.models.embed_content(
                model="models/gemini-embedding-2",
                contents=profession_name
```

Продовження лістингу 3.11

```
            )
            query_vector = response.embeddings[0].values
            search_results = univ_index.query(
                vector=query_vector,
                top_k=top_k,
                include_metadata=True
            )
            matches = []
            for item in search_results.get("matches", []):
                meta = item["metadata"]
                matches.append({
                    "university": meta.get("university",
"Невідомий ЗВО"),
                    "specialty_code":
meta.get("specialty_code", ""),
                    "specialty_name":
meta.get("specialty_name", ""),
                    "degree": meta.get("degree", "Бакалавр"),
                    "city": meta.get("city", ""),
                    "url": meta.get("url",
"https://vstup.edbo.gov.ua/")
                })
            return matches
        except Exception as e:
            print(f"Помилка пошуку університетів у Pinecone:
{e}")
            return []
```

Як показує наведена реалізація, алгоритм трансформує текстовий вердикт у багатовимірне числове подання розмірності 3072, після чого виконує математичне зіставлення за метрикою консисної подібності з векторами у просторі бази даних навчальних закладів. З отриманого результату функція динамічно розпаковує словник метаданих, імпортуючи верифіковану інформацію про ліцензовані програми ЗВО (університету, коду і назву спеціальності, ступінь, місто та посилання на ЄДБО).

Важливою архітектурною особливістю взаємодії цього модуля з інтерфейсною частиною бота є гнучке керування параметром top_k. Хоча за

замовчуванням функція налаштована на вилучення п’яти найближчих збігів, під час інтеграції з основним циклом обробки у main.py цей параметр може динамічно перевищуватись або модифікуватись залежно від стану сесії користувача та логіки формування каруселі пагінації. Це дозволяє гнучко розширювати масив релевантних освітніх програм, які зберігаються у тимчасовому контексті машини станів бота, забезпечуючи користувача більшою вибіркою варіантів для перегляду без потреби у повторному зверненні до хмарного векторного сховища.

Паралельно з підбором університетів, у моделі реалізовано функцію get_live_salary_from_workua, яка здійснює семантичний запит до індексу salaries-index для отримання актуальної фінансової аналітики. Програмну логіку фільтрації та розрахунку економічних показників наведено у лістингу 3.12.

Лістинг 3.12 – Функція семантичного пошуку фінансових показників (фрагмент vector_search.py)

```
    def get_live_salary_from_workua(profession_name: str) ->
str:
        clean_query = profession_name.split("—")[-1].strip()
if "—" in profession_name else profession_name.strip()
        try:
            response = gemini_client.models.embed_content(
                model="models/gemini-embedding-2",
                contents=clean_query
            )
            query_vector = response.embeddings[0].values
            results = salary_index.query(
                vector=query_vector,
                top_k=1,
                include_metadata=True
            )
            if results.get("matches"):
                match = results["matches"][0]
                if match["score"] > 0.65:
                    job_title = match["metadata"].get("title",
clean_query)
```

Продовження лістингу 3.12

```
                    salary = match["metadata"].get("salary",
"Уточнюється")
                    return f"💼 <b>{job_title}</b>\n💰 Середня
зарплата: <b>{salary}</b>"
            return f"💼 <b>{clean_query}</b>\n💰 Орієнтовна
ринкова зарплата: <b>~ 25 000 - 35 000 грн</b>"
        except Exception as e:
            print(f"Помилка пошуку зарплати у Pinecone: {e}")
            return f"💼 <b>{clean_query}</b>\n💰 Дані про
зарплату тимчасово недоступні."
```

Ключовим архітектурним рішенням у цьому алгоритмі є впровадження жорсткого валідаційного фільтра за математичним порогом косинусної подібності match["score"] > 0.65. Перед надсиланням запиту виконується нормалізація рядка для вилучення чистих пошукових маркерів. Отриманий вектор порівнюється з векторами фінансових профілів ринку праці. Якщо максимальний показник схожості з базою даних Work.ua перевищує 65% система імпортує точну назву посади та верифікований рівень доходу з метаданих. Проте, у випадку занадто вузьких або унікальних професій, подібність може бути нижчою за встановлену межу. Замість виведення випадкових або помилкових збігів, алгоритм безпечно активує запасний сценарій, повертаючи усереднений орієнтовний ринковий діапазон від 25 000 до 35 000 грн. Конструкція try-except додатково захищає інтерфейс від критичних збоїв у разі мережевих затримок, інформуючи користувача про тимчасову недоступність про фінансові сервіси.

Таким чином, розроблений конвеєр об’єднує аналітичні можливості генеративних мереж із надійністю векторних баз даних, забезпечуючи високу точність та фактологічну достовірність кінцевої профорієнтаційної рекомендації.

Проте весь цей потужний інтелектуальний функціонал залишається недоступним для кінцевого користувача без наявності зручного та інтуїтивно зрозумілого каналу комунікації. Тому логічним продовженням

розробки стало створення клієнтської оболонки, яка б дозволила абітурієнту комфортно взаємодіяти з системою, проходити тестування та отримувати згенеровані результати.

3.4 Програмна реалізація діалогового інтерфейсу Telegram-бота та підсистеми управління сесіями

Після успішної реалізації інтелектуального ядра та векторної бази знань ключовим інженерним завданням стала організація зручної та безперервної взаємодії з абітурієнтом. Роль клієнтського інтерфейсу в розробленій системі виконує месенджер Telegram. Таке рішення забезпечує користувачам миттєвий доступ до сервісу без необхідності встановлення додаткового програмного забезпечення, що кардинально знижує поріг входу та психологічне навантаження.

Розробка головного програмного модуля маршрутизації (файл main.py) здійснювалася за допомогою фреймворка aiogram. Застосування парадигм асинхронного програмування дозволяє серверній частині ефективно обробляти запити від багатьох користувачів одночасно. Завдяки цьому основний потік програми не блокується під час очікування відповідей від віддалених API-сервісів штучного інтелекту або запитів до бази даних, що гарантує високу пропускну здатність чат-бота.

3.4.1 Організація локального збереження даних та дедуплікація користувацьких сесій

Ефективне функціонування профорієнтаційної системи в реальних умовах експлуатації вимагає розробки надійного механізму довгострокового зберігання інформації та однозначної ідентифікація користувача. Використання виключно тимчасового кешу в оперативній пам’яті сервера є незадовільним архітектурним рішенням, оскільки воно не

забезпечує збереження прогресу при запуску програмних модулів, не дозволяє накопичувати статичні дані та унеможливлює повторний аналіз профілів користувача.

Для вирішення цього завдання було спроектовано локальне сховище на базі реляційної СУБД SQLite. Вибір цієї системи зумовлений її вбудованим характером, високою швидкістю виконання транзакцій та відсутністю потреби в адмініструванні окремого серверного процесу, що суттєво мінімізує накладні витрати обчислювальних ресурсів. Архітектура таблиці users розроблена з урахуванням необхідності фіксації унікальних ідентифікаторів клієнтів, їхніх системних імен (нікнейм) у месенджері, текстових відповідей на кожен із десяти базових кроків тестування, а також фінального вердикта профорієнтаційного коуча.

Ініціалізація структури бази даних реалізовано у модулі database/db.py. Програмний код створення таблиці представлено у лістингу 3.13.

Лістинг 3.13 – Ініціалізація структури реляційної бази даних (фрагмент db.py)

```
def db_start():
    conn = sqlite3.connect('database/vocation.db')
    cur = conn.cursor()

    cur.execute('''
        CREATE TABLE IF NOT EXISTS users (
            user_id INTEGER PRIMARY KEY,
            username TEXT,
            q1 TEXT, q2 TEXT, q3 TEXT, q4 TEXT, q5 TEXT,
            q6 TEXT, q7 TEXT, q8 TEXT, q9 TEXT, q10 TEXT,
            final_profession TEXT
        )
    ''')
    conn.commit()
    conn.close()
```

Використання конструкції CREATE TABLE IF NOT EXISTS є важливим елементом надійності: під час запуску чат-бота система автоматично перевіряє наявність цільового файлу сховища й ініціалізує таблицю лише у разі первинного розгортання проекту. Поле user_id визначено як первинний ключ (PRIMARY KEY), що гарантує унікальність кожного запису, оскільки ідентифікатори користувачів платформи Telegram є статичним та унікальним за своєю природою.

Критично важливим аспектом підсистеми управління сесіями є дедуплікація даних. При повторному зверненні абітурієнта до сервісу або у разі надсилання команди /start посеред проходження тесту, система не повинна створювати дублюючі записи, адже це призведе до штучного розширення файлу vocation.db та порушення цілісності аналітики. Для усунення цієї проблеми розроблено алгоритм безпечної реєстрації, програмну реалізацію якого наведено у лістингу 3.14.

Лістинг 3.14 – Алгоритм реєстрації користувача та дедуплікація сесій (фрагмент db.py)

```
    def add_user(user_id, username):
        conn = sqlite3.connect('database/vocation.db')
        cur = conn.cursor()
        cur.execute("INSERT OR IGNORE INTO users (user_id,
username) VALUES (?, ?)", (user_id, username))
        conn.commit()
        conn.close()
```

Застосування специфікатора INSERT OT IGNORE дозволяє перенести логіку уникнення дублікатів безпосередньо на рівень СУБД. Якщо користувач із поточним user_id уже занесений до таблиці users, операція запису буде безпечно проігнорована без генерації програмних винятків (Exception), що суттєво підвищує стабільність роботи асинхронного потоку обробників чат-бота.

Взаємодія підсистеми збереження даних із головним циклом маршрутизації повідомлень (файл main.py) відбувається в момент активації діалогового інтерфейсу. Програмну реалізацію обробки стартових інструкцій розробника та ініціалізацію тестування наведено у лістингу 3.15.

Лістинг 3.15 – Обробник команд запуску бота та активації тестування (фрагмент main.py)

```
    @dp.message(Command("start"))
    async def cmd_start(message: types.Message):
        add_user(message.from_user.id,
message.from_user.username)
        await message.answer(
            "👋 Привіт! Я — твій персональний ШІ-асистент з
вибору професії.\n\n"
            "Натискай /survey, щоб розпочати тест!"
        )
    @dp.message(Command("survey"))
    async def start_survey(message: types.Message, state:
FSMContext):
        short_q1 = [
            "💻 Програми та ігри",
            "🎨 Творчість та дизайн",
            "🤝 Спілкування з людьми",
            "📈 Організація справ"
        ]
        await message.answer(
            "1. Чим тобі найбільше подобається займатися у
вільний час?",
            reply_markup=make_inline_kb(short_q1)
        )
        await state.set_state(VocationForm.q1)
```

Послідовність дій у розроблених обробниках спроектована за принципом чіткого розділення функціональних обов’язків. Метод cmd_start реагує на первинну системну команду /start, виконуючи звернення до функції add_user для транзакції реєстрації та дедуплікації у СУБД, після чого надсилає інформаційне привітання з інструкцією щодо подальших дій.

Безпосередній запуск профорієнтаційного скрипта делеговано окремому обробнику start_survey, який активується текстовою командою /survey. Такий підхід дозволяє користувачеві свідомо розпочати інтерактивне опитуванняю Функція генерує перше запитання, за допомогою утілити make_inline_kb створює блок кнопок із варіантами відповідей та ініціалізує первинну фіксацію діалогового контексту. Саме цей крок забезпечує передачу керування підсистеми покрокового контролю, детальний аналіз та принципи функціонування якої розглянуто нижче.

3.4.2 Керування станами користувача за допомогою Finite Stare Machine (FSM) та резервне збереження даних

Процес профорієнтаційного тестування складається з тривалого діалогового ланцюжка: 10 фіксованих запитань для формування базового профілю та 3 динамічних запитань, згенерованих штучним інтелектом. Для того, щоб програма могла коректно розпізнавати, на якому саме етапі опитування знаходиться конкретний користувач, і не плутала контексти різних сесій, було впроваджено патерн «Машина станів» (Finite Stare Machine). Застосування моделі скінченного автомата гарантує передбачуваність поведінки системи та забезпечує чіткий контроль переходів між етапами діалогової взаємодії.

В екосистемі aiogram цей механізм реалізується через успадкування від класу StateGroup, де кожен крок користувача описується як ізольований об’єкт State. Програмну реалізацію моделі станів для розробленого тесту наведено у лістингу 3.16.

Лістинг 3.16 – Оголошення класів станів для управління діалоговою сесією (фрагмент main.py)

```
from aiogram.fsm.state import State, StatesGroup
class VocationForm(StatesGroup):
    q1 = State()
```

Продовження лістингу 3.16

```
q2 = State()
q3 = State()
q4 = State()
q5 = State()
q6 = State()
q7 = State()
q8 = State()
q9 = State()
q10 = State()
ai_q1 = State()
ai_q2 = State()
ai_q3 = State()
```

Наведена структура описує повний життєвий цикл сеансу профорієнтації. Логіка переходу між станами побудована за принципом суворої маршрутизації: кожен обробник подій реагує на повідомлення або натискання кнопки виключно тоді, коли користувач перебуває у відповідному для цього стані.

Важливим архітектурним викликом під час проектування машини станів стала проблема збереження проміжних результатів. За замовчуванням механізм FSMContext зберігає дані в оперативній пам’яті сервера. Це забезпечує максимальну швидкість доступу до даних під час генерації фінального запиту до ШІ. Однак, у разі перезавантаження сервера, оновлення коду бота або непередбачуваного збою, вся оперативна пам’ять очищується. Це призвело б до втрати прогресу всіх користувачів, які не встигли завершити тестування.

Для усунення цього ризику розроблено гібридну систему управління даними. Паралельно із збереженням в оперативну пам’ять, кожна відповідь користувача синхронно дублюється у локальну реляційну базу даних SQLite. Реалізація функції резервного запису в базу наведено у лістинг 3.17.

Функція update_user_data динамічно приймає ідентифікатор користувачі (user_id), назву колонки, що відповідає поточному запитанню (column), та обраний варіант відповіді (value). Такий підхід

робить функцію універсальною для всіх кроків опитування і гарантує збереження цілісності сесії.

Лістинг 3.17 – Модуль з локальною базою даних SQLite (фрагмент db.py)

```
    def update_user_data(user_id, column, value):
        conn = sqlite3.connect('database/vocation.db')
        cur = conn.cursor()
        cur.execute(f"UPDATE users SET {column} = ? WHERE
user_id = ?", (value, user_id))
        conn.commit()
        conn.close()
```

Практичне застосування гібридного методу збереження інформації та переходу між станами продемонстровано на прикладі обробника запитання у лістингу 3.18.

Лістинг 3.18 – Гібридна обробка відповіді та зміна стану сесії (фрагмент main.py)

```
    @dp.callback_query(VocationForm.q1)
    async def process_q1(callback: types.CallbackQuery, state:
FSMContext):
        answers_q1 = [
            "💻 Вивчати нові програми чи грати в ігри",
            "🎨 Малювати, писати тексти або щось створювати",
            "🤝 Спілкуватися з друзями та допомагати іншим",
            "📈 Планувати справи або щось організовувати"
        ]
        selected_answer = answers_q1[int(callback.data)]
        await state.update_data(answer1=selected_answer)
        update_user_data(callback.from_user.id, "q1",
selected_answer)
        await callback.answer()
        await
callback.message.edit_reply_markup(reply_markup=None)
        answers_q2 = [
            "🔢 Математика або інформатика",
```

Продовження лістингу 3.18

```
            "📚 Література, мови або історія",
            "🧬 Біологія, хімія або фізика",
            "⚽ Фізкультура або праця"
        ]
        await callback.message.answer(
            "2. Який шкільний предмет був (або є) твоїм
улюбленим?",
            reply_markup=make_inline_kb(answers_q2)
        )
        await state.set_state(VocationForm.q2)
```

Як видно з наведеного фрагмента, система перехоплює подію CallbackQuery, яка генерується при натисканні інтерактивної кнопки (Inline-кнопки). Відповідь спочатку кешується в оперативній пам’яті машини станів (state.update_data), після чого миттєво викликається update_user_data для запису на фізичний носій. Тільки після успішного збереження даних поточний інтерфейс оновлюється (попередні кнопки приховуються для запобігання повторних натискань), а метод state.set_state() переводить сесію на етап q2.

Особливою архітектурною деталлю є безшовний перехід від структурованої частини опитування до динамічної. Коли абітурієнт завершує базовий блок (стан q10), накопичений в оперативній пам’яті словник відповідей відправляється до модуля штучного інтелекту. Отримавши від API мовної моделі три індивідуальні запитання, програма зберігає їх у пам’яті FSM у вигляді масиву та перемикає діалог у стан ai_q1. Такий алгоритм дозволяє системі органічно адаптуватися під кожну особу, зберігаючи при цьому суворий програмний контроль над послідовністю проведення консультації.

Проте, надійна серверна логіка та строгий контроль станів є ефективним лише за умови зручної і безпомилкової взаємодії з боку клієнта. Щоб мінімізувати ймовірність ведення некоректних даних, які можуть порушити роботу алгоритму, та забезпечити комфортне подання

згенерованих результатів, систему було доповнено спеціалізованими візуальними компонентами.

3.4.3 Розробка динамічних елементів інтерфейсу та механізму каруселної навігації

Важливим критерієм якості сучасних програмних засобів автоматизації є рівень зручності інтерфейсу користувачів (UX/UI). У контексті Telegram-ботів введення даних у вигляді вільного тексту підвищує ризик виникнення помилок та ускладнює подальшу аналітику. Для усунення цієї проблеми в розробленій системі було реалізовано механізм динамічного формування інтерактивних Inline-клавіатури, які адаптуються під поточний контент на кожному кроці діалогу.

Процес побудови матриці кнопок варіантів відповідей для базового тестування автоматизовано за допомогою універсальної функції make_inline_kb. Програмну реалізацію цього методу наведено у лістингу 3.19.

Лістинг 3.19 – Автоматизована генерація Inline-клавіатури (фрагмент main.py)

```
    from aiogram.types import Message, CallbackQuery,
InlineKeyboardMarkup, InlineKeyboardButton
    def make_inline_kb(answers: list[str]) ->
InlineKeyboardMarkup:
        keyboard = []
        for i, answer in enumerate(answers):
            keyboard.append([InlineKeyboardButton(text=answer,
callback_data=str(i))])
        return InlineKeyboardMarkup(inline_keyboard=keyboard)
```

Алгоритм приймає динамічний список текстових рядків, ітерує його та для кожного елемента створює об’єкт InlineKeyboardButton. Як ідентифікатор події (callback_data) використовується індекс елемента у

масиві, перетворений у рядок. Такий підхід мінімізує обсяг метаданих, що передається всередині серверів месенджера, та уніфікує подальшу обробку натискань через єдиний тип даних.

Особливим архітектурним викликом стала презентація результатів семантичного пошуку ЗВО. Функція пошуку повертає масив із декількох знайдених освітніх програм, де кожна одиниця даних містить неформатовані технічні рядки. Для забезпечення презентабельності та естетичного відображення інформації було розроблено модуль форматування та синтаксичного очищення тексту (format_program_text), наведений у лістингу 3.20.

Лістинг 3.20 – Функція форматування та текстове очищення даних ЗВО (фрагмент main.py)

```
import re
def format_program_text(program: dict) -> str:
    code = program['specialty_code'].strip()
    raw_name = program['specialty_name'].strip()
    cleaned_name = re.sub(r'([a-яієїґa-z])([A-ЯІЄЇҐA-Z])', 
r'\1 — \2', raw_name)
    clean_specialty = f"{code} {cleaned_name}"
    return (
        f"🎓 **Університет:** {program['university']}\n\n"
        f"📍 **Місто:** {program['city']}\n\n"
        f"💻 **Спеціальність:** {clean_specialty}\n\n"
        f"📜 **Ступінь:** {program['degree']}\n\n"
        f"💡 *Оберіть цей варіант, щоб переглянути деталі 
на сайті ЄДЕБО або дізнатися про перспективи.*"
    )
```

У наведеному алгоритмі ключовим рішенням є використання інструментів регулярних виразів re.sub. Оскільки первинні метадані з векторного індексу Pinecone можуть містити дефекти злиття слів (наприклад, написання назви спеціальності без пропусків на межі регістрів), шаблон r'([а-яієїґa-z])([А-ЯІЄЇҐA-Z])', r'\1 – \2' динамічно виявляє

стики між малими та великими літерами і автоматично розмежовує їх символом тире. Результат повертається у вигляді структурованого Markdown-повідомлення.

Виведення всіх знайдених університетів одним довгим повідомленням перевантажує абітурієнта. Для оптимізації сприйняття було впроваджено механізм «каруселі» з посторінковими гортанням карток. Для безпечної передачі індексу поточної сторінки між асинхронними обробниками подій натискання кнопок було розроблено спеціалізовану фабрику за допомогою CallbackData. Схему оголошення класу та логіку побудови інтерфейсу каруселі наведено в лістингу 3.21.

Лістинг 3.21 – Конструювання інтерфейсу циклічної пагінації (фрагмент main.py)

```
    from aiogram.filters.callback_data import CallbackData
    from aiogram.utils.keyboard import InlineKeyboardBuilder
    def get_carousel_keyboard(current_page: int, total_pages:
int, url: str) -> InlineKeyboardMarkup:
        builder = InlineKeyboardBuilder()
        if url and url.startswith("http"):
            builder.row(InlineKeyboardButton(text="🔗
Детальніше на ЄДЕБО", url=url))
        builder.row(InlineKeyboardButton(text="💰 Дізнатися
рівень зарплати", callback_data="get_salary"))
        if total_pages > 1:
            prev_page = (current_page - 1) if current_page > 0
else (total_pages - 1)
            next_page = (current_page + 1) if current_page <
(total_pages - 1) else 0
            builder.row(
                InlineKeyboardButton(text="⬅ Назад",
callback_data=CarouselCallback(page=prev_page).pack()),
                InlineKeyboardButton(text=f"📄 {current_page +
1} з {total_pages}", callback_data="noop"),
                InlineKeyboardButton(text="➡ Вперед",
callback_data=CarouselCallback(page=next_page).pack())
            )
        return builder.as_markup()
    class CarouselCallback(CallbackData, prefix="carousel"):
        page: int
```

Проектування класу CarouselCallback з єдиним цілочисловим полем page та унікальним системним префіксом “carousel” дозволило значно скоротити розмір фінального callback-рядка, запобігаючи обмеження платформи Telegram на довжину пакетів даних. Логіка формування клавіатури, реалізована за допомогою InlineKeyboardBuilder, гнучко адаптує структуру інтерфейсу. Перший ряд динамічно перевіряє валідність URL-посилання та виводить пряму кнопку переходу до ЄДБО. Другий ряд резервується під статичний обробник подій “get_salary”, що дозволяє абітурієнту на будь-якому етапі перегляду каруселі ініціювати аналіз ринку праці для обраної професії.

Особливу увагу при розробці було приділено побудові навігаційного блоку. Замість лінійного обмеження, яке блокує кнопки на крайніх позиціях, у системі впроваджено замкнену циклічну пагінацію. Математичний розрахунок індексів prev_page та next_page через тернарні оператори забезпечує безперервне гортання: при натисканні кнопки «Назад» на першій сторінці (current_page == 0) користувач автоматично переміщується на фінальну картку (total_pages – 1). Центральний індикатор (current_page + 1 з total_pages) транслює поточну позицію і захищений від хибних обчислень через перехоплювач «noop».

Завершальним елементом інтерфейсу є асинхронна обробка події натискання на кнопку «Дізнатися рівень зарплати». Відокремлення фінансової аналітики від загального масиву даних університетів дозволило зберегти лаконічність стартової картки. При активації події «get_salary» програма вилучає точну назву професії з контексту машини станів або дублюючої СУБД SQLite. Дані передаються до модуля векторного аналізу get_live_salary_from_workua, а знгенерована відповідь із верифікованими показниками заробітної плати надсилаються користувачеві у вигляді окремого інформаційного повідомлення з HTML-форматування.

Завдяки впровадженню подійного підходу та алгоритму циклічної пагінації розроблений чат-бот гарантує динамічне, структуроване й

відмовостійке виведення великих масивів даних. Це зробило роботу з ботом зрозумілою та ефективною для користувача.

Таким чином, створення продуманої клієнтської оболонки з інтерактивними елементами завершило повну програмну реалізацію профорієнтаційної системи. Процес розробки, описаний у цьому розділі, охопив усі критичні етапи побудови програмного комплексу: від автоматизованого збору фактологічних даних із зовнішніх ресурсів до розгортання когнітивного ядра. Впровадження RAG-архітектури, інтеграція мовної моделі Gemini 2.5 Flash та використання векторного індексу Pinecone забезпечили високу точність рекомендацій. З і свого боку, розроблений Telegram-бот на базі фреймворку aiogram, доповнений гібридною підсистемою управління сесіями та патерном машини станів, гарантував відмовостійку та комфортну взаємодію з абітурієнтом.

Успішне об’єднання всіх розроблених ізольованих модулів – автономних парсерів, бази знань, ШІ-ядра та інтерфейсу – у єдиний функціональний продукт зумовлює необхідність практичної перевірки працездатності. Для підтвердження ефективності обраних рішень, стійкості алгоритмів до нетипової поведінки користувачів та загальної коректності роботи системи потрібно провести її комплексну перевірку. Відповідно, логічним етапом продовження роботи є опис методології тестування розробленого програмного забезпечення та демонстрація сценаріїв його використання в умовах, наближених до реальних.

4 ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ ПРОФОРІЄНТАЦІЙНОЇ СИСТЕМИ

Після успішного завершення етапу програмної реалізації, описаного у третьому розділі, критично важливим кроком є перевірка працездатності створеної профорієнтаційної системи в реальних умовах експлуатації. Цей розділ присвячений комплексній демонстрації роботи Telegram-бота, тестування його інтелектуального ядра на базі RAG-архітектури та аналізу загальної відмовостійкості програмного продукту. Для повноцінної оцінки системи процес тестування поділено на два етапи: перевірка базового сценарію взаємодії (ідеального шляху користувача) та тестування алгоритмів обробки сценарію в умовах нетипової поведінки і мережевих збоїв.

4.1 Демонстрація базового сценарію використання та інтелектуальної генерації рекомендацій

Базовий сценарій демонструє еталонний шлях абітурієнта від моменту першого запуску бота до отримання фінального результату, що включає персоналізовано підібрану професію, список релевантних навчальних закладів та актуальні дані про заробітну плату.

Процес розпочинається з активації клієнтського інтерфейсу командою /start. Програма миттєво реєструє нового користувача в базі даних SQLite, уникаючи дублювання, та виводить вітальне повідомлення з інструкцією. Після введення команди /survey ініціалізується підсистема управління станами та запускається перший етап тестування, який складається з 10 структурованих запитань (рисунок 4.1).

На цьому етапі користувач взаємодіє виключно з динамічними кнопками меню, що значно прискорює проходження опитування та зводить до мінімуму ризик синтаксичних помилок. Кожна обрана відповідь

фіксується в оперативній пам’яті сервера та синхронно дублюється на фізичний носій для збереження прогресу.

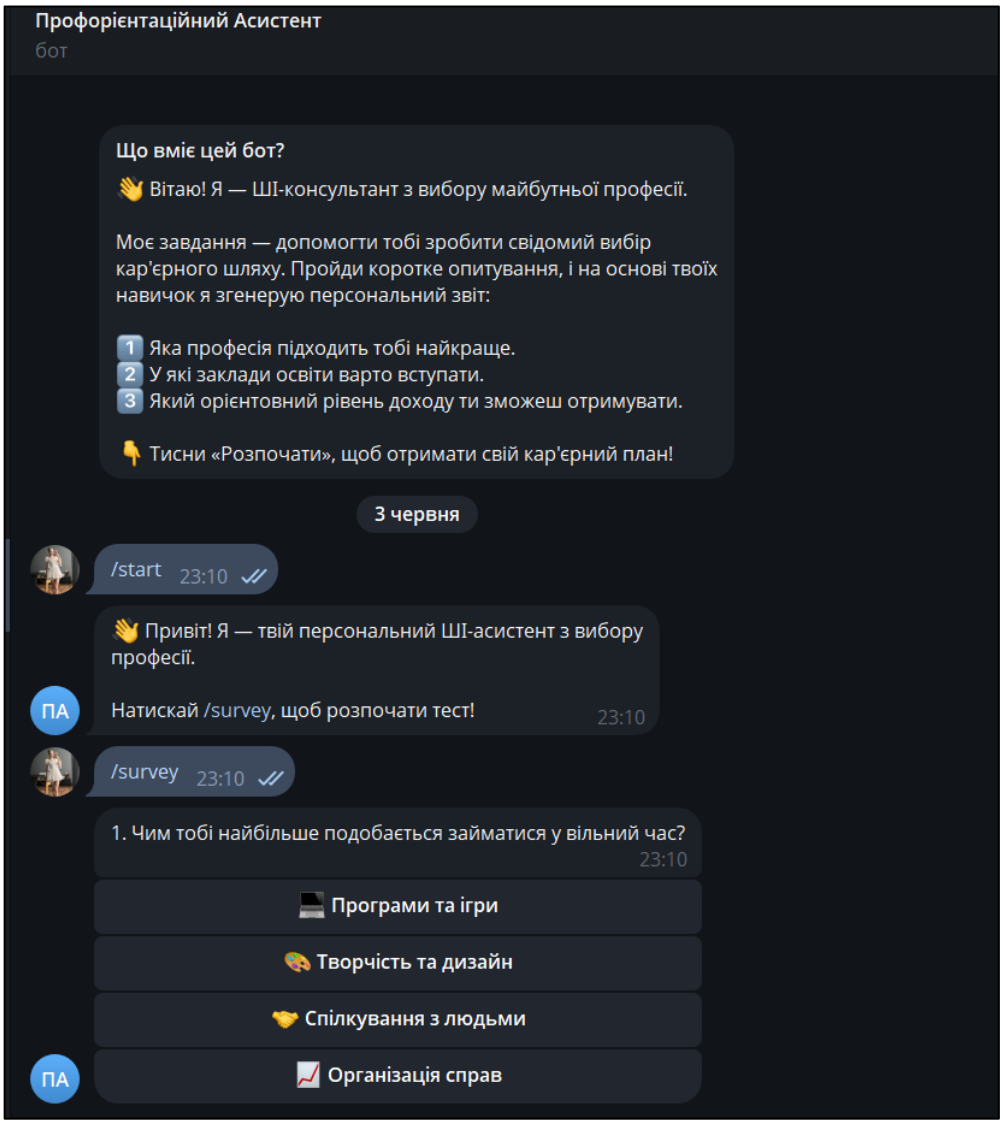


Рисунок 4.1 – Ініціалізація сесії та відображення базових Inline-клавіатур

Після успішного проходження першого блоку (10 базових запитань) система інформує про завершення етапу та переходить до етапу динамічної адаптації. Зібраний первинний профіль надсилається до мовної моделі Gemini, яка генерує три унікальні уточнюючі запитання, орієнтуючись на виявлені схильності абітурієнта (наприклад фокус на балансі між творчою

реалізацією та логічними системами). Процес цього інтерактивного діалогу з ШІ продемонстровано на рисунку 4.2.

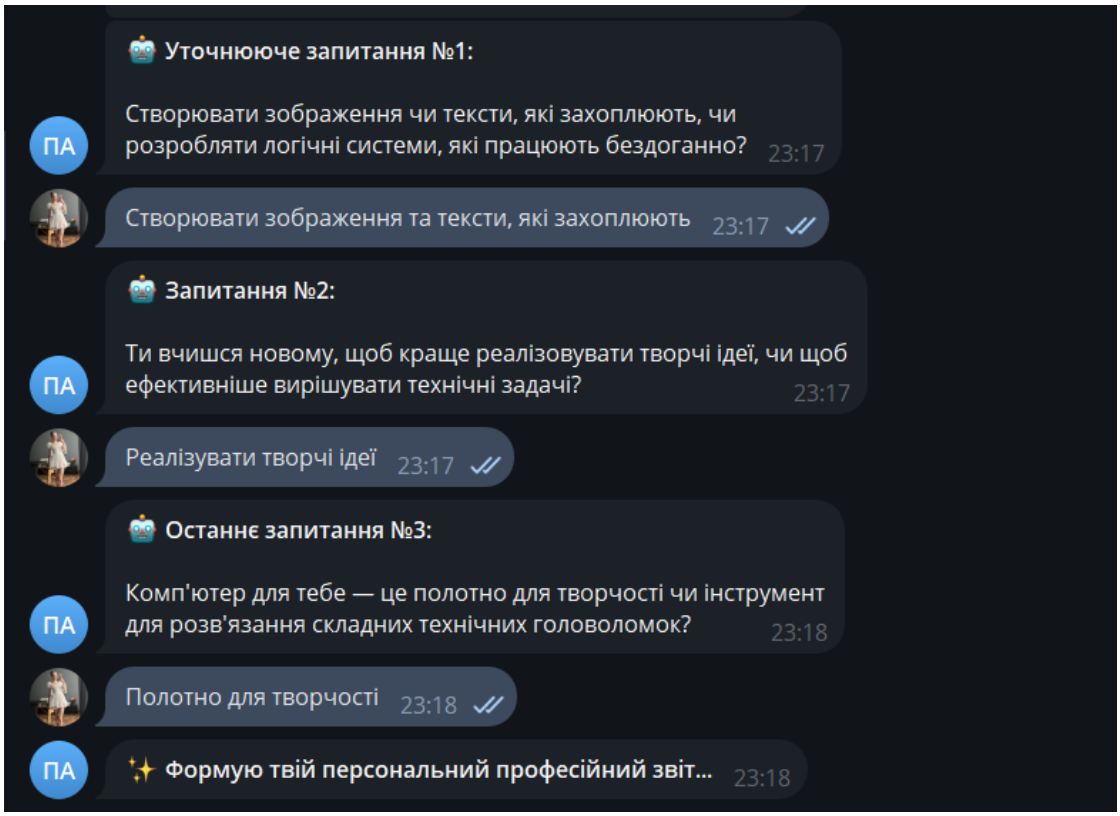


Рисунок 4.2 – Адаптивний етап тестування зі згенерованими ШІ запитаннями

Отримавши відповіді на всі уточнювальні запитання, система переходить до ключового етапу – формування фінального профорієнтаційного вердикту. Як показано на рисунку 4.3, модуль штучного інтелекту комплексно аналізує весь контекст попереднього діалогу, зіставляючи виявлені схильності та психологічні риси абітурієнта з актуальними вимогами ринку праці. У результаті такого семантичного зіставлення генерується чітка персоналізована рекомендація (у наведеному прикладі – «UX/UI Дизайнер»). Важливою особливістю архітектури є те, що система не просто видає суху назву спеціальності, а генерує розгорнуте логічне обґрунтування, детально пояснюючи користувачу, які саме з його

відповідей вказують на високу сумісність із запропонованим професійним напрямком.

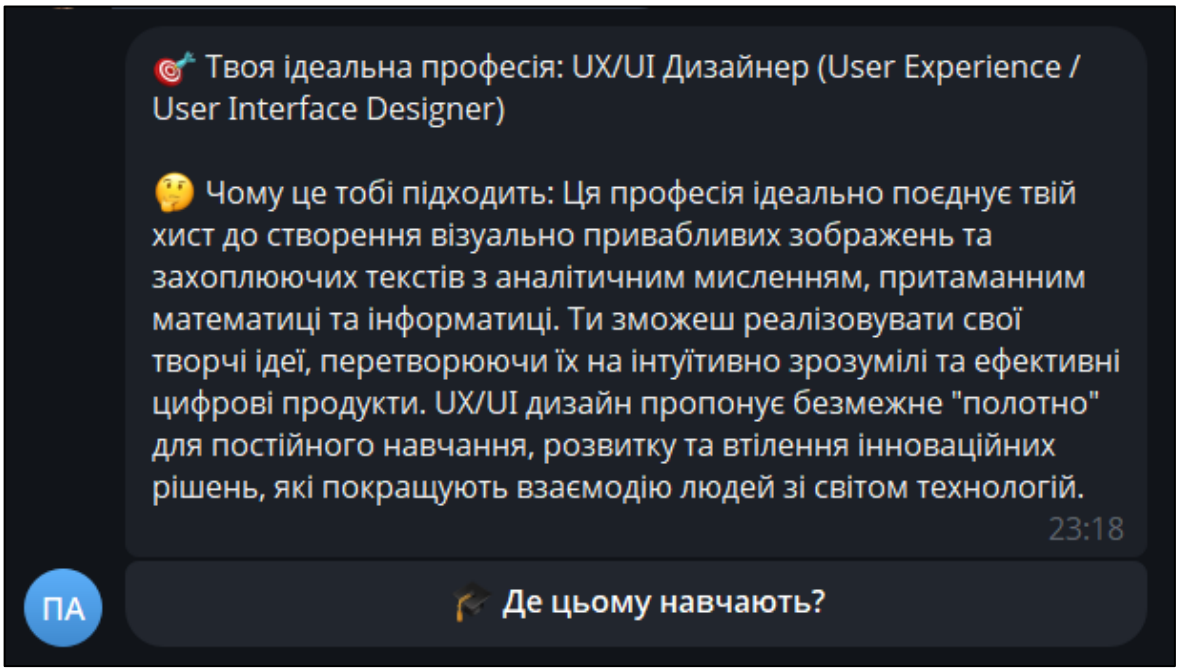


Рисунок 4.3 – Згенерований результат: персоналізована професійна рекомендація

Для подальшого дослідження обраного фаху користувач ініціює пошук навчальних закладів за допомогою інтерактивної кнопки «Де цьому навчають?». У цей момент у фоновому режимі відпрацьовує RAG-підсистема: алгоритм трансформує назву професії у вектор, виконує семантичний пошук у хмарних індексах Pinecone та підтягує верифіковані дані з реєстру ЄДБО. Результат подається у вигляді інтерактивної каруселі з кнопками циклічної пагінації (рисунок 4.4). На наведеному прикладі видно успішно знайдену релевантну освітню програму («022 Дизайн») у Національному університеті «Львівська політехніка», а також доступні дії: перехід за прямим посиланням на сайт ЄДБО або перевірка фінансових перспектив.

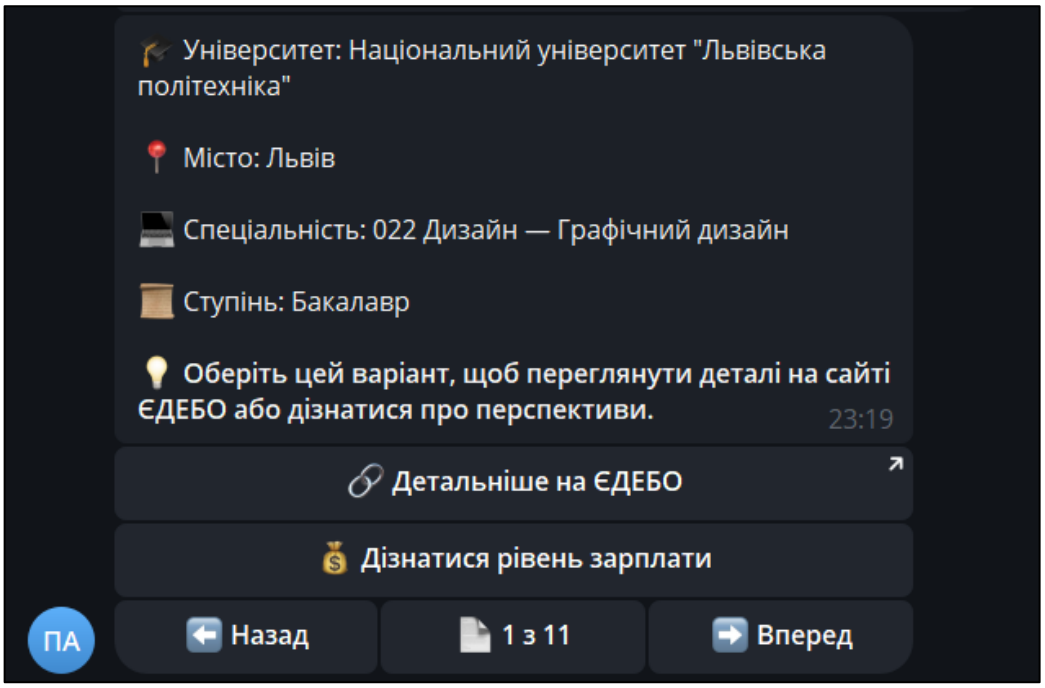


Рисунок 4.4 – Відображення освітніх програм через механізм карусельної навігації

Завершальним етапом консультації є аналіз ринку праці. При натисканні кнопки «Дізнатися рівень зарплати» система виконує векторний запит до індексу фінансової аналітики і повертає актуальну середню заробітну плату для найбільш релевантної посади зі знайдених на порталі Work.ua (наприклад, Web-дизайнер), що ілюструє рисунок 4.5.

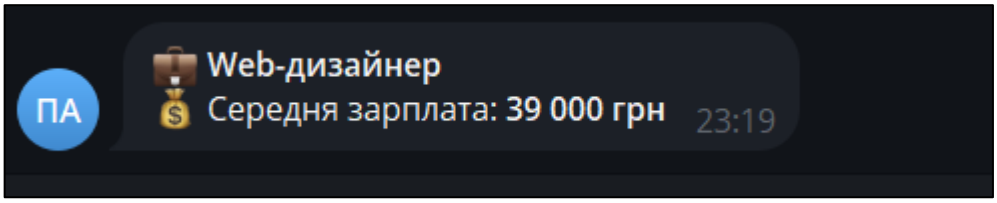


Рисунок 4.5 – Виведення актуальної фінансової аналітики ринку праці

Продемонстрований базовий сценарій підтверджує, що за ідеальних умов усі інтеграційні вузли, бази даних та алгоритми генерації працюють абсолютно коректно, забезпечуючи високу швидкість відгуку та логічну

послідовність етапів консультації. Однак, у реальних умовах експлуатації програмні продукти регулярно стикаються з непередбачуваними діями користувачів або тимчасовими збоями віддалених серверів. Тому для остаточного підтвердження надійності архітектури необхідно проаналізувати поведінку бота в критичних ситуаціях, що детально розглядається у наступному підрозділі.

4.2 Перевірка відмовостійкості системи та обробка виняткових ситуацій

Стабільність роботи програмного забезпечення визначається не лише його здатністю виконувати базові функції в ідеальних умовах, але й умінням коректно реагувати на нетипові дії користувачів та збої в роботі зовнішніх інфраструктур. Для оцінки цих параметрів було проведено серію тестів із симуляцією некоректного введення даних та переривання мережевого з’єднання.

Першим етапом практичного тестування стала комплексна перевірка стійкості підсистеми управління діалоговими сесіями (FSM) та локальної бази даних SQLite. Важливість цього етапу зумовлена тим, що саме ці компоненти відповідають за збереження правильної послідовності кроків користувача та цілісність зібраних проміжних даних. Оскільки архітектура користувацького інтерфейсу бота на етапі опитування передбачає сувору взаємодію виключно через натискання кнопок Inline-клавіатури, необхідно було перевірити надійність обробників подій при нетиповій поведінці. Для цього було навмисно зімітовано спробу порушити встановлений алгоритм маршрутизації шляхом надсилання довільного текстового повідомлення замість очікуваного системного запиту від кнопки. На рисунку 4.6 детально продемонстровано реакцію програмної системи на спробу введення користувачем неструктурованого тексту (хаотичного набору символів)

безпосередньо під час проходження базових кроків профорієнтаційного опитування.

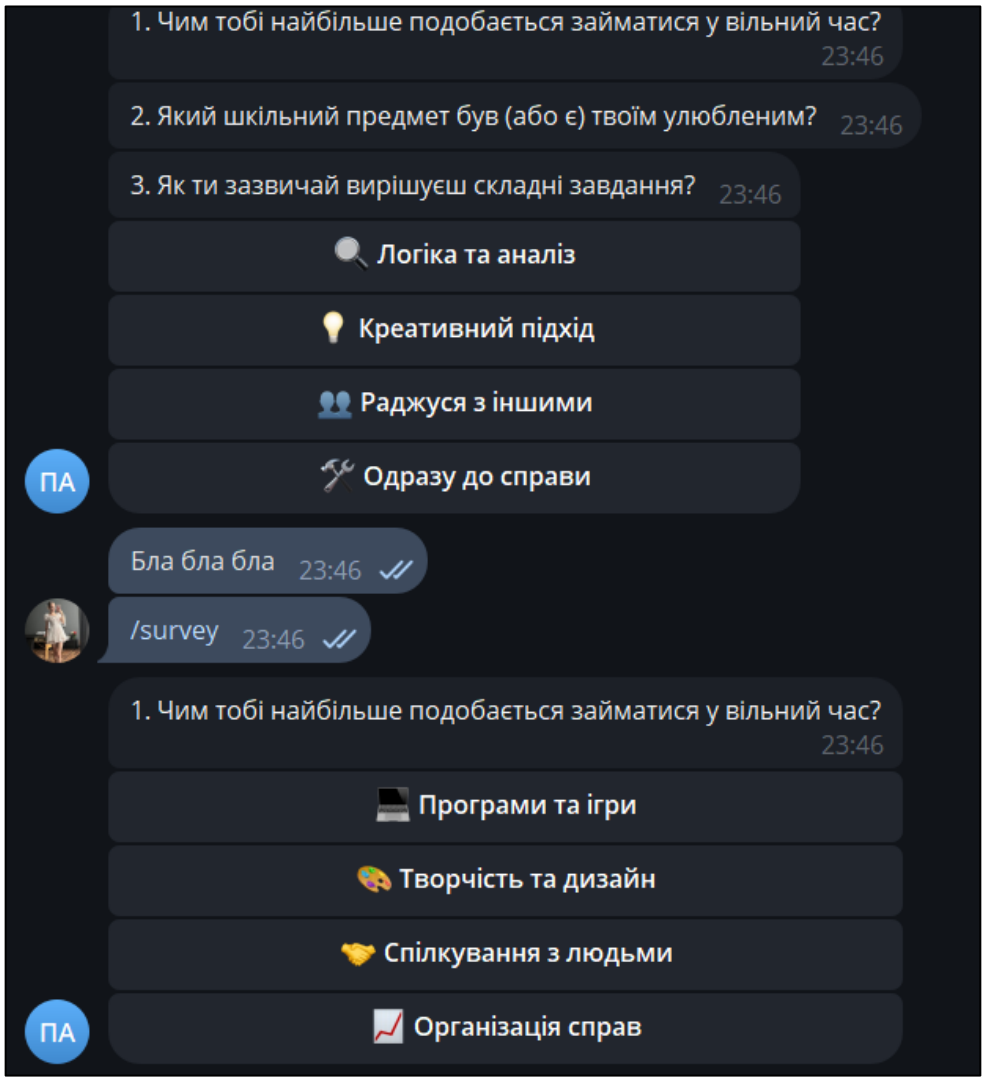


Рисунок 4.6 – Обробка некоректного текстового введення та переривання сесії

Як видно з результатів тестування, система успішно ігнорує текстові повідомлення, які не відповідають очікуваному формату подій (натискання кнопок), і не дозволяє користувачеві випадково порушити логіку маршрутизації. Бот залишається на поточному кроці, очікуючи правильної дії. Водночас, виклик системної команди /survey посеред активної сесії безпечно скидає попередній стан машин та ініціалізує тестування заново.

Важливо зазначити, що завдяки інтегрованій SQL-інструкції INSERT OR IGNORE у модулі збереження сесій, система не створює дублюючих записів у локальній базі даних, зберігаючи її структурну цілісність.

Другим критичним напрямом тестування стала перевірка стабільності системи у разі недоступності зовнішніх API. Оскільки фінальних профорієнтаційних рекомендацій прямо залежить від віддалених серверів хмарної мовної моделі Google Gemini, здатність системи витримувати мережеві тайм-аути або обмеження кількості запитів є обов'язковою вимогою.

На рисунку 4.7 наведено приклад програмної обробки помилки «503 Service Unavailable», яка виникає при тимчасовому перевантажені серверів провайдера штучного інтелекту.

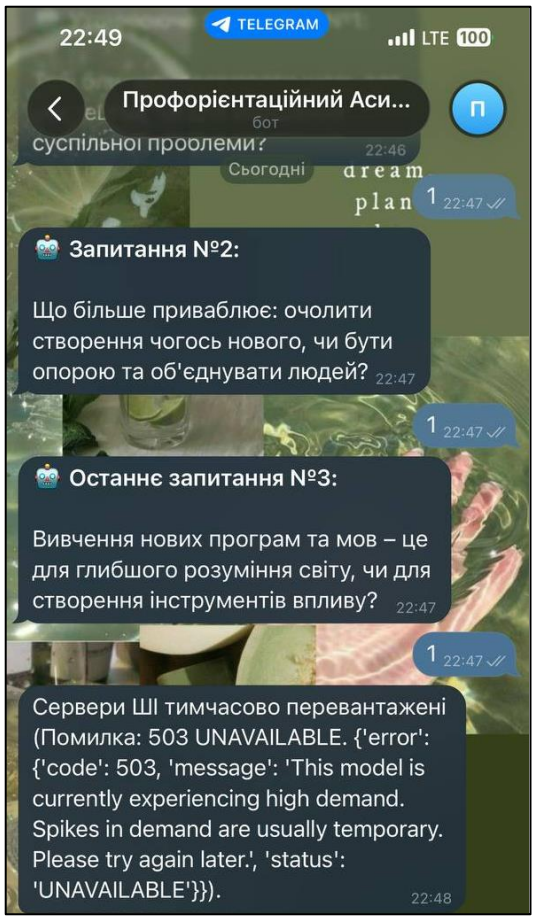


Рисунок 4.7 – Перехоплення помилки доступу до віддаленого API мовної моделі

Імплементація захисних конструкцій try-except у програмному коді повністю підтвердила свою ефективність. Замість критичного аварійного завершення серверного процесу та зупинки роботи бота для всіх інших клієнтів, програма безпечно перехоплює виняток. Бот генерує інформативне повідомлення для конкретного користувача, сповіщаючи про тимчасову затримку та пропонуючи спробувати пізніше.

Таким чином, результати проведеного тестування в нетипових умовах підтверджують високу інженерну стійкість розроблених алгоритмів. Реалізовані механізми ізоляції контексту, дедуплікації даних та перехоплення мережевих винятків гарантують безперервність функціонування Telegram-бота, захищають асинхронний потік від блокування та забезпечують високий рівень користувацького досвіду у разі виникнення позаштатних ситуацій.

Підсумовуючи результати всіх етапів перевірки, описаних у цьому розділі, можна констатувати, що комплексне тестування підтвердило повну працездатність розробленої профорієнтаційної системи в умовах, наближених до реальної експлуатації. Аналіз базового сценарію використання довів високу точність когнітивного ядра: генеративна модель успішно аналізує вхідні відповіді та адаптивно формує релевантні рекомендації, а RAG-підсистема безпомилково здійснює векторний пошук, забезпечуючи користувача веріфікованою інформацією з реєстру ЄДБО та статистикою з Work.ua. Одночасно з цим, успішна перевірка системи на відмовостійкість гарантувала надійність архітектури при нетипових діях користувача та мережевих збоях.

Отже, результати тестування підтверджують правильність обраних інженерних рішень. Розроблений програмний комплекс функціонує стабільно, мінімізує ризики виникнення програмних збоїв завдяки гнучкій системі управління сесіями та є повністю готовим до впровадження у практичний процес інтелектуальної професійної орієнтації абітурієнтів.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне наукове-прикладне завдання, яке полягає у розробці та програмній реалізації інтелектуальної системи професійної орієнтації з використанням генеративного штучного інтелекту та RAG-архітектури. Завдання виконано в повному обсязі: спроектовано та впроваджено багатофункціонального Telegram-бота, який забезпечує повний цикл профорієнтації – від збору первинних даних до видачі персоналізованих рекомендацій. З якісної точки зору розроблена система успішно усуває проблему «галюцинацій» мовних моделей завдяки жорсткій фактологічній прив’язці. Кількісні показники підтверджуються успішною масовою векторизацією даних та високою швидкістю обробки запитів, де семантичний збіг фільтрується за математичним порогом косинусної подібності понад 65%. Підсистема управління сесіями довела свою повну відмовостійкість під час імітації нетипової поведінки користувачів.

На відміну від існуючих вітчизняних аналогів, які переважно базуються на статичних психологічних тестах із жорсткими деревами рішень, розроблена система пропонує адаптивний діалог. Порівняно зі світовими рішеннями на базі глобальних великих мовних моделях, ця розробка має суттєву перевагу завдяки глибокій локалізації: вона оперує верифікованими даними офіційного українського реєстру ЄДБО та фінансовою аналітикою кадрового порталу Work.ua, що робить рекомендації максимально релевантними саме для українських абітурієнтів. Виконана робота безпосередньо пов’язана з науково-дослідною тематикою випускної кафедри штучного інтелекту у рамках досліджень напрямів інтелектуального аналізу даних, впровадження систем штучного інтелекту в освітній процес та розробка високонавантажених клієнт-серверних застосунків.

У процесі виконання роботи отримано нові практичні результати у вигляді комбінованої методики застосування технології Retrieval-Augmented Generation для задач психологічного профілювання. Цей підхід може знайти відображення у подальших наукових публікаціях на тему автоматизації освітніх процесів. У якості рекомендацій щодо подальшої роботи в даному напрямку доцільно розглянути розширення бази даних векторного сховища Pinecone.

Розроблений програмний продукт повністю готовий до використання за прямим призначенням – у закладах загальної середньої освіти для допомоги старшокласникам, а також у приймальній комісії та відділ профорієнтації університету для залучення нових вступників. Крім того, матеріали роботи, зокрема програмний код, алгоритми інтеграції API, методики побудови векторних індексів та парсингу, можуть бути ефективно використані у навчальному процесі університету під час викладення дисциплін, пов’язаних із проєктуванням систем штучного інтелекту, розробка чат-ботів та програмуванням мовою Python.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Індекс майбутнього. Професійні очікування та розвиток підлітків в Україні. *Фундація Олени Зеленської*. URL: https://zelenskafoundation.org/uk/future-index (дата звернення: 24.06.2026).

2. Освіта.UA. Про сайт «вступ.освіта.ua». *Вступ.Освіта.UA*. URL: https://vstup.osvita.ua/about.html (дата звернення: 24.06.2026).

3. Розвивай свою кар'єру через навчання. *EPAM | Campus*. URL: https://campus.epam.ua/ua/about (дата звернення: 24.06.2026).

4. Дія.Освіта – проєкти – міністерство цифрової трансформації україни. *Міністерство цифрової трансформації України*. URL: https://thedigital.gov.ua/projects/education/osvita (дата звернення: 24.06.2026).

5. About – CareerExplorer. *CareerExplorer*. URL: https://www.careerexplorer.com/about/ (date of access: 24.06.2026).

6. O*NET online help: my next move. *O*NET OnLine*. URL: https://www.onetonline.org/help/onet/mynextmove (date of access: 24.06.2026).

7. Уткін Є. І. Дослідження методу покращення роботи великих мовних моделей з використанням технології RAG : магістерська робота. Харків, 2025. 84 с. URL: https://openarchive.nure.ua/entities/publication/9ca89760-23c9-4a47-9434-4914eff88e58 (дата звернення: 24.06.2026).

8. Любар Б. В. Створення Telegram-боту з використанням OpenAI API. Київ : Нац. ун-т «Києво-Могил. акад.», 2023. 33 с. URL: https://ekmair.ukma.edu.ua/handle/123456789/28457 (дата звернення: 17.04.2026).

9. Models | gemini API | google AI for developers. *Google AI for Developers*. URL: https://ai.google.dev/gemini-api/docs/models (date of access: 24.06.2026).

10. Нич А., Праворська Н. Rag (retrieval-augmented generation) як нова парадигма корпоративної автоматизації. *Herald of Khmelnytskyi National University. Technical sciences*. 2026. Т. 361, № 1. С. 375–382. URL: https://doi.org/10.31891/2307-5732-2026-361-53 (дата звернення: 24.06.2026).

11. What is a vector database & how does it work?. *Google Cloud*. URL: https://cloud.google.com/discover/what-is-a-vector-database (date of access: 24.06.2026).

12. The faiss library / M. Douze et al. *IEEE transactions on big data*. 2025. P. 1–17. URL: https://doi.org/10.1109/tbdata.2025.3618474 (date of access: 24.06.2026).

13. Abidin A. Z., Engel M. M. Comparative analysis of performance aspects between chroma and pgvector as a vector database. *Bit-Tech*. 2025. Vol. 8, no. 2. P. 2079–2090. URL: https://doi.org/10.32877/bt.v8i2.3198 (date of access: 24.06.2026).

14. BPMN – вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/BPMN (дата звернення: 24.06.2026).

15. React.js для розробки веб-додатків та інтерфейсів – Brander. *IT-компанія з розробки цифрових продуктів в Україні – Brander*. URL: https://brander.ua/technologies/reactjs (дата звернення: 24.06.2026).

16. Brander. Vue.js: прогресивний фреймворк для сучасних інтерфейсів. *Блог Brander*. URL: http://cloud.google.com/discover/vector-databases (дата звернення: 26.04.2026).

17. Wezom. Angular vs React: порівняння потужних інструментів для фронтенду. *Блог Wezom*. URL: https://wezom.com.ua/ua/blog/react-vs-angular (дата звернення: 26.04.2026).

18. Brander. Мова Kotlin як стандарт сучасної мобільної розробки. *Блог Brander*. URL: https://brander.ua/technologies/kotlin (дата звернення: 27.04.2026).

19. Brander. Переваги та особливості фреймворку Flutter для кросплатформної розробки. *Блог Brander*. URL: https://brander.ua/technologies/flutter (дата звернення: 27.04.2026).

20. Aiogram. aiogram Documentation. *aiogram*. URL: https://docs.aiogram.dev/ (date of access: 27.04.2026).

21. OpenAI. Hello GPT-4o: The most advanced multimodal model. *OpenAI*. URL: https://openai.com/index/hello-gpt-4o/ (date of access: 19.04.2026).

22. Anthropic. Introducing Claude 3.5 Sonnet. *Anthropic*. URL: https://www.anthropic.com/news/claude-3-5-sonnet (date of access: 28.04.2026).

23. Reid M., Savinov N., Teplyashin D. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. Google DeepMind, 2024. 154 p. (Preprint. Technical Report). URL: https://doi.org/10.48550/arXiv.2403.05530 (date of access: 28.04.2026).

24. Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv.org. URL: https://arxiv.org/abs/2005.11401 (date of access: 29.04.2026).

25. Харовюк В. Chroma vs Qdrant vs pgvector: яку векторну БД обрати у 2026? WebCraft. URL: https://webscraft.org/blog/chromadb-qdrant-abo-pgvector-yak-obrati-vector-db-pid-sviy-proyekt (дата звернення: 29.05.2026).

26. Samuel H. Learning vector databases: a practical deep dive with ChromaDB. *Log in or sign up*. URL: https://www.linkedin.com/pulse/learning-vector-databases-practical-deep-dive-hamman-samuel-phd-v7rzc/ (date of access: 29.05.2026).

27. Що таке SQLite? FREEhost.UA. URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/ (дата звернення: 30.04.2026).

28. Розробка сайтів з базою даних POSTGRESQL: з нуля та під ключ – Brander. *IT-компанія з розробки цифрових продуктів в Україні – Brander*. URL: https://brander.ua/technologies/postgresql (дата звернення: 30.04.2026).

29. Що таке JSON – QALight. *QALight*. URL: https://qalight.ua/baza-znaniy/shho-take-json/ (дата звернення: 30.04.2026).

30. Про формат CSV | Dell Ukraine. *Access Denied*. URL: https://www.dell.com/support/kbdoc/uk-ua/000186211/ (дата звернення: 30.04.2026).

31. TIOBE Software BV. TIOBE Index for April 2026. *TIOBE*. URL: https://www.tiobe.com/tiobe-index/ (date of access: 03.05.2026).

32. Phillips D. Python Object-Oriented Programming: Build robust and maintainable object-oriented Python applications and libraries. Birmingham : Packt Publishing Ltd, 2021. 769 p. URL: https://studylib.net/doc/26237168/python-object-oriented-programming-build-robust-and-maint...?p=4 (date of access: 03.05.2026).

33. Casciaro M., Mammino L. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. 2nd ed. Birmingham : Packt Publishing Ltd, 2020. 674 p.

34. Rao S. JavaScript for modern web development: building a web application using HTML, CSS, and JavaScript. New Delhi : BPB Publications, 2020. 411 p.

35. Loy M., Niemeyer P. Learning Java: an introduction to real-world programming with Java. 5th ed. Sebastopol : O'Reilly Media, 2020. 544 p.

36. Sologu A. Comparing programming languages. LinkedIn. URL: https://www.linkedin.com/pulse/comparing-programming-languages-anda-sologu-bt2ff/ (date of access: 05.05.2026).

37. Мови програмування: список сучасних мов, з якої мови почати вивчення програмування – Ukrainian Digital Community. *Ukrainian Digital Community*. URL: https://ukrainiandigital.com/strong-movy-prohramuvannia-dlia-pochatkivtsiv-ta-dosvidchenykh-ohliad-populiarnykh-strong/ (дата звернення: 06.05.2026).