

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)

_____ Інтелектуальна система підтримки публічних заходів кафедри ШІ

(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-3

_____ Андрій ПАВЛУНІК
(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. Дмитро МАЛЄСВ
(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 _____ Павлуніку Андрію Юрійовичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Інтелектуальна система підтримки публічних заходів кафедри ШІ _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 8 липня 2026 р.

3. Вихідні дані до роботи Python, FastAPI, React.js, PostgreSQL, SQLAlchemy, ChromaDB, Tailwind CSS, Recharts, Google Gemini API, Ollama, Pydantic, httpx, JWT, офіційна документація до використаних технологій та науково-технічні публікації щодо архітектур RAG та систем відмовостійкості

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі

2) Проєктування архітектури та інтелектуальних компонентів системи

3) Програмна реалізація та експериментальні дослідження

4) Тестування та оцінка ефективності

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 8 червня 20 26 р.

Здобувач А.П.В.
(підпис)

Керівник роботи _____
(підпис)

ас. Дмитро МАЛЄСВ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 60 с., 14 рис., 1 дод., 16 джерел.

ГЕЙМІФІКАЦІЯ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, ПУБЛІЧНІ ЗАХОДИ, CHROMADB, FASTAPI, GENERATIVE AI, RAG ARCHITECTURE, REACT.

Об'єкт дослідження – процеси інформаційного супроводу та менеджменту публічних заходів кафедри штучного інтелекту у закладі вищої освіти.

Предмет дослідження – інтелектуальна система на основі архітектури генерації з доповненням пошуку (RAG) та великих мовних моделей для автоматизації комунікації та гейміфікації освітніх подій.

Мета роботи – підвищення ефективності взаємодії кафедри з учасниками публічних заходів шляхом розробки програмного комплексу, здатного автоматично надавати контекстні відповіді на запити на базі кафедральної документації та генерувати тести для залучення аудиторії.

Методи дослідження – використано системний аналіз, методи обробки природної мови (NLP) та векторного представлення тексту, методи проєктування архітектури програмного забезпечення; методи тестування.

У кваліфікаційній роботі розроблено інтелектуальну систему, що автоматизує менеджмент публічних заходів кафедри ІІІ. Реалізовано архітектуру RAG із використанням векторної бази даних ChromaDB, яка дозволяє ІІІ-агенту синтезувати відповіді на основі специфічної кафедральної документації. Розроблено модуль гейміфікації з трирежимним генератором тестів та системою аналітики. Практична цінність роботи полягає у забезпеченні автономності та високої фактологічної точності інформаційного супроводу заходів, що мінімізує навантаження на викладачів та підвищує залученість студентів.

ABSTRACT

Bachelor's thesis contains: 60 pp., 14 fig., 1 ann., 16 references.

CHROMADB, FASTAPI, GAMIFICATION, GENERATIVE AI, INTELLIGENT SYSTEM, PUBLIC EVENTS, RAG ARCHITECTURE, REACT.

Research Object – processes of information support and management of public events organized by the Department of Artificial Intelligence at a higher education institution.

Research Subject – an intelligent system based on Retrieval-Augmented Generation (RAG) architecture and large language models for automating communication and gamifying educational events.

The objective of this work is to improve the effectiveness of the department's interaction with participants in public events by developing a software suite capable of automatically providing context-aware responses to queries based on departmental documentation and generating quizzes to engage the audience.

Research methods – system analysis, natural language processing (NLP) and vector text representation methods, software architecture design methods, and testing methods were used.

This thesis presents an intelligent system that automates the management of public events organized by the AI Department. A RAG architecture was implemented using the ChromaDB vector database, which enables the AI agent to synthesize responses based on specific departmental documentation. A gamification module was developed, featuring a three-mode test generator and an analytics system. The practical value of this work lies in ensuring the autonomy and high factual accuracy of information support for events, which minimizes the workload on faculty and increases student engagement.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	10
1 Аналіз предметної галузі та постановка задачі.....	12
1.1 Дослідження процесів організації та інформаційного супроводу публічних заходів на кафедрі штучного інтелекту ХНУРЕ	12
1.2 Огляд та критичний аналіз існуючих систем менеджменту подій та платформ освітньої гейміфікації	14
1.3 Аналіз сучасних підходів до генеративного штучного інтелекту: архітектура RAG та великі мовні моделі (LLM).....	15
1.4 Обґрунтування доцільності розробки інтелектуальної системи та постановка інженерних задач проєктування	17
2 Проєктування архітектури та інтелектуальних компонентів системи	19
2.1 Обґрунтування архітектурного патерну системи та вибір технологічного стека.....	19
2.1.1 Переваги асинхронної архітектури FastAPI (Python) для бекенд-сервісів	19
2.1.2 Обґрунтування вибору React та Clean UI концепції для клієнтської частини.....	20
2.2 Проєктування реляційної бази даних та об'єктно-реляційного відображення (ORM SQLAlchemy)	21
2.2.1 Моделювання сутностей підсистеми Event Manager та механізмів реєстрації	22
2.2.2 Проєктування схем даних для модуля гейміфікованого тестування (Quiz, QuizQuestion, QuizResult)	23
2.3 Архітектурні рішення інтелектуального модуля RAG	24
2.4 Розробка відмовостійкого гібридного алгоритму каскадної маршрутизації ШІ-моделей (Fallback)	27
2.5 Архітектура безпеки та розмежування прав доступу користувачів ..	28

3 Програмна реалізація та експериментальні дослідження	30
3.1 Реалізація серверної логіки (Backend) та оптимізація роботи з даними	30
3.1.1 Програмування асинхронних API-маршрутів (Endpoints) та lifespan-менеджменту в FastAPI	30
3.1.2 Програмний інтерфейс взаємодії з ChromaDB	33
3.2 Програмна реалізація ШІ-агента генерації та валідації контенту	35
3.2.1 Логіка трирежимного генератора тестів (за PDF-контекстом, за вільною темою, масове збереження).....	35
3.2.2 Алгоритми жорсткої структурної валідації JSON-відповідей від мовних моделей.....	38
3.3 Розробка клієнтського інтерфейсу (Frontend) та інструментів інтерактивної аналітики	41
3.3.1 Архітектура SPA-додатка та концепція Clean UI	42
3.3.2 Розробка дашборду аналітики подій.....	43
3.3.3 Інтерфейси модуля гейміфікації та керування ШІ-генерацією.	44
3.4 Програмна реалізація механізмів автентифікації та захисту маршрутів	46
4 Тестування та оцінка ефективності	48
4.1 Тестування функціональності AI-чату.....	48
4.2 Оцінка точності генерації тестів та сценарії проходження квізів.....	50
4.3 Тестування системи розмежування прав доступу	54
Висновки	56
Перелік джерел посилання	58
Додаток А Відомість кваліфікаційної роботи	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЗВО – заклад вищої освіти;

ХНУРЕ – Харківський національний університет радіоелектроніки;

API – Application Programming Interface – інтерфейс прикладного програмування, набір інструкцій та функцій для взаємодії між програмними компонентами;

ChromaDB – векторна база даних з відкритим кодом для ефективного зберігання та семантичного пошуку векторних представлень даних;

CSS – Cascading Style Sheets – каскадні таблиці стилів, мова для опису зовнішнього вигляду вебсторінок;

FastAPI – сучасний висопродуктивний асинхронний вебфреймворк для розробки API на мові Python;

JSON – JavaScript Object Notation – текстовий формат обміну даними, що базується на синтаксисі JavaScript;

JWT – JSON Web Token – відкритий стандарт для створення токенів доступу, що забезпечують безпечний обмін інформацією між сторонами;

LLM – Large Language Model – велика мовна модель, тип штучного інтелекту, навчений на великих масивах текстових даних;

NLP – Natural Language Processing – обробка природної мови, галузь штучного інтелекту, що займається аналізом та генерацією тексту;

ORM – Object-Relational Mapping – технологія програмування, що поєднує об'єктно-орієнтовані моделі даних з реляційними базами даних;

RAG – Retrieval-Augmented Generation – архітектура генерації з доповненням пошуку, що дозволяє мовним моделям використовувати зовнішні джерела знань;

RBAC – Role-Based Access Control – модель управління доступом до системи на основі ролей користувачів;

React – бібліотека мови JavaScript для створення користувацьких інтерфейсів;

SPA – Single Page Application – односторінковий вебдодаток, що завантажує всю необхідну інформацію при першому запиті;

SQL – Structured Query Language – мова структурованих запитів до реляційних баз даних;

UI – User Interface – користувацький інтерфейс, сукупність засобів взаємодії користувача з програмним продуктом;

UX – User Experience – користувацький досвід, рівень зручності та задоволеності при взаємодії користувача з системою.

ВСТУП

Організація публічних заходів на кафедрі штучного інтелекту ХНУРЕ, таких як дні відкритих дверей, наукові семінари, олімпіади та хакатони, вимагає оперативного та зручного інформаційного супроводу. Проте традиційні підходи до інформування учасників часто стикаються з проблемою розрізненості даних. Важливі регламенти, навчальні плани та правила вступу зазвичай зберігаються у великих PDF-документах, що ускладнює швидкий пошук відповідей користувачами та створює значне операційне навантаження на викладачів й організаторів подій, які змушені однотипно відповідати на повторювані запити.

Ефективним розв'язанням цієї проблеми є впровадження систем генеративного штучного інтелекту на основі великих мовних моделей. Для уникнення поширеної проблеми «галюцинацій» та інтеграції специфічних знань кафедри доцільно використовувати архітектуру RAG (Retrieval-Augmented Generation), яка доповнює запити контекстом із документів. Разом із цим, розробка таких інтелектуальних систем потребує забезпечення високої відмовостійкості. Хмарні API комерційних моделей схильні до перевантажень та мережових збоїв, що зумовлює необхідність створення гібридної архітектури з каскадними механізмами маршрутизації (Fallback), здатними у разі помилки автоматично перемикає запити на локальний ІІІ-сервер Ollama. Важливим елементом підвищення залученості є також модулі гейміфікації та тестування, інтегровані в єдину вебплатформу.

Об'єктом дослідження є процеси інформаційного супроводу, менеджменту та інтерактивної взаємодії з користувачами під час проведення публічних освітніх заходів. Предметом дослідження виступають моделі, алгоритми та програмні засоби побудови відмовостійких систем генеративного штучного інтелекту на основі архітектури RAG, каскадної маршрутизації хмарних та локальних великих мовних моделей, а також методів комп'ютерної гейміфікації.

Метою роботи є підвищення ефективності інформаційної підтримки та залученості учасників публічних заходів кафедри штучного інтелекту шляхом проєктування та програмної реалізації відмовостійкої інтелектуальної вебсистеми. Для досягнення мети вирішено завдання з аналізу процесів організації подій, дослідження теоретичних засад RAG, проєктування векторного сховища ChromaDB та розробки каскадного алгоритму перемикання ШІ-моделей. Також виконано завдання з моделювання реляційної бази даних PostgreSQL за допомогою SQLAlchemy, асинхронної реалізації бекенду на FastAPI з використанням HTTP-клієнта httpx, створення інтерфейсу на React та проведення експериментальних досліджень працездатності системи. У роботі застосовано методи системного аналізу, обробки природної мови (NLP) та архітектурного проєктування програмного забезпечення.

Наукова новизна отриманих результатів полягає в удосконаленні алгоритму відмовостійкого функціонування інтелектуальних агентів у вебсистемах шляхом впровадження каскадної схеми маршрутизації запитів, яка здійснює динамічний перехід від високопродуктивних хмарних моделей до автономного локального ШІ-сервера Ollama у разі виникнення помилок зв'язку або структурної валідації складних JSON-структур.

Практичне значення роботи полягає у створенні повнофункціональної вебсистеми підтримки заходів для кафедри штучного інтелекту ХНУРЕ. Розроблено оптимізований модуль семантичного пошуку по документах, трирежимний ШІ-генератор квізів та інтерактивний аналітичний дашборд на базі Recharts. Створений програмний комплекс готовий до розгортання та може слугувати базою для аналогічних систем в інших підрозділах університету.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження процесів організації та інформаційного супроводу публічних заходів на кафедрі штучного інтелекту ХНУРЕ

Сучасний етап розвитку системи вищої освіти в Україні супроводжується глибинними процесами цифрової трансформації. Цифровізація стає не лише інструментом адаптації до дистанційного чи змішаного форматів навчання, але й стратегічним вектором підвищення конкурентоспроможності закладів вищої освіти (ЗВО). В умовах відкритого освітнього простору та високої мобільності абітурієнтів, ефективна комунікація та організація публічних заходів відіграють ключову роль у формуванні культури студентів навчального закладу та залученні талановитої молоді [1].

Кафедра штучного інтелекту Харківського національного університету радіоелектроніки (ХНУРЕ) є осередком активної наукової та освітньої діяльності. Бізнес-процес життєдіяльності кафедри передбачає регулярну організацію та проведення низки публічних подій, серед яких:

- дні відкритих дверей для школярів та майбутніх абітурієнтів;
- науково-практичні семінари та конференції;
- студентські олімпіади та хакатони зі штучного інтелекту;
- відкриті лекції від запрошених фахівців ІТ-індустрії (стейкхолдерів).

Аналіз існуючої практики організації цих заходів виявив низку суттєвих проблем в їх інформаційному супроводі. Як зазначають дослідники у сфері управління освітніми системами, традиційні інформаційні системи ЗВО часто характеризуються фрагментованістю та базуються на застарілих методах передачі інформації. На кафедрі штучного інтелекту значний обсяг критично важливих даних (навчальні плани, регламенти проведення олімпіад, правила вступу, методичні вказівки)

зберігається у вигляді статичних документів формату PDF або розрізнених текстових публікацій на офіційному сайті.

Такий формат подачі інформації створює високий бар'єр для користувача. Замість швидкого отримання конкретної відповіді, учасник заходу змушений самостійно завантажувати об'ємні файли та здійснювати ручний пошук потрібного фрагмента тексту. Це призводить до зниження рівня залученості (engagement) аудиторії та втрати частини потенційних учасників ще на етапі реєстрації.

З іншого боку, неефективний інформаційний супровід генерує значне додаткове навантаження на організаторів подій – викладачів та адміністративний персонал кафедри. В умовах цифровізації навантаження на викладачів часто зростає саме через необхідність постійної онлайн-комунікації та модерації. Під час підготовки до хакатонів чи днів відкритих дверей відповідальні особи змушені витратити десятки годин на опрацювання однотипних повідомлень у месенджерах, відповідаючи на питання щодо розкладу, дедлайнів або технічних вимог, які вже описані в регламентах.

Крім того, існуючі платформи для організації подій (наприклад, Google Forms або розрізнені Telegram-боти) не забезпечують єдиної екосистеми, яка б поєднувала процес реєстрації, надання інформації, гейміфікацію (тестування/квізи) та аналітику. Відсутність централізованого інструменту для збору аналітики унеможливорює оперативне прийняття управлінських рішень щодо коригування маркетингової стратегії заходу.

Отже, виникає об'єктивна необхідність у розробці єдиної інтелектуальної системи, яка б автоматизувала рутинні процеси комунікації, забезпечила миттєвий семантичний доступ до документації кафедри та підвищила рівень інтерактивності заходів шляхом впровадження гейміфікованих елементів.

1.2 Огляд та критичний аналіз існуючих систем менеджменту подій та платформ освітньої гейміфікації

Для розв'язання завдань автоматизації реєстрації учасників та підвищення їхньої залученості на сучасному ринку програмного забезпечення існує низка готових комерційних рішень. Їх можна умовно поділити на дві великі категорії: системи менеджменту подій (Event Management Systems) та спеціалізовані платформи освітньої гейміфікації.

До першої категорії належать такі популярні хмарні сервіси, як Eventbrite, Meetup, а також універсальні інструменти на кшталт Google Forms. Вони надають базовий функціонал для створення реєстраційних форм, збору електронних адрес та надсилання нагадувань. Однак головним недоліком цих платформ у контексті закладів вищої освіти є їхня інформаційна ізольованість. Такі системи не здатні інтегруватися з внутрішньою документацією кафедри, опрацьовувати об'ємні методичні матеріали чи регламенти хакатонів. Відповідно, вони не можуть надавати учасникам інтелектуальні, контекстні відповіді на специфічні запитання. Будь-який запит користувача, що виходить за межі базового опису події на сторінці реєстрації, потребує ручного втручання адміністратора, що нівелює переваги автоматизації.

Друга категорія охоплює платформи для інтерактивного тестування та гейміфікації: Kahoot!, Quizizz та відповідні компоненти системи дистанційного навчання Moodle. Як зазначають вітчизняні дослідники, впровадження ігрових механік та квізів в освітній процес стимулює внутрішню мотивацію здобувачів, нівелює стрес та сприяє глибшому залученню в матеріал [2]. Проте з інженерної та організаційної точок зору ці платформи мають суттєвий недолік – вони вимагають ручного створення контенту. Викладач або модератор заходу змушений самотійно вигадувати питання, формулювати хибні варіанти відповідей та налаштовувати логіку оцінювання, витрачаючи на це значну кількість часу. Жодна зі згаданих

платформ не має надійних вбудованих механізмів генерації тестів безпосередньо з PDF-документів користувача з використанням сучасних великих мовних моделей.

Крім того, використання хмарних SaaS-рішень (Software as a Service) часто супроводжується жорсткими тарифними обмеженнями на кількість одночасних учасників заходу або лімітами на звернення до бази даних. Це створює додаткові інфраструктурні та фінансові ризики для кафедри під час проведення масштабних подій.

Критичний аналіз існуючих програмних продуктів засвідчив відсутність на ринку єдиного комплексного рішення, яке б органічно поєднувало функціонал реєстрації учасників (Event Manager) з інтелектуальним асистентом на базі архітектури RAG та автоматизованим генератором гейміфікованих тестів. Саме тому розробка власної незалежної системи з оптимізованими запитами до локальної бази даних та каскадним відмовостійким ШІ-модулем є оптимальним шляхом забезпечення інформаційних потреб кафедри штучного інтелекту ХНУРЕ.

1.3 Аналіз сучасних підходів до генеративного штучного інтелекту: архітектура RAG та великі мовні моделі (LLM)

Швидкий розвиток великих мовних моделей (Large Language Models, LLM) відкрив нові перспективи для автоматизації процесів обробки природної мови. Сучасні LLM, такі як моделі сімейства Google Gemini, здатні генерувати зв'язні тексти, аналізувати складні запити та синтезувати програмний код. Проте використання базових (pre-trained) мовних моделей в ізольованих університетських середовищах, таких як кафедра університету, має суттєві обмеження. Головними проблемами є ефект «галюцинацій» (генерація правдоподібної, але хибної інформації) та відсутність доступу до актуальних локальних даних, які не входили до

навчального набору моделі (наприклад, розклад конкретного хакатона або внутрішні методичні вказівки).

Традиційним методом адаптації LLM до нових даних раніше вважалося донавчання (Fine-Tuning). Однак цей підхід є обчислювально затратним і не дозволяє системі динамічно оновлювати свої знання без повторного перенавчання. Кардинально іншим та найбільш ефективним інженерним підходом для науково-містких завдань стала архітектура генерації з доповненням пошуку – RAG (Retrieval-Augmented Generation). Принцип роботи цієї архітектури наведено на рисунку 1.1.

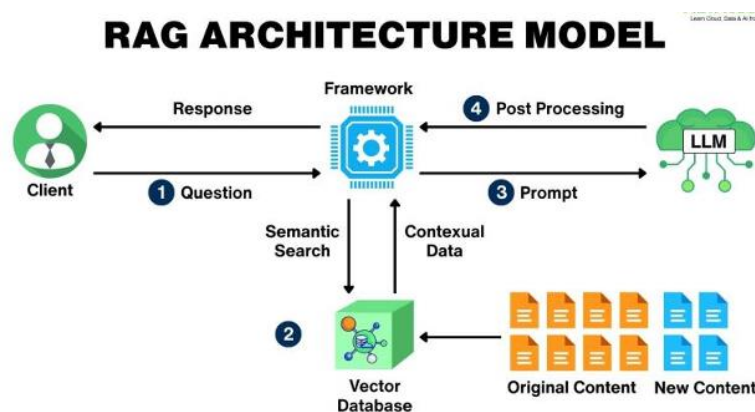


Рисунок 1.1 – Загальна архітектура конвеєра RAG (Retrieval Augmented Generation)

Концепція RAG [3] поєднує параметричну пам'ять попередньо навченої моделі з непараметричною пам'яттю (зовнішньою базою даних). Ця архітектура складається з двох основних модулів: пошукового (Retriever) та генеративного (Generator). Процес обробки запиту відбувається за таким алгоритмом: спочатку вся внутрішня документація кафедри (PDF-файли, регламенти) розбивається на змістовні фрагменти (чанки) і за допомогою моделей перетворення тексту (Embedding models) трансформується у багатовимірні числові вектори. Ці вектори зберігаються у спеціалізованій векторній базі даних (наприклад, ChromaDB).

Коли користувач задає питання або система отримує команду на генерацію тесту, пошуковий модуль конвертує цей запит у вектор та виконує пошук найближчих сусідів (Similarity Search) у векторному просторі. Знайдені релевантні фрагменти тексту вилучаються з бази та динамічно ін'єктуються у контекст підказки (Prompt) для мовної моделі. Генеративний модуль (LLM) аналізує цей контекст і формує точну, фактологічно підтверджену відповідь, спираючись виключно на надані документи.

Впровадження архітектури RAG для підтримки публічних заходів кафедри штучного інтелекту є оптимальним рішенням. Вона усуває ризик галюцинацій, дозволяє миттєво оновлювати «знання» системи шляхом простого завантаження нових PDF-файлів у базу без зміни коду чи перенавчання моделі, а також забезпечує можливість створення високоточних гейміфікованих тестів, які строго відповідають затвердженним навчальним матеріалам.

1.4 Обґрунтування доцільності розробки інтелектуальної системи та постановка інженерних задач проектування

Аналіз предметної галузі, проведений у попередніх підпунктах, свідчить про наявність суттєвого розриву між потребами сучасної кафедри штучного інтелекту в оперативній комунікації та існуючими технологічними можливостями. Наявні інструменти (статичні сайти, розрізнені месенджери, прості форми збору даних) не забезпечують необхідного рівня автоматизації, не дозволяють інтегрувати специфічні знання кафедри у процес взаємодії з абітурієнтами та не містять інструментів для аналізу результатів заходів.

Доцільність розробки спеціалізованої інтелектуальної системи зумовлена необхідністю подолання «інформаційного вакууму» та оптимізації ресурсомістких процесів організації подій. Впровадження

системи дозволить перетворити статичну документацію кафедри на динамічне джерело знань, доступне користувачам у форматі природного діалогу з ШІ-асистентом. Це не лише знижує навантаження на організаторів, а й підвищує якість сервісу для учасників, надаючи їм персоналізовану підтримку 24/7.

Вибір технологічного стека – FastAPI для бекенду та React для фронтенду – є найбільш доцільним для створення високопродуктивної, масштабованої системи з сучасним користувацьким інтерфейсом. Використання векторної бази даних ChromaDB у поєднанні з RAG-архітектурою дозволяє забезпечити високу точність відповідей, тоді як впровадження каскадного механізму Fallback-маршрутизації (Gemini API, Ollama) мінімізує ризики відмови системи в умовах нестабільного мережевого з'єднання або обмежень хмарних сервісів.

На основі проведеного аналізу сформульовано основні інженерні задачі, які необхідно розв'язати в межах кваліфікаційної роботи:

- проєктування гнучкої моделі бази даних (Event Manager, Quiz), здатної ефективно підтримувати ієрархічні структури даних подій та результати тестувань;
- розробка інтелектуального модуля обробки документів, що включає алгоритми нарізки тексту, семантичного індексування та векторного пошуку;
- програмна реалізація механізму керування ШІ-агентами, здатного до автоматичного перемикання між хмарними та локальними моделями;
- розробка інтерфейсу користувача, що поєднує функціонал адміністрування заходів та інструменти аналітичної звітності.

Вирішення цих задач дозволить створити закінчений програмний продукт, який не лише вирішить актуальні проблеми комунікації кафедри штучного інтелекту, а й стане технологічним базисом для майбутнього впровадження інтелектуальних систем в освітній процес університету.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ІНТЕЛЕКТУАЛЬНИХ КОМПОНЕНТІВ СИСТЕМИ

2.1 Обґрунтування архітектурного патерну системи та вибір технологічного стека

2.1.1 Переваги асинхронної архітектури FastAPI (Python) для бекенд-сервісів

Вибір технологічного стека для розробки інтелектуальної системи підтримки публічних заходів кафедри ІІІ базувався на вимогах до високої продуктивності, здатності до обробки асинхронних запитів та простоти інтеграції зі спеціалізованими ІІІ-бібліотеками. В якості базового фреймворку було обрано FastAPI.

FastAPI – це сучасний високопродуктивний вебфреймворк для створення API на базі стандартних типових підказок Python (Python Type Hints). Однією з ключових переваг FastAPI є його побудова на основі асинхронної архітектури (async/await), що є критично важливим для систем, які працюють із зовнішніми API, такими як Google Gemini, та виконують тривалі операції введення-виведення. На відміну від класичних синхронних фреймворків (наприклад, Flask або Django у його початкових версіях), де один запит може повністю блокувати потік обробки, FastAPI дозволяє серверу ефективно опрацьовувати тисячі одночасних запитів, не створюючи надмірного навантаження на ресурси процесора [4].

В контексті нашої системи ця архітектура має два суттєві плюси:

- неблокуюча взаємодія з ІІІ-ядрами: при відправці запиту до Google Gemini API або локального сервера Ollama бекенд не очікує завершення обробки в режимі очікування (blocking wait), а вивільняє event loop для обробки запитів інших користувачів. Це дозволяє системі підтримувати

високий рівень відгуку (low latency) навіть за умови одночасного звернення багатьох студентів;

– інтеграція з RAG: операції пошуку векторів у ChromaDB та завантаження PDF-документів є ресурсомісткими. Завдяки асинхронній природі фреймворку, процес індексації знань кафедри відбувається фоново, не заважаючи основному функціоналу системи (Event Manager).

Крім архітектурних переваг, FastAPI забезпечує автоматичну генерацію документації Swagger (OpenAPI), що суттєво спрощує процес налагодження та тестування ендпоінтів. Завдяки використанню pydantic для валідації даних, система гарантує високу надійність вхідних запитів, що особливо актуально при передачі складних JSON-структур між клієнтом та сервером. Це дозволяє мінімізувати помилки на етапі виконання (runtime errors) та забезпечує відповідність API-запитів до суворих вимог III-моделей.

Таким чином, FastAPI було визначено як оптимальний фундамент для бекенд-сервісів, що дозволяє поєднати швидкість обробки даних з високою надійністю та гнучкістю інтеграції інтелектуальних модулів.

2.1.2 Обґрунтування вибору React та Clean UI концепції для клієнтської частини

Для реалізації клієнтської частини інтелектуальної системи було обрано бібліотеку React. Це рішення зумовлене потребою у створенні високого рівня інтерактивності, що є критично важливим для дашбордів аналітики та динамічних інтерфейсів створення тестів. Компонентна архітектура React дозволяє відокремити логіку відображення від бізнес-логіки, що спрощує подальшу підтримку та розширення системи [5].

Використання віртуального DOM (Document Object Model) у React забезпечує високу швидкість оновлення інтерфейсу без необхідності перезавантаження всієї сторінки, що є принципово важливим для роботи

дашборда з інтерактивними графіками (бібліотека Recharts). Завдяки цій технології, дані про реєстрації учасників або результати квізів оновлюються динамічно, забезпечуючи «безшовну» взаємодію користувача з платформою.

Важливим аспектом проєктування клієнтської частини стала концепція Clean UI. Основною метою впровадження цього підходу є зменшення когнітивного навантаження на користувача (студента або організатора заходу) шляхом усунення зайвих візуальних елементів та фокусування на ключових функціях. У межах роботи це реалізовано через:

- типографічну ієрархію: чіткий розподіл заголовків та основного тексту для кращої читабельності документації;
- спрощені форми введення: використання мінімалістичних стилів для sticky-форм створення заходів, що мінімізує помилки при введенні даних;
- колірну семантику: використання обмеженої палітри кольорів, де кожен відтінок несе функціональне навантаження (наприклад, візуальне розділення станів тестування).

Інтеграція React-компонентів із бекендом на FastAPI відбувається через асинхронні HTTP-запити за допомогою клієнта axios, що забезпечує швидку реакцію інтерфейсу на дії користувача. Така концепція побудови інтерфейсу не лише відповідає сучасним стандартам веброзробки, а й суттєво підвищує зручність роботи з інтелектуальними інструментами системи, дозволяючи користувачу зосередитися на контенті, а не на опануванні складного навігаційного меню.

2.2 Проєктування реляційної бази даних та об'єктно-реляційного відображення (ORM SQLAlchemy)

Важливим етапом проєктування інформаційної системи є створення стабільної, нормалізованої та продуктивної архітектури збереження даних.

Для керування інформаційними потоками публічних заходів та результатами тестування було обрано реляційну систему керування базами даних (СУБД) PostgreSQL. Вибір зумовлений її високою надійністю, підтримкою ACID-транзакцій та високою швидкістю під час роботи зі складними запитам [6].

Для забезпечення взаємодії між об'єктно-орієнтованим кодом мови програмування Python та реляційною структурою бази даних впроваджено сучасну бібліотеку SQLAlchemy. Вона об'єднує механізми SQLAlchemy (інжиніринг ORM) та Pydantic (валідація типів даних). Використання SQLAlchemy у межах даного дипломного проєкту дозволило розв'язати класичну проблему дублювання коду (code duplication), коли для бази даних та для API-ендпоінтів створюються окремі моделі.

У розробленій системі один і той самий клас Python одночасно описує таблицю бази даних і структуру для автоматичної валідації вхідних та вихідних JSON-пакетів. Взаємодія з базою даних реалізована через асинхронні сесії за допомогою драйвера asyncpg, що повністю відповідає подійній неблокуючій моделі асинхронного фреймворку FastAPI.

2.2.1 Моделювання сутностей підсистеми Event Manager та механізмів реєстрації

Підсистема Event Manager призначена для централізованого управління публічними подіями кафедри штучного інтелекту ХНУРЕ та забезпечення механізму реєстрації відвідувачів. Під час проєктування цієї підсистеми було декомпоновано та змодельовано дві ключові взаємопов'язані сутності: Event (Публічний захід) та EventRegistration (Реєстрація на захід).

Сутність Event інкапсулює базову інформацію про заплановану подію. Вона містить унікальний ідентифікатор, назву заходу, детальний текстовий опис регламенту, а також типізацію події (наприклад, вебінар,

воркшоп, день відкритих дверей). Для забезпечення просторово-часової прив'язки реалізовано атрибуту дати, часу та локації (або посилання на онлайн-зустріч). Додатково передбачено збереження шляху до графічної обкладинки заходу та фіксацію системного часу створення запису.

Сутність EventRegistration відповідає за збереження даних про учасників. Вона включає персональні дані (прізвище, ім'я) та контактну інформацію (електронна пошта) користувача.

Між цими сутностями встановлено класичний реляційний зв'язок типу «один до багатьох» (One-to-Many). Один захід може містити велику кількість зареєстрованих учасників, тоді як кожна реєстрація строго прив'язана лише до однієї події через механізм зовнішнього ключа (event_id). Для забезпечення швидкого пошуку та уникнення деградації продуктивності бази даних під час масових реєстрацій студентів, поле зовнішнього ключа було примусово проіндексоване на рівні СУБД. Це дозволяє оптимізувати операції фільтрації та зчитування великих списків учасників адміністраторами кафедри.

2.2.2 Проектування схем даних для модуля гейміфікованого тестування (Quiz, QuizQuestion, QuizResult)

Модуль гейміфікованого тестування та інтерактивних квізів призначений для закріплення отриманих знань учасниками публічних заходів та оцінки ефективності засвоєння інформаційного контенту. Архітектура збереження даних для цього модуля побудована навколо трьох взаємопов'язаних сутностей: Quiz (базовий тест), QuizQuestion (запитання тесту) та QuizResult (результати проходження).

Логічна структура модуля базується на каскадних реляційних зв'язках. Батьківська сутність Quiz містить загальну інформацію: назву, стислий опис тематики тесту та опціональний зв'язок із конкретним заходом підсистеми Event Manager.

Кожен тест складається з набору унікальних питань, які описуються сутністю QuizQuestion. Вона зберігає текстове формулювання питання, згенероване штучним інтелектом або введене адміністратором вручну, та символічний показчик правильного варіанта (наприклад, «А» або «Б»). Важливим архітектурним рішенням стало використання вбудованого в PostgreSQL типу даних JSONB (через серіалізацію списку рядків у SQLModel) для збереження масиву варіантів відповідей. Це дозволяє динамічно зберігати довільну кількість варіантів без створення додаткових таблиць-довідників, суттєво знижуючи кількість операцій об'єднання (JOIN) та оптимізуючи швидкість генерації карток питань на клієнтській частині додатка.

Для фіксації кожної спроби проходження квізу використовується сутність QuizResult. Вона зберігає ім'я користувача, фінальний бал у відсотковому еквіваленті, кількість правильних відповідей та загальну кількість питань на момент тестування.

Між базовим тестом та його питаннями і результатами реалізовано зв'язок «один до багатьох» із застосуванням каскадного видалення (ON DELETE CASCADE). Поля зовнішніх ключів в обох дочірніх таблицях також проіндексовані для запобігання зниженню продуктивності бази даних при одночасному записі чи зчитуванні результатів під час масового тестування аудиторії.

2.3 Архітектурні рішення інтелектуального модуля RAG

Для забезпечення відвідувачів та учасників публічних заходів кафедри III точною, фактологічно верифікованою інформацією з офіційних першоджерел (нормативних регламентів, правил вступу, описів освітніх програм) було спроектовано та реалізовано інтелектуальний аналітичний контур на базі архітектури генерації з доповненням пошуку (Retrieval-Augmented Generation, RAG). Головним завданням цього модуля є усунення

базових недоліків великих мовних моделей, пов'язаних із відсутністю знань про вузькоспецифічні закриті дані екосистеми кафедри та схильністю до «галюцинацій». Розроблене архітектурне рішення перетворює масиви неструктурованих PDF-документів на структуровану, персистентну базу знань.

Архітектура конвеєра (pipeline) обробки, трансформації та індексування інформаційних матеріалів кафедри штучного інтелекту має чітку послідовну структуру. Графічно цей інженерний контур представлено у вигляді алгоритму обробки контенту на рисунку 2.1.

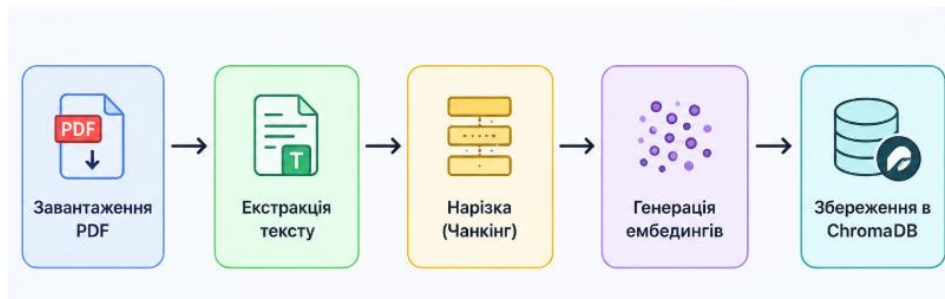


Рисунок 2.1 – Концептуальний алгоритм обробки та індексування текстової документації в ядрі RAG

Як показано на рисунку 2.1, початковим етапом функціонування системи є асинхронне завантаження PDF-файлів через адміністративну панель системи. Сирий текстовий потік, отриманий після парсингу файлів, не може бути переданий до мовної моделі цілком через обмеження контекстного вікна та ризик деградації уваги нейромережі (needle in a haystack problem). Для вирішення цієї проблеми впроваджено оптимізовану стратегію сегментації тексту (Chunking).

Текстовий масив розбивається на логічно завершені блоки фіксованого розміру, де базове вікно сегментації становить 800 символів. Оскільки механічний розрив тексту на фіксованій довжині може призвести до втрати смислового зв'язку (наприклад, розірвати математичну формулу

розрахунку конкурсного бала або назву спеціальності), алгоритм використовує стратегію ковзного вікна з перекриттям (Overlap) розміром 150 символів. Це забезпечує збереження семантичного контексту між сусідніми чанками та гарантує цілісність інформації на межах розриву сторінок першоджерела.

Наступним архітектурним рівнем є математичне кодування текстових сегментів. Кожен чанк обробляється моделлю ембедингів (Embedding Model), яка відображає символічну інформацію у щільні вектори дійсних чисел фіксованої багатовимірної розмірності. Отримані числові структури кодують приховані значеннєві ознаки тексту, що дозволяє системі надалі порівнювати документи не за точним збігом слів, а за семантичним змістом.

Для збереження згенерованих векторів спроектовано контур векторного сховища на базі бази даних ChromaDB. Замість використання нестабільних інкапсульованих хмарних рішень, було реалізовано персистентну локальну архітектуру. Ініціалізація сховища виконується через персистентний клієнт `chromadb.PersistentClient`, який жорстко фіксує шлях збереження векторних індексів на сервері у директорію `./chroma_db`. Усередині сховища реєструється та ізолюється виділена колекція `nure_ai_docs`.

Критично важливим інженерним розширенням розробленої архітектури RAG є впровадження механізму збагачення векторів структурними метаданими. Разом із числовим вектором кожного текстового чанку в колекцію ChromaDB записується словник метаданих, що містить ідентифікатор та унікальну назву вихідного файлу (filename). Це дозволило реалізувати підсистему пре-фільтрації (Metadata Filtering) на рівні векторної бази даних.

Коли користувач або адміністратор взаємодіє з інтелектуальним модулем, система виконує наступний алгоритм:

- запит користувача трансформується у вектор за допомогою тієї ж моделі ембедингів, що використовувалася при індексації;

- формується умовна конструкція фільтрації;
- chroma-клієнт виконує високоефективний пошук найближчих сусідів, відсікаючи нерелевантні джерела на етапі сканування бази знань;
- з колекції вилучається фіксована кількість N найбільш схожих за змістом текстових фрагментів ($N=5$).

Знайдений контекст асинхронно ін'єктується у системну інструкцію (System Prompt), яка передається до генеративного віртуального ШІ-агента. Мовна модель отримує жорстке розпорядження формувати відповідь або генерувати варіанти тестових питань, спираючись виключно на надані першоджерела кафедри. Такий двоконтурний підхід забезпечує абсолютну фактологічну точність системи, високу швидкість відгуку API та гнучкість управління знаннями без необхідності обчислювально затратного перенавчання (fine-tuning) нейромереж.

2.4 Розробка відмовостійкого гібридного алгоритму каскадної маршрутизації ШІ-моделей (Fallback)

При проєктуванні інтелектуальних систем, критично залежних від зовнішніх хмарних провайдерів штучного інтелекту, одним із найважливіших архітектурних завдань є забезпечення безвідмовності (fault tolerance) та високої доступності (high availability) сервісів. У розробленій системі інтелектуальні функції генерації тестових завдань та семантичного аналізу контексту RAG покладені на моделі сімейства Google Gemini API. Однак пряма експлуатація комерційних хмарних API у реальних освітніх екосистемах супроводжується ризиками інфраструктурної нестабільності: виникненням мережових збоїв, перевантаженням серверів провайдера (HTTP помилка 503 Service Unavailable), вичерпанням квот на безкоштовні запити (HTTP помилка 429 Rate Limit Exceeded) або спонтанними порушеннями структурної цілісності вихідних даних.

Для нівелювання цих ризиків у межах дипломного проєкту було розроблено та програмно реалізовано унікальний відмовостійкий каскадний алгоритм маршрутизації III-запитів (Fallback-механізм). Було реалізовано пряму REST-взаємодію з Google REST API за допомогою асинхронного HTTP-клієнта `httpx`. Це дозволило отримати повний низькорівневий контроль над структурою тіла запиту, HTTP-заголовками та передачею параметра `response_mime_type="application/json"`, що є обов'язковим для стабільного парсингу згенерованих питань у реляційній моделі даних [8].

2.5 Архітектура безпеки та розмежування прав доступу користувачів

Розробка корпоративних інформаційних систем для закладів вищої освіти вимагає імплементації надійних механізмів захисту даних. Оскільки розроблена система передбачає спільну роботу кількох адміністраторів (різних викладачів кафедри), виникла гостра інженерна необхідність у розмежуванні прав доступу до створеного контенту.

Для розв'язання цієї задачі архітектуру системи було доповнено моделлю управління доступом на основі ролей та авторства (Role-Based Access Control із елементами Attribute-Based Access Control). Базова логіка цієї моделі полягає у тому, що кожен публічний захід або гейміфікований тест (Quiz) у базі даних отримує обов'язковий атрибут прив'язки до свого творця – поле `author_email` або `creator_id`.

Архітектура безпеки спроектована таким чином, що система розрізняє два рівні авторизації:

- рівень глобального читання (Read-Only): будь-який авторизований адміністратор має право переглядати загальний дашборд аналітики, читати перелік усіх заходів кафедри та переглядати статистику проходження чужих тестів. Це необхідно для забезпечення прозорості робочих процесів на кафедрі;

– рівень локального редагування (Write/Update/Delete): право на модифікацію сутності (редагування тексту питань, зміну правильних варіантів відповідей або видалення тесту) надається виключно тому користувачу, чий унікальний ідентифікатор (email) збігається з ідентифікатором автора цієї сутності в реляційній базі даних.

Для обміну даними про сесію користувача між клієнтським React-додатком та FastAPI-сервером обрано стандарт JSON Web Tokens (JWT). Використання JWT дозволяє системі працювати за принципом Stateless (без збереження стану сесії на сервері), що знижує навантаження на базу даних та відповідає сучасним RESTful архітектурам [9].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Реалізація серверної логіки (Backend) та оптимізація роботи з даними

Серверна частина (Backend) інтелектуальної системи розроблена з використанням мікрофреймворку FastAPI на мові Python. Основним завданням серверної логіки є обробка HTTP-запитів від клієнтського застосунку, управління з'єднаннями з реляційною та векторною базами даних, а також координація роботи інтелектуальних агентів генерації контенту.

3.1.1 Програмування асинхронних API-маршрутів (Endpoints) та lifespan-менеджменту в FastAPI

Сучасні вебдодатки, особливо ті, що інтенсивно використовують бази даних та зовнішні API штучного інтелекту, вимагають ефективного управління життєвим циклом (lifecycle) серверного процесу. У розробленій системі керування ресурсами при запуску та зупинці сервера реалізовано за допомогою механізму lifespan контекстних менеджерів, вбудованого у FastAPI [8]. Такий підхід забезпечує автоматичну ініціалізацію необхідних компонентів під час запуску застосунку та їх коректне завершення після зупинки сервера. Це дозволяє уникнути витоків ресурсів, підвищує стабільність роботи системи та спрощує підтримку програмного коду.

В нашому випадку цей підхід дозволяє гарантовано ініціалізувати пули з'єднань з PostgreSQL та векторною базою ChromaDB до того, як сервер почне приймати перші клієнтські запити. Фрагмент коду, що реалізує життєвий цикл та базову конфігурацію застосунку, наведено на лістингу 3.1.

Лістинг 3.1 – Ініціалізація сервісів та налаштування FastAPI додатка

```

from dotenv import load_dotenv
load_dotenv()

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
from database import init_db
from vector_db import init_vector_db
from fastapi.staticfiles import StaticFiles
import os

from routes import user_routes, event_routes, ai_routes,
quiz_routes

@asynccontextmanager
async def lifespan(app: FastAPI):
    print("Initializing services...")
    await init_db()
    init_vector_db()
    yield

app = FastAPI(
    title="AI Event Support API",
    version="1.0.0",
    lifespan=lifespan
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

```

Як видно з лістингу 3.1, функція `lifespan` забезпечує строгу послідовність запуску підсистем. Асинхронний виклик `await init_db()` створює необхідні таблиці в PostgreSQL (якщо вони відсутні) на основі моделей `SQLModel`, описаних у розділі 2. Виклик `init_vector_db()` гарантує готовність векторного простору до виконання семантичного пошуку.

Для забезпечення маршрутизації та логічного розділення бізнес-логіки системи (Separation of Concerns) код API було розбито на незалежні модулі-роутери (Routers): `user_routes`, `event_routes`, `ai_routes` та `quiz_routes`. Всі роутери підключаються до основного екземпляра додатка за допомогою методу `app.include_router()`.

Окрему увагу приділено оптимізації маршрутів підсистеми управління заходами (Event Manager). На лістингу 3.2 наведено приклад оптимізованого асинхронного маршруту для отримання списку подій.

Лістинг 3.2 – Реалізація асинхронного маршруту отримання списку заходів

```
@router.get("/")
async def get_all_events(session: AsyncSession = Depends(get_session)):
    try:
        statement =
select(Event).order_by(Event.start_time)
        results = await session.exec(statement)
        events_list = results.all()
        return events_list
    except Exception as e:
        print(f"[GET EVENTS] Помилка отримання подій:
{str(e)}")
        raise HTTPException(status_code=500, detail=str(e))
```

Такий підхід до програмування маршрутів забезпечує неблокуючу обробку (non-blocking I/O) паралельних запитів. Завдяки ін'єкції

залежностей (Depends(get_session)) фреймворк FastAPI автоматично керує відкриттям та закриттям сесій бази даних для кожного окремого HTTP-запиту, що унеможливорює витoki пам'яті (memory leaks) та блокування з'єднань під час пікових навантажень [10].

3.1.2 Програмний інтерфейс взаємодії з ChromaDB

Для реалізації аналітичного контуру архітектури RAG було створено програмний інтерфейс взаємодії з векторною базою даних ChromaDB. Відповідну логіку інкапсульовано у файлі vector_db.py.

Враховуючи необхідність автономної роботи системи, було використано патерн персистентного клієнта (PersistentClient), який зберігає векторні індекси локально у файловій системі сервера. Програмну реалізацію функції ініціалізації та виконання векторного пошуку наведено на лістингу 3.3.

Лістинг 3.3 – Реалізація ініціалізації і векторного пошуку в ChromaDB

```
import chromadb
from typing import List

_collection = None

def init_vector_db():
    global _collection
    if _collection is None:
        client = chromadb.PersistentClient(path="./chroma_db")
        _collection = client.get_or_create_collection(name="nure_ai_docs")
        print("База даних ChromaDB успішно ініціалізована!")
    return _collection
```

Продовження лістингу 3.3

```
def get_collection():
    global _collection
    if _collection is None:
        init_vector_db()
    return _collection

def search_chroma(filenamees: List[str], query_text: str):
    coll = get_collection()
    try:
        where_clause = {"filename": {"$in": filenamees}} if
filenamees else None
        results = coll.query(
            query_texts=[query_text],
            where=where_clause,
            n_results=3
        )
        if results and results.get('documents') and
results['documents'][0]:
            return "\n".join(results['documents'][0])
        return ""
    except Exception as e:
        print(f" Помилка пошуку в ChromaDB: {e}")
        return ""
```

Ключовим інженерним рішенням у наведеному коді (лістинг 3.2) є реалізація динамічної метафільтрації за допомогою змінної `where_clause`. Оператор `{"$in": filenamees}` дозволяє ШІ-агенту жорстко обмежувати зону пошуку контексту лише тими документами, які вибрав користувач (наприклад, шукати відповідь виключно у «Правилах вступу», ігноруючи іншу документацію кафедри).

Також було проведено RAG-тюнінг: параметр `n_results` збільшено до 3, що гарантує надання мовній моделі достатнього обсягу контексту для

вирішення складних завдань (наприклад, генерації масиву тестів), унеможливаючи втрату критично важливої інформації [11].

3.2 Програмна реалізація ШІ-агента генерації та валідації контенту

Ядром інтелектуальної системи підтримки публічних заходів кафедри є автономний ШІ-агент. Його головна функція полягає у семантичному аналізі вхідних запитів, вилученні контексту з бази знань (RAG) та динамічній генерації гейміфікованих завдань (квізів). Реалізація цього модуля вимагала відмови від стандартних підходів до промптингу на користь жорстко типізованих HTTP-запитів до Google REST API.

3.2.1 Логіка трирежимного генератора тестів (за PDF-контекстом, за вільною темою, масове збереження)

Для задоволення різних освітніх сценаріїв кафедри штучного інтелекту, архітектуру генератора було спроєктовано за принципом поліморфної обробки запитів. Алгоритм підтримує три незалежні режими роботи, кожен з яких використовує власну стратегію формування контексту та системної інструкції (System Prompt). Маршрутизація між режимами здійснюється на рівні ендпоінту `/ai/generate-quiz` через аналіз тіла POST-запиту.

Режим 1: генерація за суворим PDF-контекстом (Strict RAG Mode). Цей режим активується, коли адміністратор обирає конкретні файли (наприклад, регламент олімпіади) як джерело для тесту. Логіка роботи цього режиму є найбільш ресурсомісткою, оскільки вона задіює векторну базу даних ChromaDB. Алгоритм виконує наступні кроки:

- клієнт передає масив ідентифікаторів файлів `filenames`;
- сервер викликає функцію `search_chroma`, яка обчислює ембединги запиту та вилучає 5 найбільш релевантних текстових чанків;

– знайдений текст «зшивається» у єдиний блок контексту і додається до базового промпту з обов'язковою директивою: «Генеруй питання ВИКЛЮЧНО на основі наступного тексту. Якщо текст не містить відповіді, не вигадуй її».

Фрагмент програмної реалізації збірки контексту наведено на лістингу 3.4.

Лістинг 3.4 – Формування контексту для суворого RAG-режиму

```
def generate_rag_prompt(request: QuizGenRequest):
    """
    РЕЖИМ 1: Генерація ТІЛЬКИ на основі контексту з вибраних
    PDF-файлів (RAG)
    """
    print(f"[RAG DEBUG] Шукаємо контекст для файлів:
    {request filenames} за запитом: '{request.quiz_title}'")

    context = search_chroma(filenames=request.filenames,
    query_text=request.quiz_title)

    prompt = (
        "Ти – суворий методист кафедри Штучного Інтелекту
        ХНУРЕ.\n"
        "Твоя задача: створити академічний тест,
        використовуючи ТІЛЬКИ наданий КОНТЕКСТ З PDF ФАЙЛІВ.\n"
        "ПРАВИЛА:\n"
        "- Запитання та варіанти відповідей повинні
        базуватися ВИКЛЮЧНО на фактах із наданого тексту.\n"
        "- Не вигадуй нічого від себе, не використовуй
        сторонні знання про світ чи інтернет.\n"
        "- Якщо у контексті написано про регламент практики,
        питання мають бути про практику. Якщо про нейромережі – про
        нейромережі.\n\n")
```

Продовження лістингу 3.4

```
f"КОНТЕКСТ З PDF ФАЙЛІВ ДЛЯ АНАЛІЗУ:\n{context}\n\n"
f"ЗАВДАННЯ: Створи {request.count} унікальних питань
для тесту.\n\n"
" СУВОРА ВИМОГА ДО ФОРМАТУ ВІДПОВІДІ:\n"
"Поверни відповідь СУВОРО у форматі JSON об'єкта із
єдиним верхнім ключем 'questions'.\n"
"Кожне питання всередині масиву повинно мати такі
назви ключів:\n"
"{\n"
"  'question_text': 'Текст запитання',\n"
"  'option_a': 'Варіант А',\n"
"  'option_b': 'Варіант В',\n"
"  'option_c': 'Варіант С',\n"
"  'option_d': 'Варіант D',\n"
"  'correct_option': 'Велика латинська літера (А,
В, С або D)'\n"
"}\n"
"Поверни ТІЛЬКИ чистий JSON. Не додавай розмітку
Markdown на кшталт ```json."
)
return prompt
```

Режим 2: генерація за вільною темою (Free-topic Mode). Цей режим використовується для створення загальних квізів (наприклад, «Історія розвитку ШІ» для Дня відкритих дверей), коли специфічний контекст кафедри не потрібен. Якщо масив filenames порожній, бекенд ігнорує виклик ChromaDB (щоб заощадити ресурси сервера) і формує прямий промпт до Gemini, використовуючи лише вбудовані параметричні знання моделі.

Режим 3: масове збереження (Bulk Save Mode). Оскільки генерація великих тестів (15 – 20 питань) за допомогою LLM займає певний час, у системі реалізовано механізм попереднього перегляду (Preview).

Згенерований III масив JSON-об'єктів спочатку повертається на фронтенд, де адміністратор має змогу відредагувати питання або змінити правильні варіанти. Лише після візуальної перевірки натискається кнопка «Масового збереження».

На цьому етапі клієнт відправляє валідований масив на ендпоінт бекенду, де сервер ітеративно створює екземпляри моделі QuizQuestion та фіксує їх у PostgreSQL за допомогою асинхронної сесії `session.add_all()`.

Завдяки такій трирежимній архітектурі, система забезпечує максимальну гнучкість: від жорсткого тестування за нормативними документами до творчої гейміфікації публічних лекцій [12].

3.2.2 Алгоритми жорсткої структурної валідації JSON-відповідей від мовних моделей

Використання великих мовних моделей (LLM) у промислових бекенд-системах супроводжується фундаментальною проблемою стохастичності. На відміну від детермінованих алгоритмів, генеративні моделі схильні до додавання непередбачуваних текстових артефактів: вступних фраз (наприклад, «Ось ваші згенеровані питання...»), Markdown-розмітки або порушення синтаксису кінцевого формату. Оскільки клієнтський React-додаток та реляційна база даних PostgreSQL вимагають суворо типізованих об'єктів, пряма передача сирої відповіді від III до бази даних є інженерно неприпустимою.

Для розв'язання цієї проблеми в системі реалізовано багаторівневий алгоритм примусової структуризації та жорсткої валідації даних, який працює у три етапи: пре-кондиціювання, санітизація (очищення) та об'єктна валідація.

Етап 1: пре-кондиціювання на рівні API та Промпту.

Під час формування HTTP-запиту до Google Gemini REST API алгоритм бекенду застосовує подвійний захист. По-перше, на рівні

параметрів API (payload) примусово встановлюється конфігурація `generationConfig: {"responseMimeType": "application/json"}`. По-друге, системний промпт розширюється жорсткою схемою очікуваного результату (JSON Schema). Фрагмент інструкції наведено на лістингу 3.5.

Лістинг 3.5 – Фрагмент системної інструкції для контролю формату виводу

```
system_instruction = """
Ти — суворий генератор тестових питань.

Твоя єдина мета — повернути масив JSON-об'єктів. Жодного
додаткового тексту.

Формат відповіді ПОВИНЕН строго відповідати цій структурі:
[
  {
    "question_text": "Текст питання?",
    "options": ["А. Варіант 1", "Б. Варіант 2", "В. Варіант
3", "Г. Варіант 4"],
    "correct_option": "А"
  }
]
"""
```

Етап 2: санітизація (очищення) сирої відповіді.

Незважаючи на встановлені параметри, під час високого навантаження модель `gemini-2.5-flash` іноді повертає JSON, загорнутий у Markdown-блоки. Для запобігання падінню вбудованого в Python парсера `json.loads()`, у системі розроблено алгоритм первинної текстової санітизації з використанням методів маніпуляції рядками, який автоматично видаляє артефакти розмітки до початку обробки.

Етап 3: жорстка об'єктна валідація (Pydantic).

Вирішальним етапом алгоритму є структурна перевірка очищеного тексту за допомогою бібліотеки Pydantic. Бекенд намагається серіалізувати

текстовий JSON у строгі моделі даних (Data Transfer Objects), які вимагають наявності всіх необхідних полів та правильних типів даних (наприклад, щоб options гарантовано був масивом з чотирьох рядків). Програмна реалізація цього процесу наведена на лістингу 3.6.

Лістинг 3.6 – Алгоритм очищення та Pydantic-валідації відповіді ШІ

```
import json
from pydantic import BaseModel, ValidationError
from typing import List

# DTO схема для валідації результату
class QuizQuestionSchema(BaseModel):
    question_text: str
    options: List[str]
    correct_option: str

def parse_and_validate_ai_response(raw_text: str) ->
List[QuizQuestionSchema]:
    # Етап 2: Санітизація Markdown-артефактів
    cleaned_text = raw_text.strip()
    if cleaned_text.startswith("`json`"):
        cleaned_text = cleaned_text[7:]
    if cleaned_text.startswith("`"):
        cleaned_text = cleaned_text[3:]
    if cleaned_text.endswith("`"):
        cleaned_text = cleaned_text[:-3]
    cleaned_text = cleaned_text.strip()

    # Етап 3: Парсинг та жорстка Pydantic валідація
    try:
        json_data = json.loads(cleaned_text)
        validated_data = [QuizQuestionSchema(**item) for
item in json_data]
        return validated_data
```

Продовження лістингу 3.6

```
except (json.JSONDecodeError, ValidationError) as e:
    # Логування помилки структури для активації
    Fallback-механізму
    print(f"[ERROR]          Валідація          ШІ-відповіді
    провалилася: {e}")
    raise ValueError("Invalid JSON structure from AI")
```

Як видно з лістингу 3.6, якщо мовна модель пропускає поле або генерує некоректний тип даних, бібліотека генерує виключення `ValidationError`. Це виключення перехоплюється блоком `try...except`, який генерує `ValueError`. Саме ця навмисна генерація помилки є тригером для активації каскадного алгоритму `Fallback`. Завдяки цій архітектурі, при отриманні «поламаного» JSON від швидкої моделі `gemini-2.5-flash`, сервер автоматично перенаправляє той самий промпт до потужнішої `gemini-1.5-pro` або локальної `Ollama`, гарантуючи, що база даних `PostgreSQL` та клієнтський застосунок отримають валідовану та безпечну структуру даних [13].

3.3 Розробка клієнтського інтерфейсу (Frontend) та інструментів інтерактивної аналітики

Клієнтська частина (Frontend) інтелектуальної системи розроблена у форматі вебдодатка (Single Page Application, SPA) з використанням бібліотеки `React`. Головним завданням цього етапу було створення інтуїтивно зрозумілого, швидкого та адаптивного інтерфейсу, який би ефективно обслуговував дві цільові аудиторії: адміністраторів (викладачів кафедри) та кінцевих користувачів (студентів, абітурієнтів). Головна сторінка усіх подій зображена на рисунку 3.1.

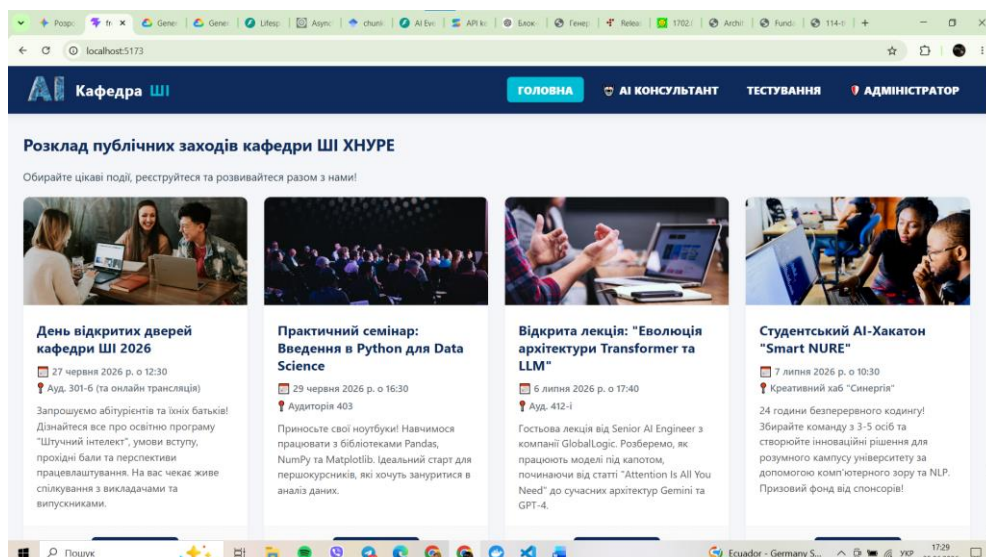


Рисунок 3.1 – Головна сторінка інтерфейсу користувача

3.3.1. Архітектура SPA-додатка та концепція Clean UI

Для стилізації компонентів додатка було обрано фреймворк Tailwind CSS. На відміну від традиційного підходу з використанням глобальних каскадних таблиць стилів, Tailwind використовує концепцію utility-first, що дозволило інкапсулювати візуальні властивості безпосередньо всередині React-компонентів, уникаючи конфліктів класів.

Основою візуального проєктування стала концепція Clean UI (чистий інтерфейс) [14]. Згідно з цією парадигмою, з інтерфейсу було видалено всі надлишкові візуальні елементи, фокус користувача спрямовано виключно на контент (тексти документів, питання квізів, реєстраційні форми). Для підсистеми управління заходами (Event Manager) розроблено оптимізовані форми вводу (Sticky-форми), які залишаються в полі зору користувача при прокручуванні довгих списків або регламентів.

Маршрутизація всередині SPA реалізована за допомогою механізму умовного рендерингу в головному компоненті App.jsx. Це забезпечує миттєве перемикавання між екранами без перезавантаження сторінки з боку сервера. На рисунку 3.2 наведено головний екран підсистеми Event Manager.

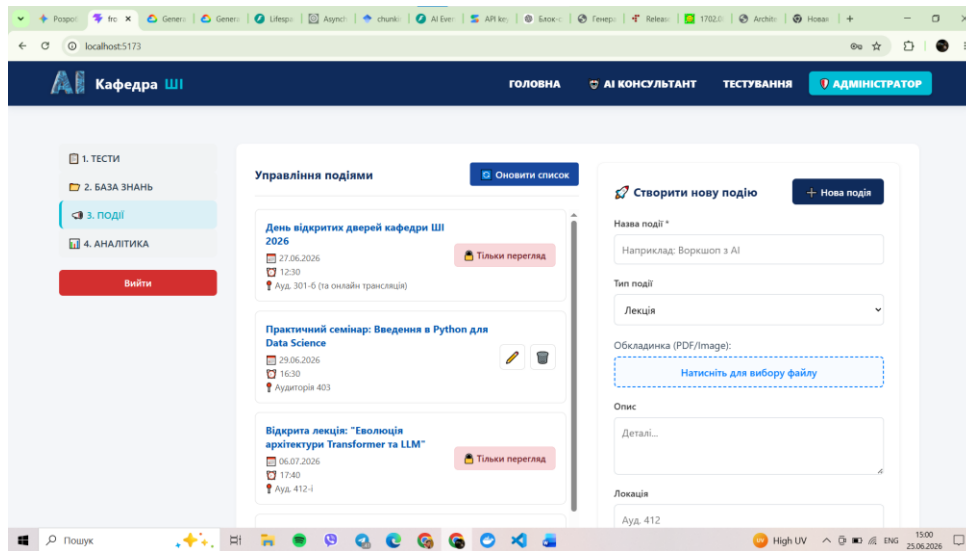


Рисунок 3.2 – Інтерфейс підсистеми управління публічними заходами (Event Manager)

3.3.2. Розробка дашборду аналітики подій

Для забезпечення керівництва кафедри своєчасною інформацією щодо ефективності проведення заходів, у системі розроблено аналітичний дашборд. Візуалізація статистичних даних реалізована за допомогою бібліотеки Recharts, яка базується на D3.js та надає декларативні React-компоненти для побудови складних діаграм.

Дашборд агрегує масиви даних, отримані з бекенду (FastAPI), та трансформує їх у графічні репрезентації:

- гістограми (Bar Charts): використовуються для відображення динаміки реєстрацій на різні заходи (наприклад, порівняння кількості учасників Хакатону та Дня відкритих дверей). для роботи з великими масивами даних впроваджено адаптивний скрол всередині графіків;
- кругові діаграми (Pie Charts): застосовуються для аналізу розподілу заходів за типами. Діаграми оснащено кастомними тултипами (Tooltips) з українською локалізацією.

Зовнішній вигляд розробленого дашборду аналітики представлено на рисунку 3.3.

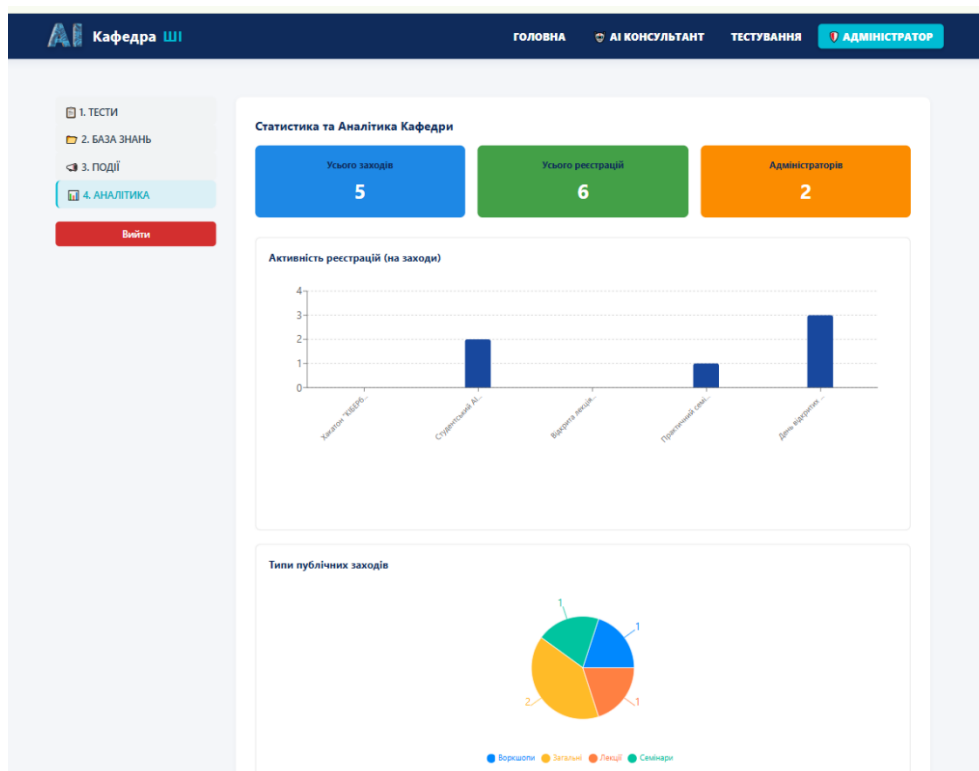


Рисунок 3.3 – Інтерактивний дашборд аналітики заходів кафедри ШІ

3.3.3. Інтерфейси модуля гейміфікації та керування ШІ-генерацією

Найбільш складною з точки зору UX/UI дизайну стала розробка інтерфейсу для взаємодії з ШІ-агентом генерації квізів. Оскільки процес генерації з використанням RAG-архітектури та векторного пошуку вимагає гнучких налаштувань, інтерфейс адміністратора було розділено на дві робочі зони: панель конфігурації (де обирається один із трьох режимів: за контекстом PDF, за вільною темою або вручну), зону попереднього перегляду (Preview). Інтерфейс вибору режиму створення квізу наведено на рисунку 3.4.

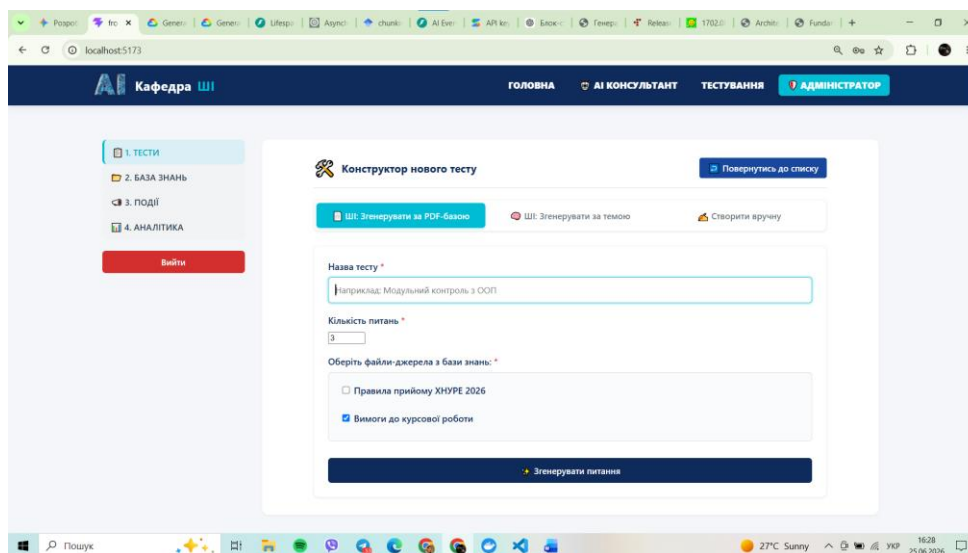


Рисунок 3.4 – Інтерфейс вибору режиму створення квізу

Після того, як ШІ-модель (Gemini) повертає згенерований та валідований масив JSON-об'єктів, фронтенд динамічно рендерить їх у вигляді інтерактивних карток запитань. Кожна картка чітко відображає текст питання, чотири варіанти відповіді (A, B, C, D) та підсвічує правильний варіант.

Саме на цьому екрані впроваджено функцію «Масового збереження» (Bulk Save). Адміністратор може візуально перевірити коректність роботи ШІ, за необхідності відредагувати текст прямо в картці, і лише після цього відправити весь масив у базу даних PostgreSQL одним POST-запитом [15].

Для кінцевого користувача (студента) розроблено мінімалістичний інтерфейс проходження тесту, який відображає на екрані лише одне питання одночасно (Pagination), забезпечуючи високу концентрацію уваги та зменшуючи когнітивне навантаження під час проходження тестування. Після завершення тестування система автоматично перевіряє правильність відповідей, миттєво обчислює підсумковий результат і відображає його у відсотковому співвідношенні. Такий підхід дозволяє швидко оцінити рівень засвоєння матеріалу.

3.4 Програмна реалізація механізмів автентифікації та захисту маршрутів

Імплементація архітектури безпеки вимагала синхронних змін як на серверній, так і на клієнтській частинах додатка. На рівні бекенду захист API-маршрутів реалізовано за допомогою вбудованої у FastAPI системи залежностей (Dependency Injection) та схеми безпеки OAuth2PasswordBearer.

Кожен HTTP-запит, який потребує модифікації даних, проходить через проміжний шар перевірки (Middleware). Функція-залежність `get_current_user` вилучає JWT-токен із заголовка `Authorization`, розшифровує його, валідує термін дії та витягує email поточного користувача. Фрагмент логіки захисту ендпоінту редагування тесту наведено на лістингу 3.7.

Лістинг 3.7 – Реалізація перевірки прав авторства у FastAPI

```
@router.put("/quizzes/{quiz_id}")
async def update_quiz(
    quiz_id: int,
    update_data: QuizUpdateSchema,
    session: AsyncSession = Depends(get_session),
    current_user: User = Depends(get_current_user)  #
    Валідація JWT
):
    # Отримання тесту з бази даних
    quiz = await session.get(Quiz, quiz_id)
    if not quiz:
        raise HTTPException(status_code=404, detail="Тест
    не знайдено")

    # КЛЮЧОВИЙ БЛОК БЕЗПЕКИ: Перевірка авторства
    if quiz.author_email != current_user.email:
        raise HTTPException(
            status_code=403,
```

Продовження лістингу 3.7

```
        detail="Відмовлено в доступі. Ви не є автором  
цього тесту."  
    )  
  
    # Логіка оновлення...  
    return {"status": "success", "message": "Тест  
оновлено"}
```

Як видно з лістингу 3.7, спроба адміністратора відредагувати чужий тест блокується сервером із поверненням HTTP-статусу 403 Forbidden.

На стороні клієнтського інтерфейсу (Frontend) логіку безпеки імплементовано на рівні React-компонентів. Для забезпечення концепції Clean UI (зменшення візуального шуму), кнопки «Редагувати» та «Видалити» не просто видають помилку при натисканні, а фізично приховуються з DOM-дерева, якщо email авторизованого користувача (збережений у React Context) не збігається з author_email відмальованої картки. Це унеможливорює помилкові кліки та покращує загальний користувацький досвід (UX).

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

Основним завданням етапу тестування є перевірка функціональної придатності спроектованої системи та наочна демонстрація коректності роботи ключових модулів: інтелектуального чату (AI-асистента) та підсистеми генерації квізів. Експериментальні дослідження проводилися шляхом моделювання реальних сценаріїв користування (Use Cases) як з боку адміністратора (викладача), так і з боку кінцевого користувача (студента чи абітурієнта).

4.1. Тестування функціональності AI-чату

Інтелектуальний чат є основним інтерфейсом взаємодії користувачів із базою знань кафедри штучного інтелекту. Для оцінки його ефективності було проведено тестування здатності ШІ-агента знаходити специфічну інформацію у завантажених PDF-документах та синтезувати зрозумілу відповідь українською мовою.

Сценарій тестування: в адміністративній панелі системи до векторного сховища ChromaDB було завантажено внутрішній документ кафедри – «Правила прийому ХНУРЕ 2026». Документ містить різні правила, витяги з наказів та переліки різних документів.

Через клієнтський інтерфейс AI-чату було відправлено запит, сформульований природною мовою: «Як розраховується конкурсний бал?».

Як видно на рисунку 4.1, система миттєво обробила запит. Модуль RAG успішно виконав семантичний пошук по локальній базі даних, вилучив потрібні правила з регламенту і передав їх мовній моделі Gemini. Модель, у свою чергу, згенерувала чітку, структуровану відповідь, спираючись виключно на наданий контекст, уникнувши генерації загальної інформації з мережі Інтернет.

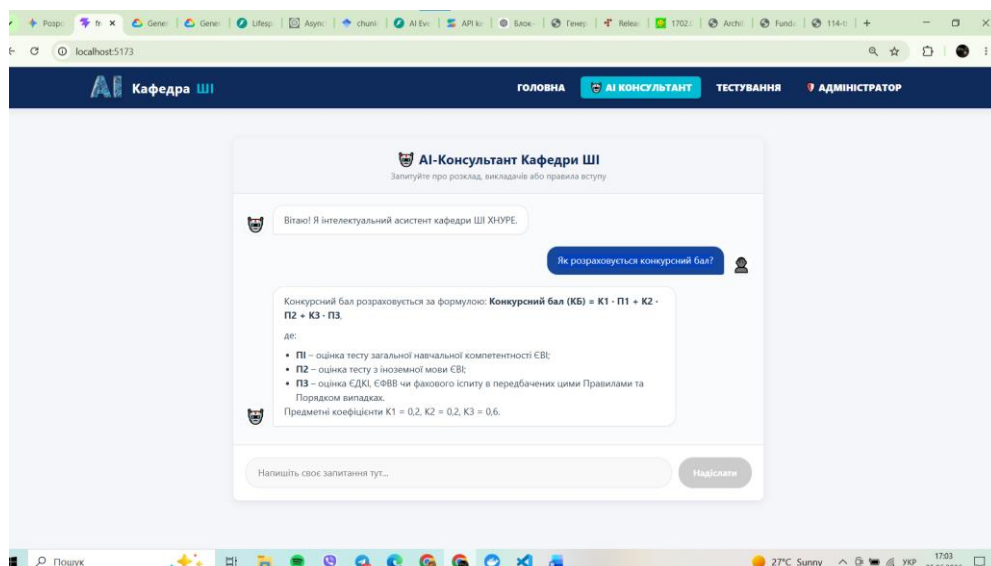


Рисунок 4.1 – Результат роботи AI-чату

У процесі тестування також було перевірено відмовостійкість архітектури на основі ще одного запиту щодо правил вступу, продемонстрованого на рисунку 4.2.

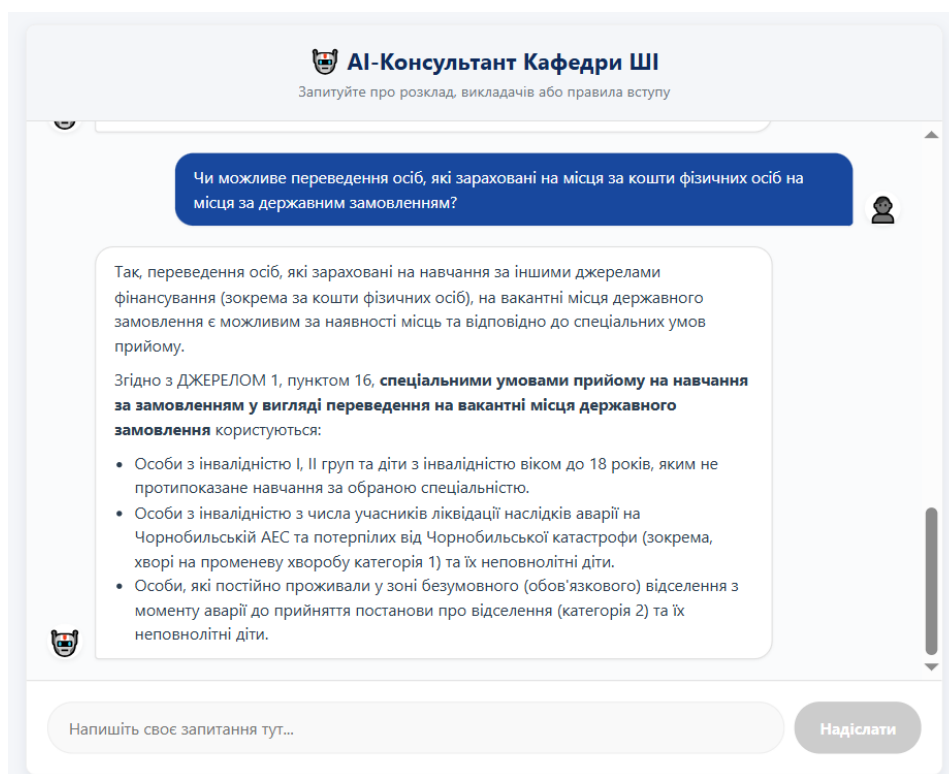
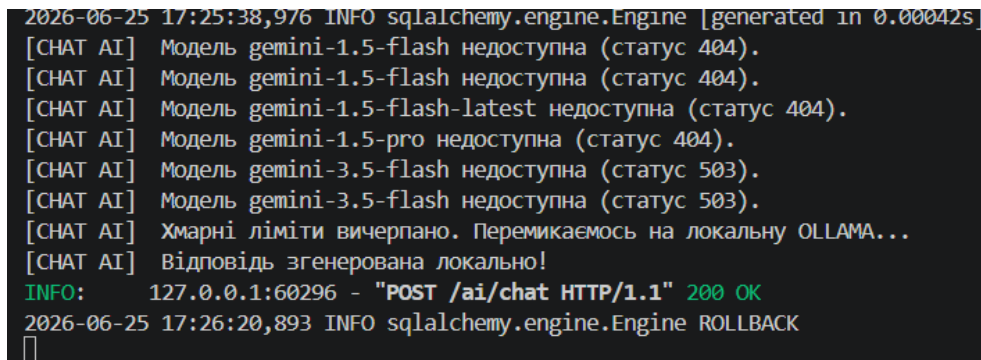


Рисунок 4.2 – Результат роботи AI-чату при перевантаженому сервері

Під час перевантаження серверів, клієнтський інтерфейс не видав системної помилки. Натомість, завдяки розробленому каскадному алгоритму (Fallback), запит був непомітно для користувача перенаправлений на резервну модель.

На рисунку 4.3 наведено консольний вивід серверної частини (FastAPI), що фіксує момент успішного перехоплення помилки та перемикання на резервну локальну модель.



```

2026-06-25 17:25:38,976 INFO sqlalchemy.engine.Engine [generated in 0.00042s]
[CHAT AI] Модель gemini-1.5-flash недоступна (статус 404).
[CHAT AI] Модель gemini-1.5-flash недоступна (статус 404).
[CHAT AI] Модель gemini-1.5-flash-latest недоступна (статус 404).
[CHAT AI] Модель gemini-1.5-pro недоступна (статус 404).
[CHAT AI] Модель gemini-3.5-flash недоступна (статус 503).
[CHAT AI] Модель gemini-3.5-flash недоступна (статус 503).
[CHAT AI] Хмарні ліміти вичерпано. Перемикаємось на локальну OLLAMA...
[CHAT AI] Відповідь згенерована локально!
INFO:      127.0.0.1:60296 - "POST /ai/chat HTTP/1.1" 200 OK
2026-06-25 17:26:20,893 INFO sqlalchemy.engine.Engine ROLLBACK

```

Рисунок 4.3 – Логування відпрацювання алгоритму відмовостійкості на стороні сервера

4.2. Оцінка точності генерації тестів та сценарії проходження квізів

Модуль гейміфікації покликаний підвищити рівень залученості учасників заходів. Тестування цього модуля складалося з двох етапів: перевірки зручності генерації питань адміністратором та оцінки користувацького досвіду (UX) при проходженні тесту студентом.

Етап 1: генерація тестів в адміністративній панелі.

Для тестування було використано режим генерації за вільною темою (Free-topic Mode). В адміністративній панелі було введено тему: «Історія розвитку штучного інтелекту», вказано кількість питань – 3, та обрано відповідний захід для прив'язки.

Після обробки запиту ШІ-моделлю, система успішно здійснила структурну валідацію отриманого JSON-масиву за допомогою бібліотеки Pydantic. Результат генерації було виведено в інтерфейс у вигляді інтерактивних карток попереднього перегляду (Preview Mode).

Як показано на рисунку 4.4, адміністратор отримав змогу візуально перевірити коректність запитань (наприклад, питання про тест Тюрінга або рік створення першого перцептрону), після чого тест був збережений у базу даних PostgreSQL.

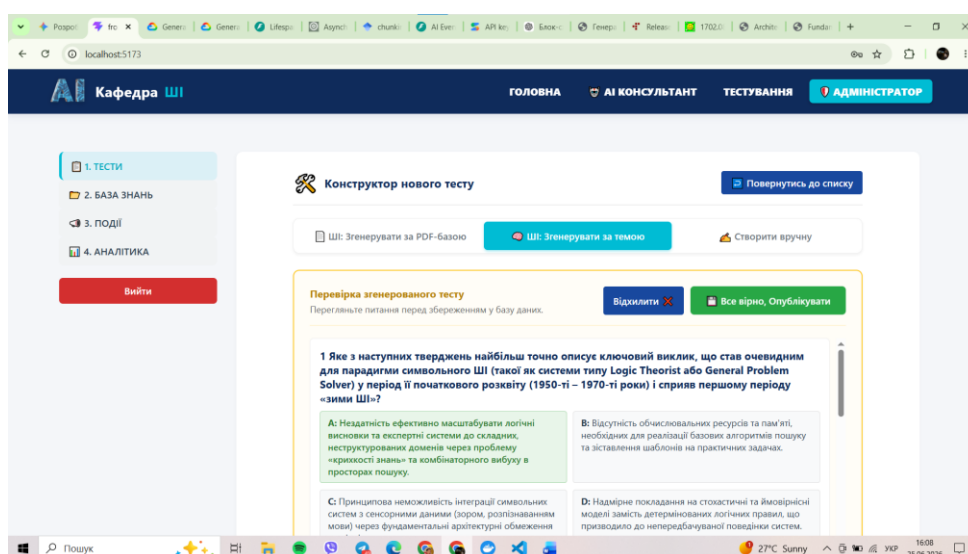


Рисунок 4.4 – Інтерфейс валідації та збереження згенерованих ШІ квізів

Етап 2: проходження тесту кінцевим користувачем.

Наступним кроком була перевірка процесу тестування з боку студента. Користувач, перейшовши за відповідним посиланням, потрапляє у мінімалістичний інтерфейс тестування. Кожне запитання представлене у вигляді окремого блоку з варіантами відповідей. Після вибору відповіді на запитання, сам варіант підсвічується кольором щоб користувач міг розуміти на які питання вже обрана відповідь, а на які ще залишилось обрати. Інтерфейс, про який зараз описується представлений на рисунку 4.5.

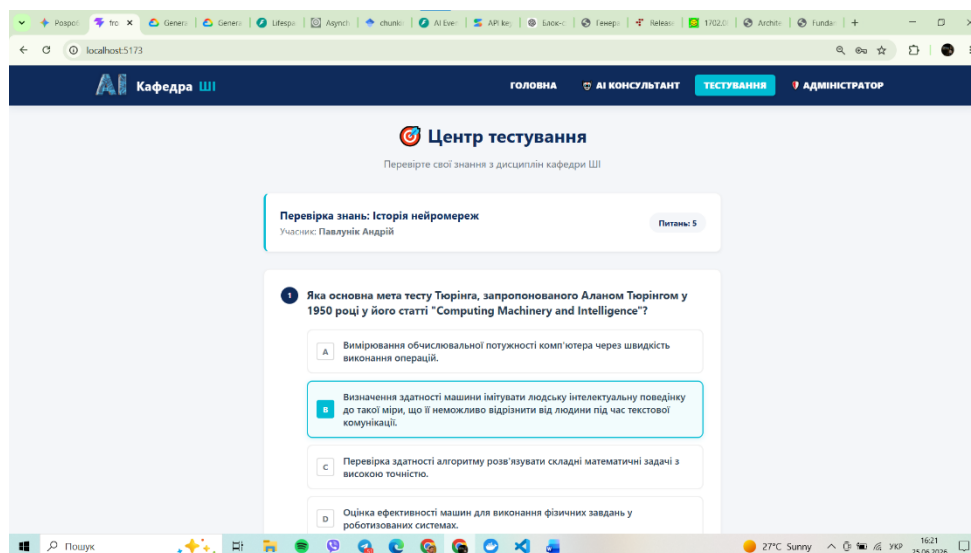


Рисунок 4.5 – Інтерфейс проходження квізу учасником

Після надання відповідей на всі запитання, клієнтський додаток відправляє масив обраних варіантів на сервер. Серверна логіка порівнює відповіді з еталонними значеннями в базі даних, розраховує фінальний бал і створює запис у таблиці QuizResult. Клієнту миттєво повертається екран з результатами (рисунок 4.6).

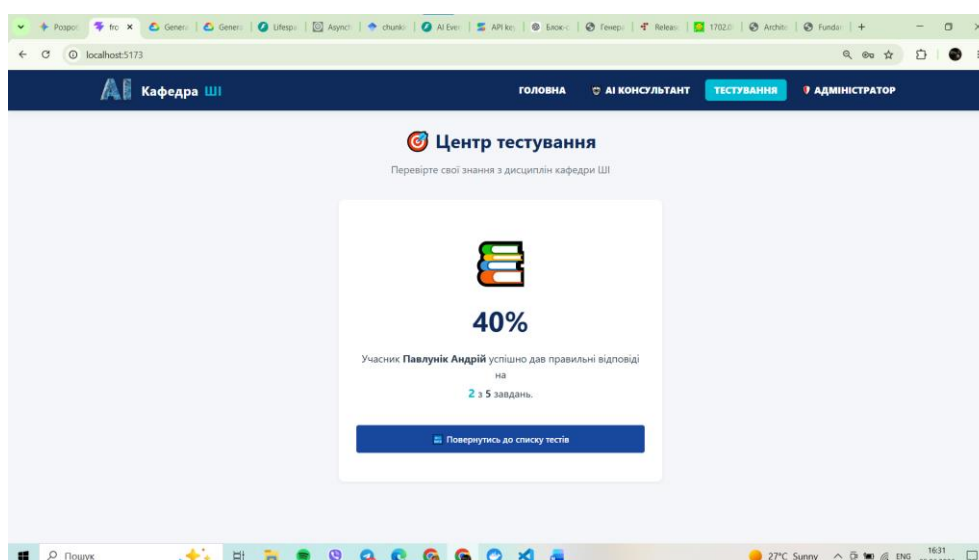


Рисунок 4.6 – Екран результатів після успішного проходження квізу студентом

Фінальним етапом тестування стала перевірка роботи модуля аналітики. Дані про пройдені студентами квізи та нові реєстрації на заходи миттєво синхронізуються із загальним дашбордом системи. Як було показано на рисунку 3.2, інтерактивні графіки бібліотеки Recharts коректно відмальовують динаміку активності, дозволяючи керівництву кафедри візуально оцінити успішність проведеного заходу.

Але окрім загальної статистики подій і тестів Кафедри ШІ, представленої в дашборді, також є можливість переглядати статистику тестувань безпосередньо серед учасників конкретного тесту. Адміністратор має можливість переглядати кількість набраних балів учасниками тестування, а також спостерігати загальну ситуацію щодо пройденого тестування в вигляді простої стовпчикової діаграми. До того ж в разі проявленого бажання або вимушеної зміни кількості запитань чи їх варіантів відповідей, адміністратор має можливість редагувати вже створений тест, тобто окремо кожне запитання тесту, відповіді на запитання квізу і навіть можна переобрати інший варіант правильним в тому чи іншому запитанні. Увесь цей функціонал продемонстровано на рисунку 4.7.

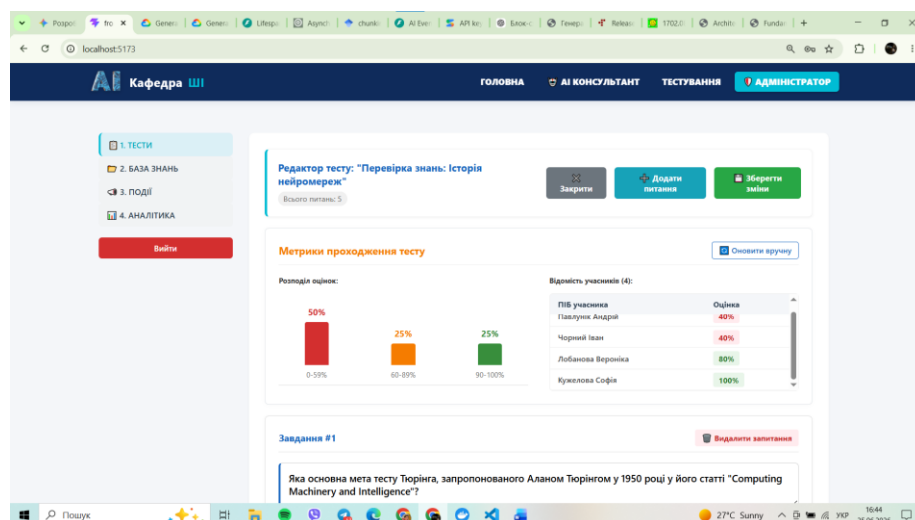


Рисунок 4.7 – Статистика тесту та можливість його редагування на панелі адміністратора

Загалом, проведене функціональне тестування підтверджує, що розроблена інформаційна система працює стабільно, всі інженерні підсистеми інтегровані коректно, а інтерфейс відповідає сучасним стандартам юзабіліті та задовольняє поставлені у роботі завдання [16].

4.3 Тестування системи розмежування прав доступу

Для перевірки коректності роботи механізмів авторизації та візуального приховування елементів управління було проведено тестування системи з використанням двох незалежних облікових записів адміністраторів: `admin_a@nure.ua` (Користувач А) та `admin_b@nure.ua` (Користувач Б).

Сценарій тестування: користувач А створив новий захід та згенерував для нього ШІ-тест. Після цього було здійснено вихід із системи (Logout) та повторну авторизацію під обліковим записом Користувача Б. Мета експерименту – перевірити, чи зможе Користувач Б змінити параметри тесту, створеного Користувачем А.

Результати експерименту: при переході на сторінку створеного заходу клієнтський React-додаток успішно завантажив дані з бекенду. Однак логіка умовного рендерингу (Conditional Rendering) розпізнала відмінність між значеннями `current_user.email` та `event.author_email`. У результаті інтерфейс динамічно приховавав панель керування тестом (кнопки редагування та видалення), залишивши доступними лише функції перегляду.

Додатково було перевірено, що обмеження не зводяться лише до приховування елементів інтерфейсу. У разі спроби звернення до API для редагування або видалення тесту сервер виконує повторну перевірку прав доступу на основі даних автентифікованого користувача. Якщо автор запиту не збігається з автором створеного ресурсу, сервер повертає повідомлення про відмову в доступі, що унеможливорює несанкціоновану зміну даних навіть у випадку модифікації клієнтської частини.

На рисунку 4.8 наведено порівняльний вигляд інтерфейсу для автора контенту та для стороннього адміністратора.

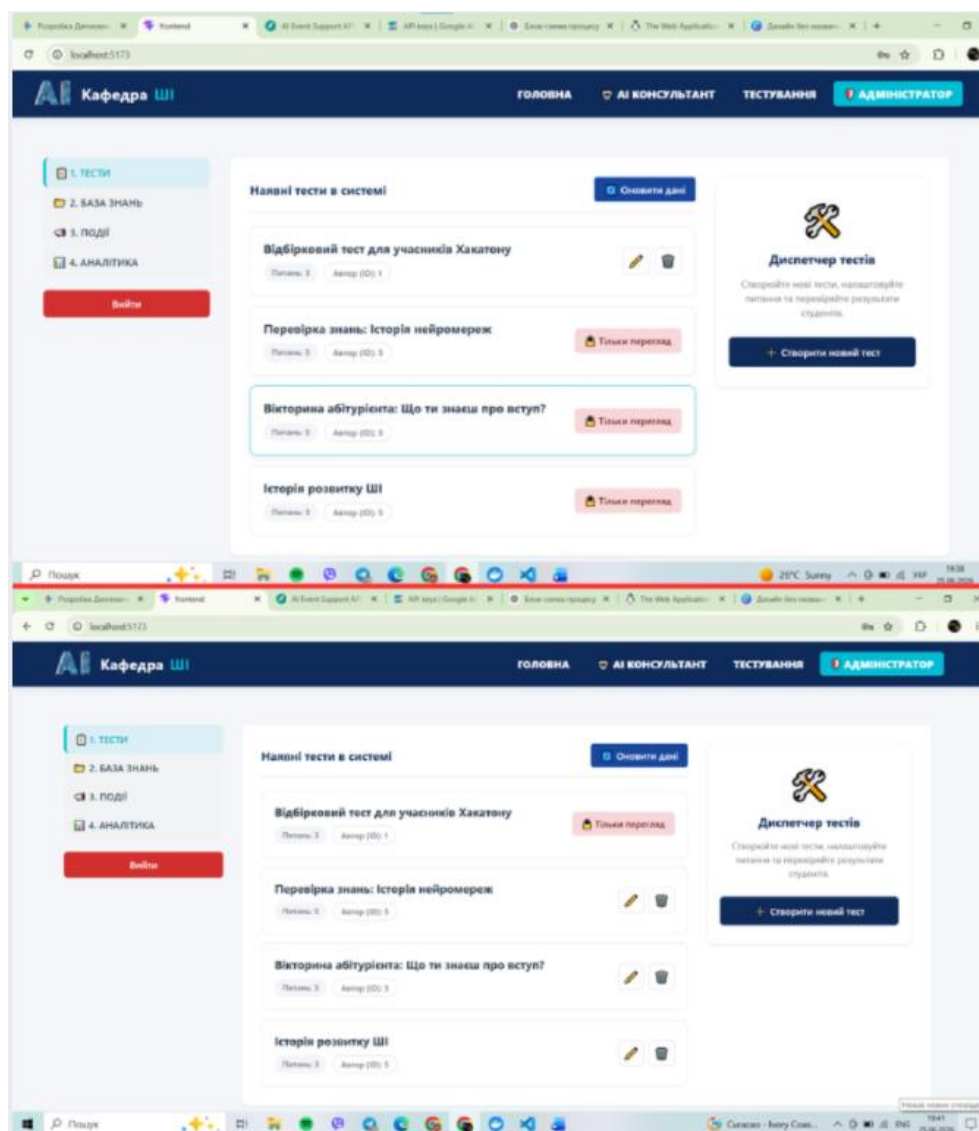


Рисунок 4.8 – Адаптація клієнтського інтерфейсу відповідно до прав доступу користувача

Таким чином, тестування підтвердило коректну реалізацію механізму розмежування прав доступу як на рівні користувацького інтерфейсу, так і на рівні серверної логіки. Це забезпечує цілісність даних, запобігає несанкціонованому редагуванню інформації та підвищує загальний рівень безпеки вебзастосунку.

ВИСНОВКИ

Отже, в результаті виконання кваліфікаційної роботи успішно вирішено актуальну науково-практичну задачу – спроектовано та програмно реалізовано інтелектуальну систему підтримки публічних заходів кафедри штучного інтелекту.

Розроблений клієнт-серверний вебдодаток забезпечує автоматизацію реєстрації учасників, надає їм цілодобову інтелектуальну підтримку на базі офіційної документації (RAG) та дозволяє генерувати гейміфіковані тести. Аналіз досягнутих показників свідчить про високу ефективність системи: автоматизація процесу генерації тестів за допомогою ШІ дозволила суттєво оптимізувати часові витрати викладачів у порівнянні з ручним створенням контенту. Водночас використання архітектури генерації з доповненням пошуку (RAG) на основі локального PDF-контексту забезпечило високу точність відповідей AI-чату та дозволило мінімізувати проблему фактологічних «галюцинацій», притаманну базовим мовним моделям, хоча й залишає простір для подальшого вдосконалення механізмів фільтрації контексту.

Співставлення виконаної розробки з вітчизняними та світовими аналогами (Eventbrite, Kahoot!, Moodle) підтверджує її конкурентні переваги та інноваційність. На відміну від класичних систем менеджменту подій, які є інформаційно ізольованими, розроблена система тісно інтегрує організаційні процеси з інтелектуальним RAG-асистентом. У порівнянні з популярними платформами освітньої гейміфікації, де контент здебільшого створюється вручну, дане рішення використовує автономного ШІ-агента для автоматичної генерації квізів із багаторівневою структурною валідацією JSON-відповідей, що робить його комплексним гібридним продуктом, адаптованим спеціально для освітнього середовища.

Виконана робота має безпосередній зв'язок із науково-дослідними напрямками та стратегією цифровізації Харківського національного

університету радіоелектроніки, зокрема випускової кафедри штучного інтелекту. Розроблена система виступає єдиним цифровим простором, що об'єднує та оптимізує процеси комунікації між адміністрацією кафедри, викладачами, абітурієнтами та студентами під час підготовки та проведення днів відкритих дверей, хакатонів і наукових семінарів.

Практична новизна отриманих результатів полягає у розробці та імплементації відмовостійкого гібридного алгоритму каскадної маршрутизації (Fallback) та системи управління доступом на основі ролей (RBAC). Каскадний алгоритм здатен динамічно перемикає запити між різними хмарними моделями (Gemini 2.5 Flash, Gemini 1.5 Pro) та локальним сервером (Ollama) у разі виникнення мережових збоїв чи помилок валідації, що значно підвищує надійність системи. Рекомендації щодо подальшої роботи в даному напрямку включають вдосконалення алгоритмів семантичного пошуку, розширення системи модулем автоматичної генерації іменних PDF-сертифікатів для переможців квізів, а також інтеграцію бекенд-сервісів із популярними месенджерами (Telegram, Viber) для підтримки користувачів.

Матеріали кваліфікаційної роботи готові до використання за прямим призначенням – для практичного інформаційного супроводу публічних подій кафедри штучного інтелекту ХНУРЕ. Крім того, розроблений програмний код, архітектура векторного сховища ChromaDB та механізми інтеграції великих мовних моделей можуть бути використані в навчальному процесі університету як наочна демонстраційна база для лабораторних і практичних робіт із дисциплін, пов'язаних із проєктуванням сучасних інформаційних систем та розробкою прикладних рішень на базі штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Розвиток інформаційних систем управління освітою як інструмент реалізації державної освітньої політики : монографія / за ред. С. Л. Лондара ; ДНУ «Інститут освітньої аналітики». Київ, 2020. 258 с. URL: https://iea.gov.ua/wp-content/uploads/2020/07/monografiyaFINAL_sajt.pdf (дата звернення: 12.06.2026).
2. Антонов Є. Гейміфікація як засіб підвищення якості освіти: досвід використання комп'ютерних ігор у навчальному процесі. *Перспективи та інновації науки*. 2022. № 9(14). С. 229–258. URL: <https://eprints/39737> (дата звернення: 13.06.2026).
3. Retrieval-augmented generation for knowledge-intensive NLP tasks / P. Lewis et al. *arXiv*. 2020. URL: <https://doi.org/10.48550/arXiv.2005.11401> (date of access: 15.06.2026).
4. Wu X. FastAPI as a backend framework : bachelor's thesis. Vaasa: Vaasa University of Applied Sciences, 2024. 25 p. URL: https://www.theseus.fi/bitstream/handle/10024/855956/Wu_Xiaomin.pdf (date of access: 16.06.2026).
5. Banks A., Porcello E. Learning react: modern patterns for developing react apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 296 p. URL: [https://ebooks.karbust.me/Technology/Learning\(2020\).pdf](https://ebooks.karbust.me/Technology/Learning(2020).pdf) (date of access: 17.06.2026).
6. Coronel C., Morris S. Database systems: design, implementation, & management. 13th ed. Boston : Cengage Learning, 2018. 837 p. URL: [https://soclbrary.futa.edu.ng/books/2019%20Steven%20Morris%20\(z-lib.org\).pdf](https://soclbrary.futa.edu.ng/books/2019%20Steven%20Morris%20(z-lib.org).pdf) (date of access: 19.06.2026).
7. Nygard M. T. Release it!: design and deploy production-ready software. 2nd ed. Raleigh : Pragmatic Bookshelf, 2007. 350 p. URL: http://www.r-5.org/files/books//Design_and_Deploy_Production-Ready_Software-EN.pdf (date of access: 19.06.2026).

8. Lifespan events. *FastAPI*. URL: <https://fastapi.tiangolo.com/advanced/events/> (date of access: 20.06.2026).
9. Stuttard D., Pinto M. The web application hacker's handbook: finding and exploiting security flaws. 2nd ed. Indianapolis : Wiley, 2011. 912 p. URL: <https://digtvbg.com/files/hacking/The%20Web%20Application%20Hacker.pdf> (date of access: 20.06.2026).
10. Asynchronous I/O (asyncio). *SQLAlchemy 2.0 Documentation*. URL: <https://docs.sqlalchemy.org/en/20/orm/extensions/asyncio.html> (date of access: 21.06.2026).
11. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*. 2019. Vol. 7, no. 3. P. 535–547.
12. Generative AI beginner's guide. *Gemini Enterprise Agent Platform*. URL: <https://cloud.google.com/vertex-ai/docs/generative-ai/learn/overview> (date of access: 22.06.2026).
13. Percival H., Gregory B. Architecture patterns with python: enabling test-driven development, domain-driven design, and event-driven microservices. Sebastopol : O'Reilly Media, 2020. 280 p.
14. Wathan A., Schoger S. Refactoring UI. 2018. 252 p.
15. Wilke C. O. Fundamentals of data visualization: a primer on making informative and compelling figures. Sebastopol : O'Reilly Media, 2019. 390 p. URL: <https://clauswilke.com/dataviz/index.html> (date of access: 25.06.2026).
16. Myers G. J. The art of software testing. 3rd ed. Hoboken : John Wiley & Sons, 2012. 240p. URL: <https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf> (date of access: 25.06.2026).