КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій проєктування та прикладної математики

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТРА»

на тему: «Інтелектуальна інформаційна система дистанційного навчання студентів»

КОРСУН ІЛЛЯ РОМАНОВИЧ
(прізвище, ім’я та по батькові студента повністю)

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій проєктування та прикладної математики

(кафедра)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТППМ

д.т.н., професор Бородавка Є.В.

„___” _____2025 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТРА»

на тему: «Інтелектуальна інформаційна система дистанційного навчання студентів»

Виконав: студент 2-го курсу, групи ІСТм-24

Спеціальності: 126 «Інформаційні системи та технології»

(шифр і назва напряму підготовки, спеціальності)

Магістрант: Корсун І.Р.

(прізвище та ініціали)

Керівник: д.т.н., професор Бородавка Є.В.

(прізвище та ініціали)

Рецензент: д-р філос. Рябчун Ю.В.

(прізвище та ініціали)

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: інформаційних технологій
Освітній рівень: «магістр» за ОП
Спеціальність: 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТППМ

д.т.н., професор Бородавка Є.В.

„___” _____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТРА»

Корсун Ілля Романович

1. Тема роботи: Інтелектуальна інформаційна система дистанційного навчання студентів
   затверджена наказом ректора КНУБА №1619/23/25 від 29.09.2025 р.
2. Керівник роботи: Бородавка Євгеній Володимирович д.т.н., професор кафедри інформаційних технологій проектування та прикладної математики
3. Строк подання студентом роботи до захисту: 18.12.2025 р.
4. Зміст пояснювальної записки за розділами:
   Р.1. Аналіз предметної області та постановка задачі
   Р.2. Методи побудови систем дистанційного навчання
   Р.3. Розробка архітектури системи
   Р.4. Програмна реалізація системи
   Р.5. Тестування та впровадження системи
5. Інформаційні слайди:
   С.1. Порівняльна таблиця функціональних можливостей популярних платформ
   С.2. Дерево цілей
   С.3. Моделі взаємодії Google та Moodle
   С.4. Архітектурна модель системи та логічна модель даних
   С.5. Діаграма Чена
   С.6. Ключові сценарії
   С.7. Структура сайту
   С.8. Діаграми переходу станів інтерфейсу

С.9. Функціональна структура системи
С.10. Модуль відображення навчальних матеріалів
С.11. Модуль інтелектуального помічника
С.12. Інтерфейс сайту
С.13. Результати тестування
С.14. Оцінка зручності використання

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Аналіз предметної області та постановка задачі	Вересень 2025 р.
Р.2. Методи побудови систем дистанційного навчання	Вересень 2025 р.
Р.3. Розробка архітектури системи	Жовтень 2025 р.
Р.4. Програмна реалізація системи	Жовтень 2025 р.
Р.5. Тестування та впровадження системи	Листопад 2025 р.
Остаточне оформлення роботи	Грудень 2025 р.
Направлення роботи на рецензування, перевірку на плагіат	Грудень 2025 р.
Попередній захист роботи на кафедрі	Грудень 2025 р.

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	Дата	Підпис
Прийом програмного продукту	д-р філос. Рябчун Ю.В.		

8. Дата видачі завдання: 01.09.2025 р.

Керівник _____ Бородавка Є.В.
(підпис) (прізвище та ініціали)

Магістрант _____ Корсун І.Р.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ

Київський національний університет будівництва і архітектури

Корсун Ілля Романович

факультет автоматизації і інформаційних технологій,
група ІСТм-24

Тема атестаційної випускної роботи: «Інтелектуальна інформаційна система дистанційного навчання студентів»
освітній рівень: магістр,

спеціальність: 126 «Інформаційні системи і технології»,

Науковий керівник: Бородавка Євгеній Володимирович,

доктор технічних наук, професор кафедри інформаційних технологій проектування та прикладної математики

Обсяг роботи. Атестаційна випускна робота магістра складається: розділів 5, стор. 125, таблиць 3, рис. 41, завдання, резюме, анотації, вступу, висновків, списку використаних джерел, додатку.

Актуальність полягає у підвищенні ефективності дистанційного навчання шляхом впровадження інтелектуальних систем підтримки. Існуючі платформи не забезпечують оперативної допомоги студентам поза аудиторією. Застосування великих мовних моделей та методу RAG дозволяє створити автоматизованого асистента для миттєвих консультацій по навчальних матеріалах, компенсуючи відсутність викладача під час самостійної роботи.

У вступі обґрунтовано актуальність теми, сформульовано мету та основні завдання системи, вказані об’єкт та предмет дослідження.

У першому розділі «Аналіз предметної області та постановка задачі» проаналізовано предметну область та недоліки існуючих LMS (Moodle, Google Classroom, Coursera), виявлено проблему “комунікативного розриву”.

Обґрунтовано доцільність використання технологій LLM та RAG для створення інтелектуального помічника.

У другому розділі «Методи побудови систем дистанційного навчання» досліджено методи побудови та архітектурні принципи провідних платформ дистанційного навчання. На основі проведеного аналізу здійснено вибір методу розробки для проєктування власної системи.

У третьому розділі «Розробка архітектури системи» розроблено архітектуру системи, що складається з чотирьох підсистем. Спроєктовано гібридну базу даних, яка поєднує реляційні моделі з векторним модулем знань, а також деталізовано поведінкову модель та сценарії взаємодії акторів.

У четвертому розділі «Програмна реалізація системи» виконано програмну реалізацію спроєктованої архітектури з використанням стеку Astro, TypeScript, PostgreSQL, ChromaDB та Google Gemini. Описано реалізацію алгоритмів динамічного контенту та RAG-запитів.

У п'ятому розділі «Тестування та впровадження системи» проведено комплексне тестування, яке підтвердило стабільність системи та високу зручність. Встановлено, що система пришвидшує пошук інформації у 2.2 рази.

Ключові слова: дистанційне навчання, штучний інтелект, інтелектуальний асистент.

Keywords: distance learning, artificial intelligence, intelligent assistant.

Якість оформлення проекту. Атестаційна випускна робота магістра оформлена у відповідності до діючих нормативних документів та методичних вказівок до виконання дипломних робіт для студентів спеціальності 126 «Інформаційні системи і технології».

Загальний висновок стосовно роботи та присвоєння авторові освітньо-кваліфікаційного рівня «магістр». Робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінки «відмінно».

Науковий керівник _____/ д.т.н, професор Бородавка Є.В. /
(підпис)

Посада, місце роботи: КНУБА, пр-т Повітряних Сил, 31, доктор технічних наук, професор кафедри інформаційних технологій проектування та прикладної математики.

«_____» _____2025 р.

АНОТАЦІЯ

Корсун І.Р. «Інтелектуальна інформаційна система дистанційного навчання студентів».

Атестаційна випускна робота магістра за спеціальністю: 126 «Інформаційні системи і технології». — Київський національний університет будівництва та архітектури. — Київ, 2025.

Атестаційна робота магістра присвячена вирішенню проблеми підвищення ефективності самостійної роботи студентів в умовах дистанційного навчання шляхом розробки інтелектуальної інформаційної системи.

Ключові слова: дистанційне навчання, штучний інтелект, інтелектуальний асистент.

SUMMARY

Korsun I.R. “Intelligent Information System for Distance Learning of Students.”

Master's thesis in the specialty: 126 “Information Systems and Technologies.” — Kyiv National University of Construction and Architecture. — Kyiv, 2025.

The master's thesis is devoted to solving the problem of improving the effectiveness of students' independent work in distance learning by developing an intelligent information system.

Keywords: distance learning, artificial intelligence, intelligent assistant.

Зміст

ВСТУП

Актуальність теми. В умовах глобальної цифровізації та масового переходу до дистанційних та змішаних форм навчання, ефективність освітніх платформ набуває критичного значення. Сучасні системи управління навчання, такі як Moodle, Google Classroom, Coursera успішно вирішують задачі зберігання та доставки навчальних матеріалів, функціонуючи як ефективні цифрові репозиторії.

Однак, цей перехід виявив фундаментальну проблему — відсутність миттєвої, персоніфікованої підтримки студентів під час самостійної роботи. Коли студент стикається з труднощами поза межами аудиторних годин, виникає значний "комунікаційних розрив" — часова затримка між появою питання та отриманням кваліфікованої відповіді від викладача. Це призводить до втрати навчального імпульсу, зниження мотивації та загального падіння ефективності самостійного засвоєння матеріалу.

Водночас, останні роки відзначилися стрімким розвитком великих мовних моделей (Large Language Model (LLM)), які продемонстрували здатність розуміти та генерувати людиноподібний текст на надзвичайно високому рівні. Поява архітектури Retrieval-Augmented Generation (RAG) відкрила можливість "заземлити" ці моделі, змусивши їх генерувати відповіді, базуючись виключно на верифікованому навчальному контенті.[8]

Таким чином, актуальність цієї кваліфікаційної роботи полягає у нагальній потребі освітньої сфері у подоланні комунікаційного розриву та підвищенні ефективності дистанційного навчання, а також в одночасній появі потужних технологічних інструментів LLM та RAG, що дозволяють вирішити цю проблему на якісно новому рівні, створивши інтелектуального помічника, доступного 24/7.

Метою кваліфікаційної роботи є розробка та програмна реалізація інтелектуальної інформаційної системи на базі існуючого освітнього сайту, здатної підвищити ефективність самостійної роботи студентів шляхом надання миттєвої,

персоніфікованої підтримки на основі великих мовних моделей та архітектури RAG.

Для досягнення поставленої мети були визначені наступні **завдання:**

1. Дослідити ринок існуючих систем дистанційного навчання (Learning Management System (LMS)), виявити їхні ключові недоліки, а також проаналізувати сучасні підходи до застосування технологій штучного інтелекту в освіті.
2. Розробити концептуальну, функціональну та поведінкову моделі системи, а також визначити її інфраструктурну топологію та логічну структуру баз даних.
3. Обґрунтувати вибір технологічного стеку Astro, TypeScript та реалізувати ключові компоненти, включаючи інтерфейс користувача, API-маршрути (Application Programming Interface) для взаємодії з базою даних та інтелектуальний модуль (AI Core) з інтеграцією LLM.
4. Здійснити функціональне тестування, тестування продуктивності (навантажувальне) та оцінку зручності використанні (System Usability Scale (SUS)), щоб підтвердити працездатність та ефективність розробленого рішення.

Об’єктом дослідження є процес інформаційної підтримки самостійної роботи студентів в умовах дистанційного навчання у закладах вищої освіти.

Предметом дослідження є методи, архітектурні рішення та програмні засоби для проєктування та реалізації інтелектуального помічника на основі великих мовних моделей з використанням архітектури RAG для інтеграції в освітні інформаційні системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Цей розділ присвячено дослідженню предметної області дистанційної освіти. Проводиться детальний аналіз функціональних можливостей, переваг та недоліків існуючих інформаційних систем, таких як Moodle, Google Classroom та Coursera. На основі виявлених проблем, зокрема відсутності миттєвої підтримки та персоналізації, обґрунтовується необхідність впровадження інтелектуальних компонентів. Врешті, формулюються проблеми, визначається мета та завдання кваліфікаційної роботи.

1.1 Огляд сучасних систем дистанційного навчання

1.1.1 Порівняльний аналіз функціональних можливостей популярних платформ

Moodle — це повноцінна LMS з відкритим вихідним кодом, що робить її надзвичайно гнучкою. Вона ідеально підходить для університетів, яким потрібен повний контроль над навчальним процесом.

Викладачі можуть створювати складні курси з різноманітних елементів: лекції, завдання з різними типами файлів, складні тексти з базою питань, інтерактивні H5P-елементи, SCORM-пакети.[9] Студенти отримують доступ до всіх матеріалів, можуть виконувати завдання, проходити тести та відстежувати свій прогрес. Платформа пропонує найширші можливості для комунікації: форуми (загальні та для груп), персональні повідомлення, чати, спільна робота над вікі-сторінками, семінари з можливістю взаємного оцінювання.

Moodle має зручний журнал оцінок, викладач може налаштовувати вагові коефіцієнти для різних завдань, створювати власні критерії оцінювання, використовувати рубрики для детального оцінювання та генерувати зведені звіти. Завдяки відкритому коду та величезній спільності, Moodle має сотні безкоштовних плагінів. Її можна інтегрувати практично з будь-чим: сервісами вебінарів, системами перевірки на плагіат, інформаційними системами університету, репозиторіями контенту.

Moodle — це сучасне, але складне рішення. Його варто обирати, коли потрібне максимальне налаштування та контроль на усіма аспектами навчання.

Google Classroom — це не стільки LMS, скільки надбудова над сервісами Google Workspace for Education. Її головна перевага це простота та миттєве розгортання.

Основний функціонал обертається навколо створення завдань. Викладач може публікувати оголошення, додавати матеріали, файли з Google Drive, відео з YouTube, посилання та створювати завдання. Для кожного студента автоматично створюється копія документа, що спрощує процес здачі та перевірки робіт. Комунікація обмежена, є стрічка для курсу, де можна залишати оголошення та приватні/публічні коментарі до завдань. Для повноцінного спілкування потрібно використовувати інші сервіси, як Google Meet.

Викладач виставляє бали за завдання, може залишати коментарі прямо в документі студента і повертати роботу на доопрацювання. Усі оцінки збираються у вкладці “Оцінки”, яку можна експортувати в Google Sheets. Інтеграція зосереджена на екосистемі Google (Drive, Docs, Sheets, Calendar, Meet), це одночасно і перевага, і недолік. Інтеграція зі сторонніми освітніми інструментами можлива, але не така глибока, як у Moodle.

Google Classroom — ідеальний вибір для шкіл або університетів, які вже використовують Google Workspace і потребують простого інструмента для організації змішаного чи дистанційного навчання без складних налаштувань.

Coursera відрізняється від перших двох платформ, це в першу чергу постачальник контенту. Версія “for Campus” дозволяє університетам інтегрувати тисячі готових курсів від провідних компаній та вишів світу у свої навчальні програми.

Викладачі виступають у ролі кураторів, вони можуть створювати навчальні програми з готових курсів Coursera, відстежувати прогрес студентів та інтегрувати цей контент у власні дисципліни. Студенти отримують доступ до високоякісних

відеолекцій, практичних завдань та проєктів. Взаємодія відбувається переважно в межах конкретного курсу на Coursera. Це дискусійні форуми, де студенти з усього світу або лише з вашого університету можуть ставити запитання та допомагати кожному. Важливою частиною є система, де студенти оцінюють роботи один одного за заданими критеріями.

Оцінювання автоматизоване та стандартизоване в межах платформи. Це тести, вікторини для кожного модуля, програмувальні завдання з автоматичною перевіркою та проєкти, що оцінюються іншими студентами. Університет отримує детальну аналітику щодо прогресу кожного студента. Coursera for Campus пропонує API для інтеграції з існуючими LMS, як Moodle, що дозволяє безперешкодно зараховувати результати з курсів Coursera в основний журнал оцінок університету. Також підтримується система єдиного входу.

Coursera for Campus — це рішення не для створення курсів “з нуля”, а для збагачення існуючих освітніх програм якісним готовим контентом та фокусування на розвитку актуальних для ринку праці навичок.

Таблиця 1.1 Порівняльна таблиця

Критерій	**Moodle**	**Google Classroom**	**Coursera for Campus**
Інструменти	Лекції, семінари, глосарії, тести, завдання, SCORM-пакети.	Завдання, запитання, оголошення, матеріали курсу.	Доступ до тисяч курсів, проєктів, відеолекцій від провідних університетів.
Взаємодія	Форуми, чати, вебінари, вікі, семінари.	Коментарі до завдань, стрічка оголошень.	Дискусійні форуми в межах курсів, взаємне оцінювання.

Продовження таблиці 1.1 Порівняльна таблиця

Оцінювання	Журнал оцінок, різні шкали, критерії, ручне/автоматичне оцінювання.	Бали за завдання, коментарі, експорт оцінок у Google Sheets.	Тести, вікторини, взаємне оцінювання, проєктні роботи.
Інтеграція	Сотні плагінів, LTI, інтеграція з SIS, BigBlueButton.	Глибока інтеграція з екосистемою Google.	API для зв'язку з внутрішніми LMS/LXP, SSO.

1.1.2 Визначення основних недоліків існуючих рішень

Незважаючи на переваги, проведений аналіз виявив чотири системні проблеми, які є спільними для більшості існуючих платформ та створюють нішу для впровадження інтелектуальних рішень.

Рис.1.1 Проблеми існуючих рішень

Проблема 1 — обмежена та негнучка персоналізація навчання.

Більшість сучасних платформ працюють за моделлю “один розмір для всіх”. Контент та завдання подаються лінійно, без урахування індивідуального темпу, рівня знань чи стилю навчання студента.

Прояв у системах:

- Google Classroom практично немає інструментів персоналізації. Всі студенти отримують однакові завдання в однаковій послідовності.
- Moodle дозволяє створювати нелінійні курси, але це вимагає від викладача величезних затрат часу на ручне налаштування всіх можливих сценаріїв. Це не адаптивне навчання, а лише заздалегідь прописані гілки.
- Coursera пропонує високоякісний, але статичний контент. Платформа не адаптує подачу матеріалу, якщо студент не зрозумів якусь тему вона просто пропонує переглянути відео ще раз.

Суть проблеми полягає у відсутності механізмів, які б у реальному часі аналізували дії студента і пропонували йому саме той матеріал, який потрібен для закриття його “прогалин у знаннях”.

Проблема 2 — поверхнева аналітика навчального прогресу.

Існуючі системи збирають багато даних, але надають мало корисної інформації. Аналітика переважно фокусується на результатах, а не на процесі навчання.

Прояв у системах:

- Google Classroom обмежується простим журналом оцінок. Неможливо дізнатися, скільки часу студент витратив на завдання або з якими саме питаннями в тесті виникли труднощі.
- Moodle збирає детальні журнали активності, але вони представлені у вигляді “сирих” даних, які складно інтерпретувати без спеціальних навичок.

Викладач бачить, що студент зробив, але не розуміє, чому він виконав це так чи інакше.

- Coursera має потужну внутрішню аналітику, але вона переважно доступна адміністраторам платформи, а не викладачам чи самим студентам.

Суть проблеми полягає у відсутності прогнозованої аналітики, яка б могла виявляти студентів у “групі ризику” до того, як вони провалять іспит. А також інструментів для глибокого аналізу поведінки, що могли б дати викладачу та студенту цінні поради для покращення процесу навчання.

Проблема 3 — недостатня інтерактивність та низька залученість.

Навчальний процес часто зводяться до пасивного споживання контенту: перегляд відео, читання тексту, проходження тестів. Це призводить до зниження мотивації та залученості студентів.

Прояв у системах:

- Основні інструменти взаємодії в Moodle та Coursera — це асинхронні форуми. Студент ставить питання і може чекати на відповідь годинами або й днями, що руйнує динаміку навчання.
- Більшість завдань не передбачає симуляції реальних умов чи негайного зворотного зв’язку. Наприклад, студент не може поставити уточнююче питання під час проходження тесту.
- Відсутні ігрові елементи (досягнення, бали за активність, рейтинги), які б стимулювали студентів до регулярної роботи.

Суть проблеми — платформи є скоріше сховище контенту, аніж інтерактивним навчальним середовищем, що провокує відчуття ізоляції та знижує ефективність самостійної роботи.

Проблема 4 — відсутність миттєвої підтримки.

Коли у студента виникає проблема чи питання під час самостійної роботи, він не має до кого звернутись.

Прояв у система:

- Комунікація не структурована між різними інструментами: оголошення, форуми, особисті повідомлення, коментарі до завдань. Немає єдиного центру для отримання допомоги.
- Не існує механізму миттєвої, автоматизованої підтримки 24/7. Студент залишається сам на сам із проблемою до моменту, коли викладач зможе відповісти.

Суть проблеми полягає у відсутності інтелектуального помічника, який міг би миттєво відповісти на типові питання за матеріалами курсу, тим самим підтримуючи безперервність навчального процесу та знижуючи навантаження на викладача.

Ці невирішені проблеми чітко окреслюють нішу для розробки інтелектуальної інформаційної системи, яка б фокусувалася на адаптивному навчанні, глибокій аналітиці, інтерактивності та миттєвій підтримці студентів.

1.2 Особливості використання вебплатформ у навчанні

1.2.1 Роль вебплатформ у забезпеченні доступності та інтерактивності освітнього процесу

Вебплатформи кардинально змінюють роль викладача, перетворюючи його з лектора-транслятора знань на модератора навчального процесу. Замість того, щоб витрачати час на читання лекцій, викладач може записати їх на відео та надати студентам для самостійного перегляду. Аудиторний або онлайн час використовується для обговорень, розв’язання практичних завдань та відповідей на питання. Платформи, як Moodle чи Coursera, є ідеальною базою для розміщення таких відео-лекцій та супутніх матеріалів.

Системи збирають дані про успішність студентів у реальному часі. Викладач може миттєво побачити, яка тема викликає найбільше труднощів наприклад, за результатами тесту, та адаптувати наступне заняття, щоб приділити їй більше уваги, це робить викладання більш цілеспрямованим та ефективним. Автоматична

перевірка тестів, збір робіт, розсилка нагадувань про кінцеві терміни звільняють час викладача. Цей час можна інвестувати в індивідуальну роботу зі студентами, надання більш розгорнутого зворотного зв'язку та розробку якісніших навчальних матеріалів.

Вебплатформи пропонують інструменти, які перетворюють пасивне слухання на активну участь, що є ключовим фактором для утримання уваги та мотивації. Замість статичних PDF-файлів, викладачі можуть використовувати інтерактивні тести, опитування, симуляції та вбудовані відео з питаннями. Це стимулює студентів до взаємодії з матеріалом, а не просто його споживання.

Впровадження ігрових елементів, таких як бали за активність, значки за досягнення, рейтингові таблиці та шкали прогресу, створює здоровий дух змагання та мотивує студентів виконувати завдання вчасно і докладати більше зусиль. Інструменти для спільної роботи, такі як форуми, вікі, спільні проєкти в Google Docs, дозволяють студентам працювати в командах, обмінюватися ідеями та вчитися одне в одного. Це долає відчуття ізоляції, характерне для дистанційного навчання.

Гнучкість — одна з головних переваг онлайн-освіти, і вебплатформи є її технологічним фундаментом. Студенти отримують доступ до навчальних матеріалів 24/7 з будь-якої точки світу. Це дозволяє їм навчатися у власному темпі, поєднуючи освіту з роботою чи іншими обов'язками. Календарі та нагадування допомагають не губитися в кінцевих термінах. Технології дозволяють створювати адаптивні сценарії навчання, наприклад, студент, який успішно склав тест з певної теми, може отримати доступ до поглиблених матеріалів, тоді як іншому система пропонує додаткові ресурси для повторення, щоб закрити прогалини в знаннях. Замість годинних лекцій матеріал можна подавати у вигляді коротких, концентрованих блоків: 5-хвилинні відео, короткі статті, тести з кількох питань. Такий формат краще відповідає сучасним моделям споживання інформації та дозволяє навчатися навіть у коротких проміжках часу.

1.2.2 Аналіз потреб та вимог ключових користувачів: студенти та викладачі

Для успішного проєктування інформаційної системи необхідно чітко визначити основні групи користувачів та їхні потреби. У контексті системи дистанційного навчання виділяються дві основні групи: студенти та викладачі.

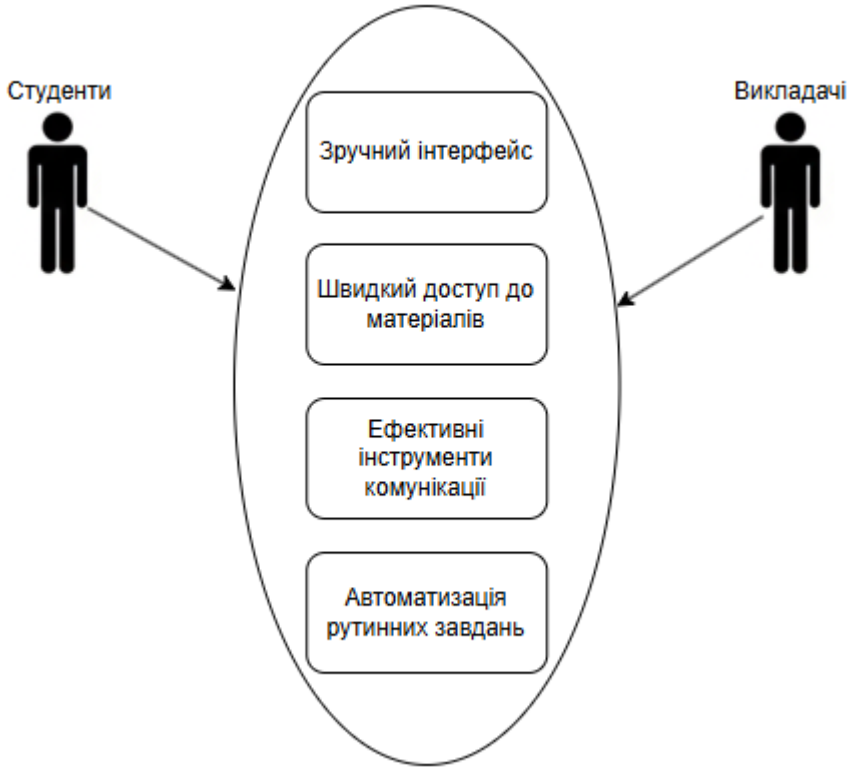


Рис.1.2 Ключові потреби та вимоги від користувачів

Сегмент 1 — студенти.

Основна мета студента — ефективно засвоїти навчальний матеріал, вчасно виконувати завдання та отримувати зворотний зв’язок.

Зручний інтерфейс:

Студент повинен витрачати час на пошук потрібного курсу, завдання чи лекції. Структура має бути логічною та передбачуваною. Більшість студентів використовують смартфони для навчання, тому інтерфейс має коректно відображатися та бути повністю функціональним на мобільних пристроях.

Головний екран має відображати актуальну інформацію: найближчі кінцеві терміни, нові повідомлення, прогрес по курсах.

Швидкий доступ до матеріалів:

Усі матеріали (лекції, відео, книги, посилання) мають бути зібрані в одному місці в межах курсу. Наявність глобального пошуку по всіх курсах та локального в межах одного курсу для швидкого знаходження потрібної інформації. Можливість завантажувати матеріали для перегляду без доступу до інтернету.

Ефективні інструменти комунікації:

Отримувати сповіщення або електронний лист про нові завдання, оцінки, повідомлення від викладача чи відповіді на форумі. Зручний та швидкий спосіб поставити приватне питання викладачу. Наявність форумів або чатів для обговорення завдань та спільної роботи над проєктами.

Автоматизація рутинних завдань:

Автоматичне відстеження переглянутих лекцій та виконаних завдань, наприклад, у вигляді шкал прогресу. Система має автоматично нагадувати про наближення термінів здачі робіт. Автоматичне оцінювання тестів для негайного отримування результату та розуміння своїх помилок.

Сегмент 2 — викладачі.

Основна мета викладача — ефективно організувати навчальний процес, надавати якісний контент, оцінювати знання студентів та мінімізувати адміністративну роботу.

Зручний інтерфейс:

Можливість легко створювати та редагувати курси, додавати матеріали, налаштовувати завдання без потреби в технічних знаннях. Зручне відображення успішності всієї групи, швидкий перехід до робіт конкретного студента,

можливість сортування та фільтрації. Швидкий доступ до всіх інструментів керування: список студентів, аналітика, налаштування.

Швидкий доступ до матеріалів:

Підтримка різних форматів файлів (PDF, DOCX, відео, аудіо), можливість вбудовувати контент зі сторонніх сервісів (YouTube, Google Drive). Можливість зберігати матеріали та завдання для повторного використання в інших курсах чи навчальних роках. Всі здані роботи мають бути зібрані в одному місці для зручної перевірки.

Ефективні інструменти комунікації:

Можливість швидко надіслати оголошення всій групі студентів. Зручний функціонал для коментування та рецензування робіт студентів, наприклад, текстову та голосові коментарі. Інструменти для керування форумами та чатами, щоб підтримувати дискусію в конструктивному напрямі.

Автоматизація рутинних завдань:

Конструктор тестів, які система перевіряє автоматично, заощаджуючи десятки годин роботи. Автоматична перевірка робіт на наявність плагіату, а також створення звітів про активність та успішність студентів.

1.3 Аналіз вимог до інтелектуальних компонентів

1.3.1 Застосування технологій штучного інтелекту в освіті: огляд підходів

Інтеграція технологій штучного інтелекту (ШІ) в освітні платформи відкриває можливості для фундаментальної трансформації навчального процесу, що робить його більш персоналізованим, ефективним та доступним. Розглянемо ключові напрямки застосування ШІ в сучасних освітніх системах.

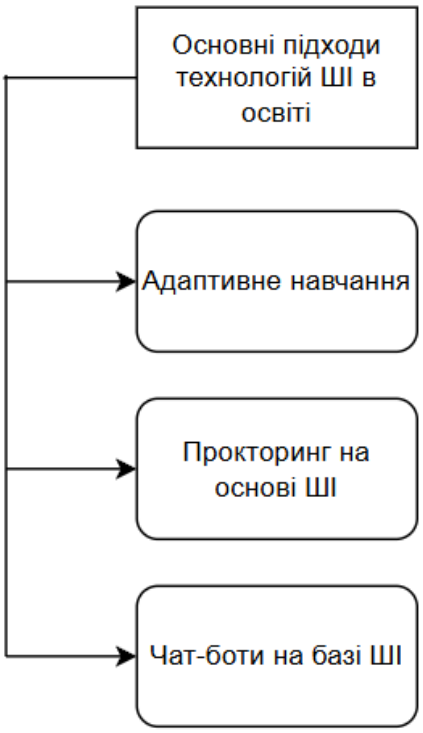


Рис.1.3 Аналіз підходів технологій ШІ в освіті

Адаптивне навчання — це підхід, за якого освітня система в реальному часі підлаштовує подачу матеріалу та складність завдань під індивідуальні потреби кожного студента.[6] ШІ-алгоритми аналізують дані про взаємодію студента з платформою: швидкість виконання завдань, типові помилки, обрані відповіді та історію переглядів. На основі цього аналізу система будує персоналізовану освітню траєкторію:

- Якщо студент демонструє впевнене володіння темою, система пропонує йому складніші завдання або дозволяє перейти до наступного модуля.
- Якщо студент стикається з труднощами, ШІ надає додаткові пояснення, спрощені приклади або фонові теоретичні матеріали для заповнення прогалин у знаннях.

Це дозволяє перейти від лінійної моделі освіти до гнучкої, орієнтованої на індивідуальний прогрес.

Моніторинг на основі ШІ — це система автоматизованого нагляду за процесом складання онлайн-іспитів з метою забезпечення академічної

доброчесності. За допомогою вебкамери, мікрофона та аналізу активності на екрані комп'ютера, ШІ відстежує поведінку студента під час тестування. Система може ідентифікувати потенційні порушення, такі як:

- Відведення погляду від екрана на тривалий час.
- Присутність сторонніх осіб або звуків у приміщенні.
- Використання мобільних пристроїв чи інших заборонених ресурсів.
- Відкриття сторонніх вкладок у браузері.

При виявленні підозрілої активності система фіксує цей момент та робить запис або знімок екрану та сповіщає екзаменатора для подальшої перевірки.

Сучасні ШІ-системи здатні оцінювати не лише тести з закритими питаннями, але й складніші завдання, такі як есе, програмний код чи математичні розв'язки. Це реалізується за допомогою технологій обробки природної мови та машинного навчання.

Штучний інтелект аналізує текст за низкою критеріїв: граматика, орфографія, структура, відповідність темі, унікальність та навіть логічність аргументації. Система автоматично компілює та тестує програмний код, перевіряючи його на коректність виконання, ефективність та відповідність стандартам написання коду.

Автоматизація оцінювання дозволяє надавати студентам миттєвий зворотний зв'язок та значно зменшує навантаження на викладачів.

За аналогією до систем рекомендацій у комерційних системах такі як, Netflix, Amazon, освітні рекомендаційні системи аналізують дані про навчальну історію, інтереси та цілі студента, щоб пропонувати йому релевантний додатковий контент.

Це можуть бути:

- Наукові статті або відео для поглиблення знань з поточної теми.
- Додаткові практичні завдання для закріплення навичок.
- Онлайн-курси з суміжних дисциплін, які можуть бути корисними для майбутньої кар'єри студента.

Такі системи допомагають студентам виходити за межі обов'язкової програми та формувати власну унікальну освітню траєкторію.

Чат-боти на базі ШІ, часто побудовані на великих мовних моделях, виступають у ролі віртуальних асистентів, доступних 24/7. Їхні основні функції:

- Відповіді на поширені запитання щодо навчальних матеріалів або організаційних аспектів курсу.
- Чат-бот може проводити опитування, ставити уточнюючі питання для перевірки знань та надавати підказки під час виконання завдань.
- Надання інформації про розклад, кінцеві терміни, процедури реєстрації тощо.

Інтеграція чат-ботів дозволяє забезпечити безперервну підтримку студентів та автоматизувати відповіді на типові запити.

1.3.2 Роль великих мовних моделей (LLM) штучного інтелекту у персоналізації навчання та автоматизації підтримки студентів

Великі мовні моделі є класом моделей глибокого навчання, що здатні опрацьовувати та генерувати людиноподібний текст на основі гігантський обсягів даних, на яких вони навчалися. Їх інтеграція в освітні системи дозволяє створити багатофункціональних інтелектуальних помічників, здатних виконувати низку ключових завдань.

Основною перевагою LLM є їхня здатність розуміти семантичний контекст запитань та знаходити релевантну інформацію. Для обмеження відповідей виключно рамками навчальних матеріалів курсу та уникнення "галюцинацій" моделі застосовується архітектура RAG.

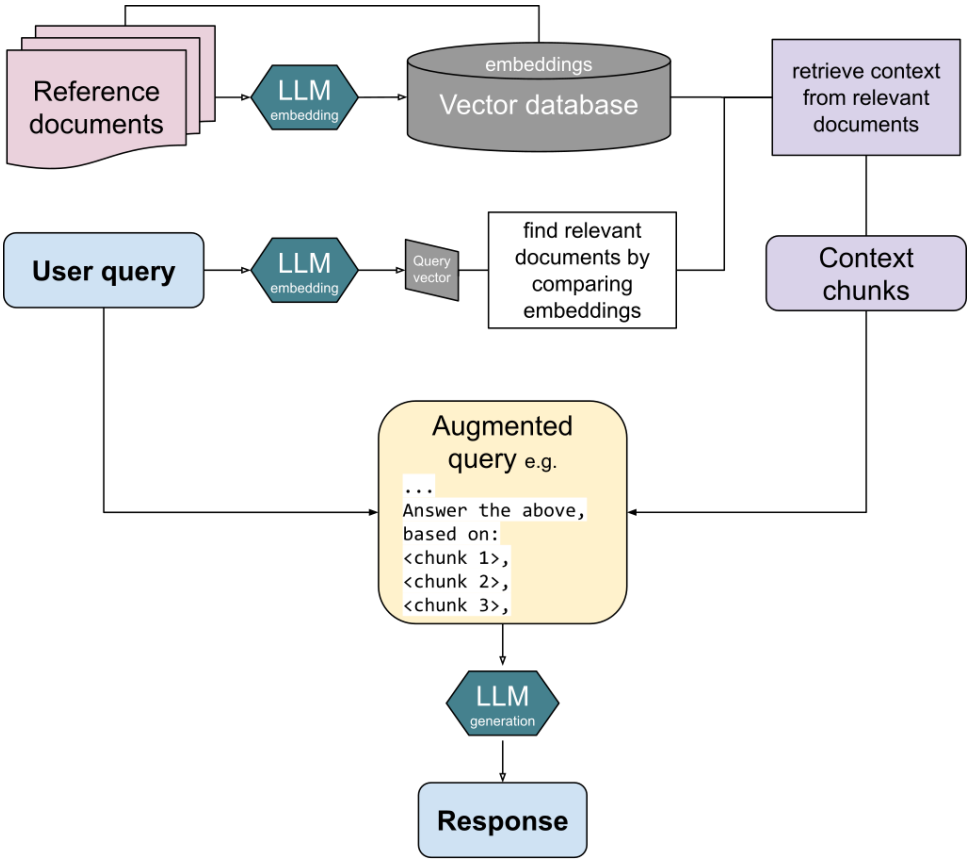


Рис.1.4 Процес RAG[12]

Процес функціонування RAG:

1. Усі навчальні матеріали (лекції, PDF-файли, статті) попередньо обробляються та перетворюються на векторні представлення, які зберігаються у спеціалізованій векторній базі даних. Цей процес дозволяє машині зрозуміти семантичне значення кожного фрагмента тексту.
2. Коли студент ставить питання, система перетворює його запит на такий самий вектор і здійснює пошук у базі даних, знаходячи найбільш семантично близькі фрагменти навчального матеріалу.
3. Знайдені фрагменти тексту разом з оригінальним питанням студента подаються на вхід LLM як контекст. Модель отримує інструкцію згенерувати відповідь, базуючись виключно на наданій інформації.

Такий підхід гарантує, що відповіді є точними, релевантними до курсу та позбавленими сторонньої інформації.

LLM можуть виступати як інструмент для автоматичного створення великих унікальних текстових завдань, що є важливим для об’єктивного оцінювання та самоперевірки знань студентів. За допомогою технік інженерії запитів, викладач може подати моделі фрагмент навчального тексту та сформулювати запит на генерацію завдань різного типу:

- Модель здатна не тільки сформулювати правильну відповідь, але й згенерувати правдоподібні, але невірні варіанти.
- Створення тверджень, що вимагають від студента оцінки їхньої істинності.
- Формулювання питань, що вимагають короткої текстової відповіді або розгорнутого есе.

Це дозволяє створювати варіативні тести, що знижує можливість плагіату та дозволяє студентам практикуватися на необмеженій кількості унікальних завдань.

Інтелектуальний помічник на базі LLM може виконувати роль вчителя, надаючи студентам дозовану допомогу під час вирішення складних завдань. Цей процес, відомий як поетапної допомоги, спрямований не на надання готової відповіді, а на стимулювання мислення студента.

Способи реалізації:

- Якщо студент не може розв’язати задачу, система може поставити питання, що наштовхне його на правильний хід думок.
- Помічник може порекомендувати прочитати ще раз конкретний розділ лекції або переглянути відео, де пояснюється необхідна концепція.
- Для складних багатоетапних завдань штучний інтелект може запропонувати перший крок або розбити задачу на менші, більш керовані підзадачі.

Такий підхід сприяє розвитку навичок критичного мислення та самостійного вирішення проблем, на відміну від простого копіювання готових розв’язків.

1.3.3 Формулювання вимог до інтелектуального модуля системи

Для проєктування та розробки ефективного інтелектуального компонента необхідно визначити чіткі функціональні та нефункціональні вимоги. Вони слугуватимуть критеріями для вибору архітектурних рішень, технологій та подальшого тестування системи.

Функціональні вимоги описують, що саме повинна робити система, її основні операції та можливості.

- Система повинна приймати на вхід текстові запити від користувачів, сформульовані українською мовою у довільній формі, та коректно їх інтерпретувати.
- Система має здійснювати семантичний пошук релевантної інформації у заздалегідь завантаженому корпусі документів або навчальних матеріалах. На основі знайдених фрагментів модуль повинен генерувати зв’язну, фактологічно коректну відповідь, що не виходить за межі наданого контексту.
- Система повинна мати функціонал для створення тестових завдань на основі вказаного фрагмента навчального матеріалу.
- Система повинна надавати програмний інтерфейс або адміністративну панель для завантаження нових документів. Після завантаження модуль має автоматично обробляти, сегментувати та проводити векторну індексацію тексту для його подальшого використання у пошуку.

Нефункціональні вимоги визначають, як система повинна виконувати свої функції, встановлюючи критерії якості, продуктивності та надійності.

- Не менше 95% згенерованих відповідей повинні бути фактично коректними та повністю відповідати джерелам інформації з бази документів. Система не винна вигадувати факти.
- Відповідь системи має прямо стосуватися поставленого питання користувача, без надання надлишкової або нерелевантної інформації.

- Час від моменту отримання запиту до надання повної текстової відповіді кінцевому користувачу не повинен перевищувати 3 секунди для 90% усіх запитів при середньому навантаженні.
- Ця вимога визначає ефективність процесу, що здійснює самонавчання. Процес повної індексації нового навчального документа обсягом до 50 сторінок та його інтеграції в базу знань не повинен займати більше 10 хвилин.
- Архітектура інтелектуального модуля повинна забезпечувати можливість горизонтального масштабування. Система має стабільно обробляти одночасні запити від щонайменше 100 користувачів без суттєвої деградації швидкості реакції.
- Інтелектуальний модуль повинен бути доступним для використання не менше 99.5% часу, що передбачає наявність механізмів витривалості та моніторингу його стану.

1.4 Постановка задачі дослідження

1.4.1 Формулювання проблеми та обґрунтування актуальності розробки

Масовий перехід до дистанційних та змішаних форм навчання виявив ключовий недолік сучасних систем управління навчанням: вони ефективно функціонують як архіви навчальних матеріалів, але не забезпечують належної динамічної підтримки студентів у процесі самостійної роботи.[19] Коли студент стикається з труднощами у розмінні матеріалу або при виконанні завдання поза межами розкладу занять, виникає значна часова затримка між появою питання та отриманням кваліфікованої відповіді від викладача.

Цей комунікаційний розрив спричиняє наступні негативні наслідки:

- Втрата навчального імпульсу;
- Зниження мотивації;
- Збільшення когнітивного навантаження.

Таким чином, проблема полягає у відсутності в існуючих системах дистанційного навчання інструментів для надання миттєвої, персоніфікованої та контекстуально-залежної підтримки студентам, що призводить до зниження ефективності їхньої самостійної роботи та загальних академічних результатів.

Актуальність розробки інтелектуальної інформаційної системи зумовлена збігом трьох ключових факторів: освітнього, технологічного та соціально-економічного.

Глобалізація та цифровізація освіти зробили дистанційне навчання не тимчасовим рішенням, а повноцінною частиною освітнього ландшафту. Підвищення його ефективності є пріоритетним завданням для освітніх установ по всьому світу. Існуючий дефіцит інструментів для підтримки студентів є однією з головних перешкод на цьому шляху.

Поява та стрімкий розвиток великих мовних моделей вперше в історії робить можливим створення ефективних віртуальних асистентів, здатних розуміти природну мову та вести змістовний діалог у вузькоспеціалізованій предметній області. Технології, що ще кілька років тому були експериментальними, сьогодні досягли рівня, достатнього для їх практичного застосування в освітніх продуктах.

Сучасна педагогічна наука визнає, що індивідуальний підхід до навчання є значно ефективнішим за єдину модель.[24] Розробка інтелектуального помічника дозволяє масштабувати персоналізована підтримку, надаючи кожному студенту фактично індивідуального вчителя, доступного 24/7. Це є неможливим для реалізації силами лише викладацького складу при великих групах студентів.

Отже, актуальність роботи визначається, з одного боку, нагальною потребою освітньої сфери в підвищенні якості дистанційного навчання, а з іншого — наявністю потужних технологічних інструментів, що дозволяють вирішити цю проблему на якісно новому рівні.

1.4.2 Визначання мети та завдання кваліфікаційної роботи

Мета — розробити інтелектуальну інформаційну систему, на базі існуючого освітнього сайту КНУБА, що підвищить ефективність самостійної роботи студентів шляхом надання миттєвої, персоніфікованої підтримки на основі великих мовних моделей.

Завдання:

1. Провести аналіз предметної області.
2. Спроєктувати архітектуру системи.
3. Розробити програмні модулі.
4. Інтегрувати та протестувати систему.
5. Провести експериментальну апробацію.

1.4.3 Побудова дерева цілей проєкту

Головна мета — розробити інтелектуальну інформаційну систему, здатну підвищити ефективність самостійної роботи студентів.

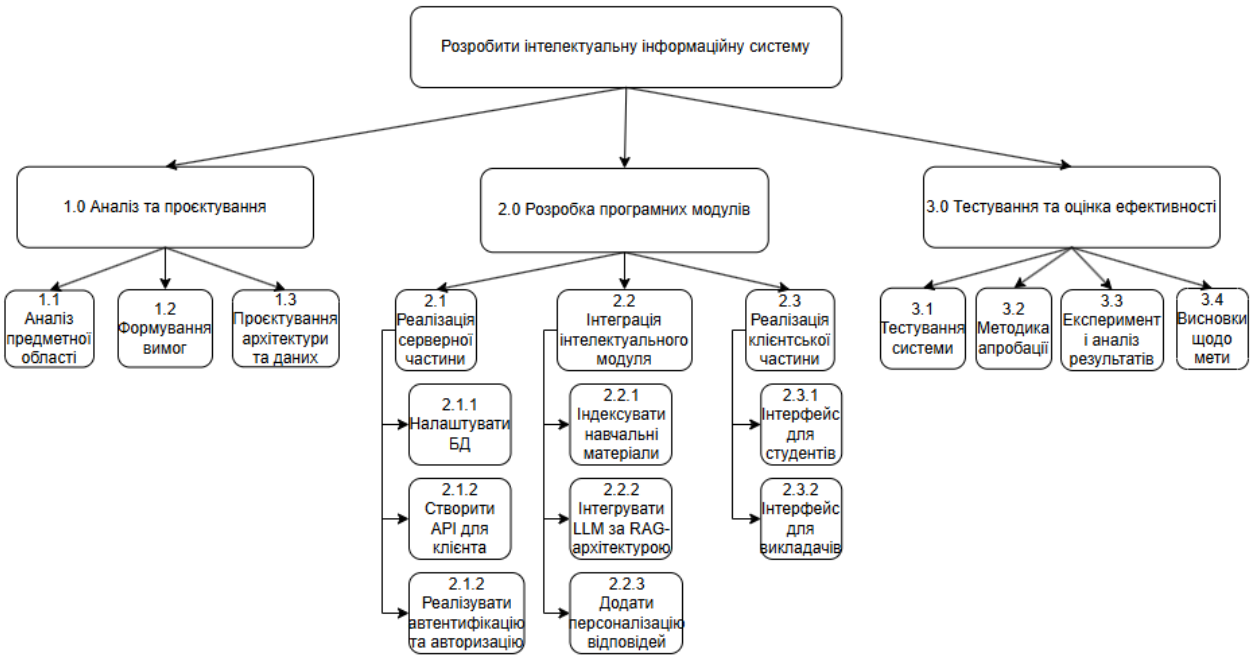


Рис.1.5 Дерево цілей

1.0. Аналіз проєктування

1.1. Провести аналіз предметної області та існуючих рішень.

1.2. Сформулювати функціональні та нефункціональні вимоги до системи.

1.3. Спроєктувати архітектуру системи та модель даних.

2.0. Розробка програмних модулів

2.1. Реалізація серверної частини (Backend)

2.1.1. Налаштувати базу даних (схеми користувачів, курсів, контенту).

2.1.2. Створити API для взаємодії з клієнтською частиною.

2.1.3. Розробити модуль автентифікації та авторизації.

2.2. Інтеграція інтелектуального модуля (AI Core)

2.2.1. Реалізувати механізм індексації навчальних матеріалів.

2.2.2. Інтегрувати велику мовну модель за RAG-архітектурою.

2.2.3. Розробити логіку для персоналізації відповідей.

2.3. Реалізація клієнтської частини (Frontend)

2.3.1. Створити інтерфейс для студентів (перегляд курсів, чат з AI-помічником).

2.3.2. Створити інтерфейс для викладачів (керування курсами та матеріалами).

3.0. Тестування та оцінка ефективності

3.1. Провести комплексне тестування системи (функціональне, навантажувальне).

3.2. Розробити методику експериментальної апробації.

3.3. Провести експеримент, зібрати дані та проаналізувати результати.

3.4. Сформулювати висновки щодо досягнення головної мети.

2. МЕТОДИ ПОБУДОВИ СИСТЕМ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1 Методи аналізу та проєктування інформаційних систем

Проєктування інформаційних систем (ІС) є складним, багатоетапним процесом, що вимагає застосування структурованих методологій для забезпечення якості, надійності та відповідності кінцевого продукту вимогам замовника. Вибір конкретного методу залежить від масштабу проєкту, його складності та специфіки предметної області. Історично склалися два фундаментальні підходи до аналізу та проєктування ІС — структурний та об’єктно-орієнтований.

Структурний підхід, що домінував у 70-80-х роках XX століття, базується на принципі декомпозиції. Складна задача розбивається на простіші підзадачі які, у свою чергу, також можуть бути декомпозовані. Основний акцент робиться на функціях, які система повинна виконувати, та на потоках даних між цими функціями.[4] Для візуалізації та аналізу використовуються наступні моделі:

Діаграми потоків даних графічно представляють процеси перетворення даних в системі, джерела, приймачі та сховища даних. Ці діаграми дозволяють моделювати систему на різних рівнях абстракції, від загальної контекстної діаграми до детального опису кожного процесу.

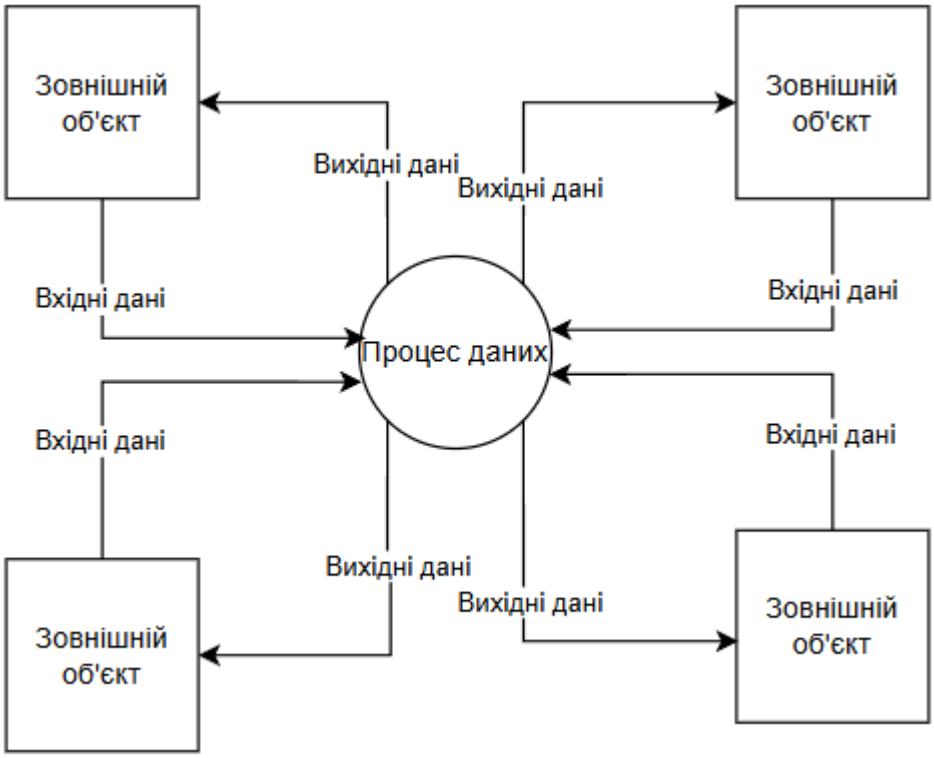


Рис.2.1 Діаграма потоків даних

Модель «сутність-зв’язок» використовується для проєктування баз даних. Вона описує предметну область у вигляді набору сутностей (об’єктів), їх атрибутів та зв’язків між ними, наприклад, “студент” складає “іспит”.

Рис.2.2 Приклад схеми «сутність-зв’язок»

Діаграми переходів станів моделюють поведінку системи, що залежить від часу, показуючи, як система переходить з одного стану в інший під впливом зовнішніх подій.

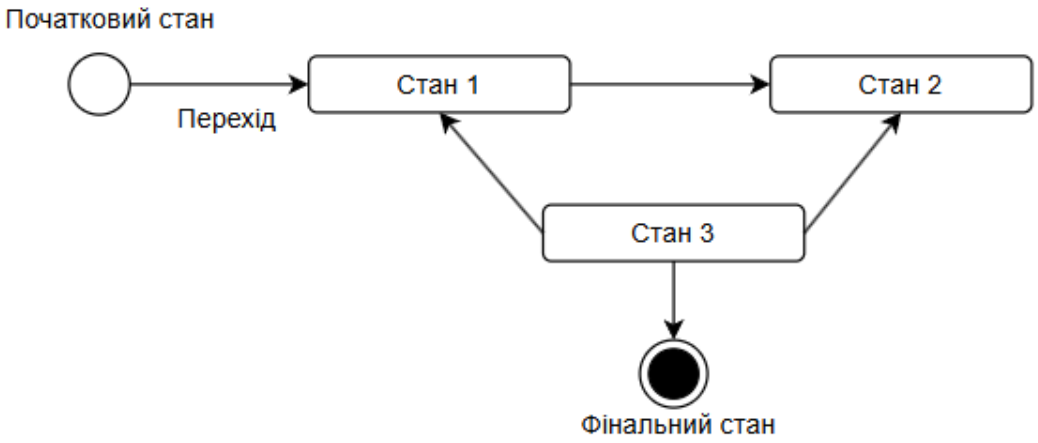


Рис.2.3 Діаграма станів

Недоліком структурного підходу є розрив між моделями функцій та моделями даних, що ускладнює модифікацію системи в майбутньому.

Об’єктно-орієнтований підхід (ООП), що став стандартом у сучасному програмному проєктуванні, розглядає систему як сукупність взаємодіючих об’єктів. Кожен об’єкт укладає в собі як дані, так і методи для їх обробки. Це дозволяє створювати більш гнучкі, масштабовані та легкі для супроводу системи. Ключовими принципами ООП є зводити в єдине ціле, успадкування та різноманітність. Для моделювання використовується уніфікована мова моделювання, яка пропонує набір стандартних діаграм для візуалізації різних аспектів системи:

Діаграми варіантів використання описують функціональність системи з точки зору користувача (актора), визначаючи хто і що може робити в системі.

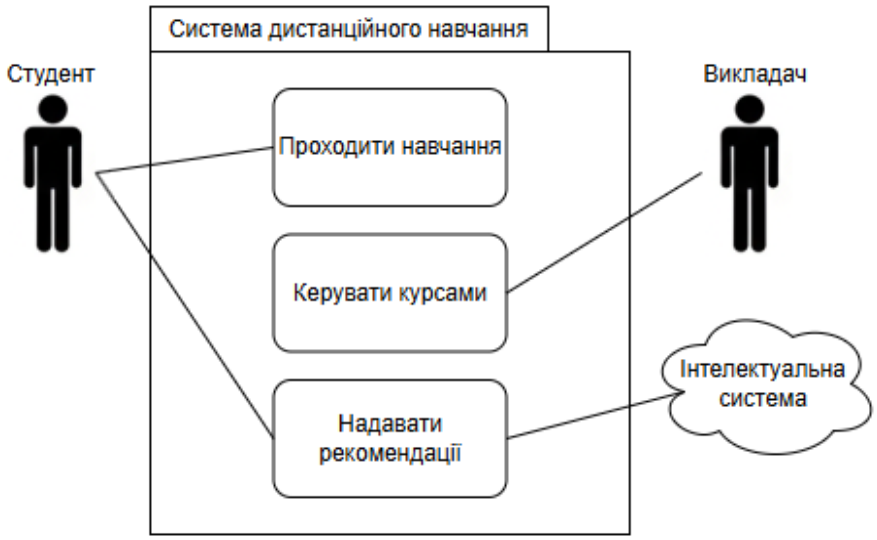


Рис.2.4 Діаграма прецедентів

Діаграми класів показують статичну структуру системи: класи, їх атрибути, методи та зв’язки між ними (асоціація, злиття, успадкування). Це фактично є кресленням архітектури програми.

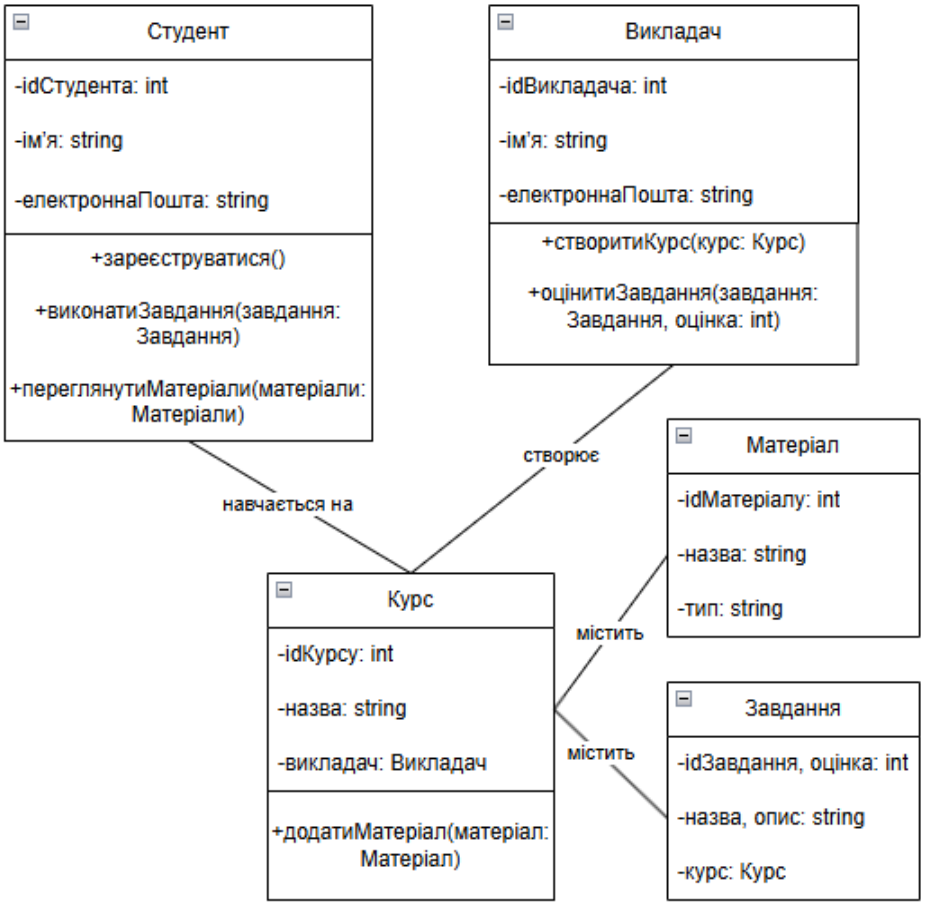


Рис.2.5 Приклад діаграми класів

Діаграми послідовності демонструють динамічну взаємодію об’єктів у часі для виконання конкретного сценарію, наприклад, послідовність викликів методів при авторизації користувача.

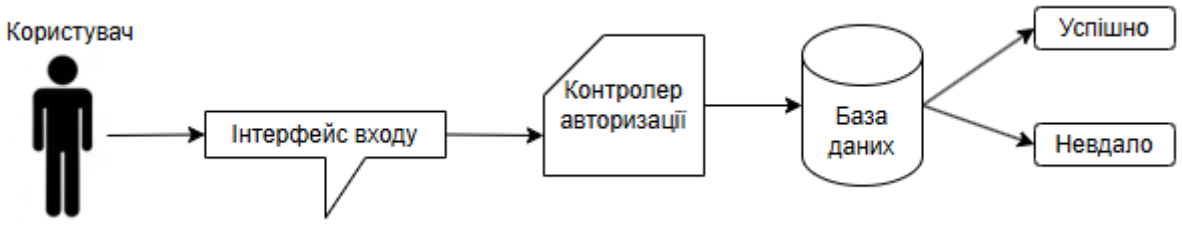


Рис.2.6 Діаграма послідовності для авторизації користувача

Діаграми станів аналогічні структурному підході, але описують життєвий цикл одного конкретного об’єкта.

Життєвий цикл розробки ІС

Незалежно від обраного підходу, процес розробки інформаційної системи керується певною моделлю життєвого циклу, найбільш поширеними є:

Каскадна модель — класична модель, що передбачає послідовне проходження етапів: аналіз вимог, проєктування, реалізація, тестування, впровадження та супровід. Перехід на наступний етап можливий лише після повного завершення попереднього. Модель є жорсткою і погано адаптується до змін вимог.

Ітеративні та інкрементальні моделі — сучасні гнучкі методології, які передбачають розробку системи короткими циклами (ітераціями або спринтами). Наприкінці кожного циклу створюється робоча версія продукту з новим функціоналом. Це дозволяє швидко отримувати зворотний зв’язок від замовника та оперативно вносити зміни до проєкту.

Для розробки сучасної інтелектуальної інформаційної системи найбільш доцільним є використання об’єктно-орієнтованого підходу в рамках гнучкої методології розробки. Це дозволить створити гнучку архітектуру, здатну до

подальшого розширення та ефективно реагувати на можливі зміни у вимогах до функціоналу системи.

В рамках об’єктно-орієнтованого підходу, UML (Unified Modeling Language) виступає як стандартний інструмент для візуального проєктування програмного забезпечення. Він надає набір нотацій для створення діаграм, що описують систему з різних точок зору, забезпечуючи чітке розуміння її структури та поведінки для всіх учасників розробки. Для проєктування інформаційної системи дистанційного навчання найбільш актуальними є:

Діаграма варіантів використання дозволяє ідентифікувати ключових акторів системи, наприклад, студент чи викладач та визначити варіанти використання — ті задачі, які вони виконують за допомогою системи, наприклад, переглянути лекцію, завантажити навчальний матеріал, поставити питання AI-помічнику. Ця діаграма є основною для визначення функціональних вимог.

Діаграма класів є статичним “кресленням” системи. Вона визначає основні класи програмного коду, наприклад, User, Course, Lecture, AI_Assistant, їхні атрибути та методи, а також відношення між ними (асоціація, успадкування). Ця діаграма є фундаментом для написання коду та проєктування бази даних.

Діаграма послідовності моделює динамічну поведінку системи, показуючи взаємодію об’єктів у часі для виконання конкретного сценарію.

Після визначення методології та інструментів моделювання, наступним кроком є вибір архітектурного зразку — високорівневого шаблону організації системи.

Клієнт-серверна архітектура — це фундаментальна модель більшості сучасних вебсистем. Вона передбачає розподіл ролей:

- Клієнт: програмне забезпечення на стороні користувача, зазвичай веббраузер, яке відповідає за відображення інтерфейсу та відправку запитів на сервер.

- Сервер: потужний комп’ютер, що зберігає дані та бізнес-логіку, обробляє запити від клієнтів та повертає їм результати.

На базі цієї моделі будуються більш складніші архітектурні рішення.

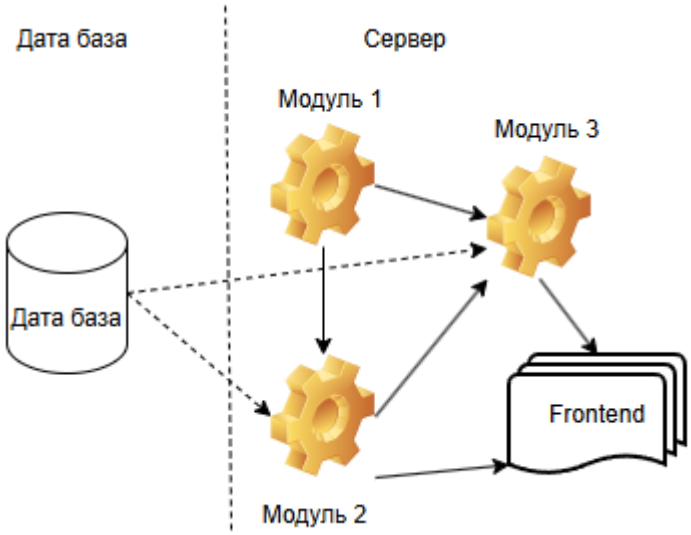


Рис.2.7 Монолітна архітектура

Історично перший та найпростіший підхід, при якому вся функціональність програми реалізована в єдиному, неподільному модулі. Всі компоненти — користувацький інтерфейс, бізнес-логіка, доступ до даних тісно пов’язані та розгортаються як єдине ціле. Класичним прикладом є стандартна інсталяція Moodle.

Переваги: простота розробки на початкових етапах, легкість тестування та розгортання.

Недоліки: ускладнення супроводу та модифікації зі зростанням системи; низька відмовостійкість, наприклад, збій одного компонента може вивести з ладу всю систему; технологічна обмеженість це коли вся система має бути написана на одному стеку технологій.

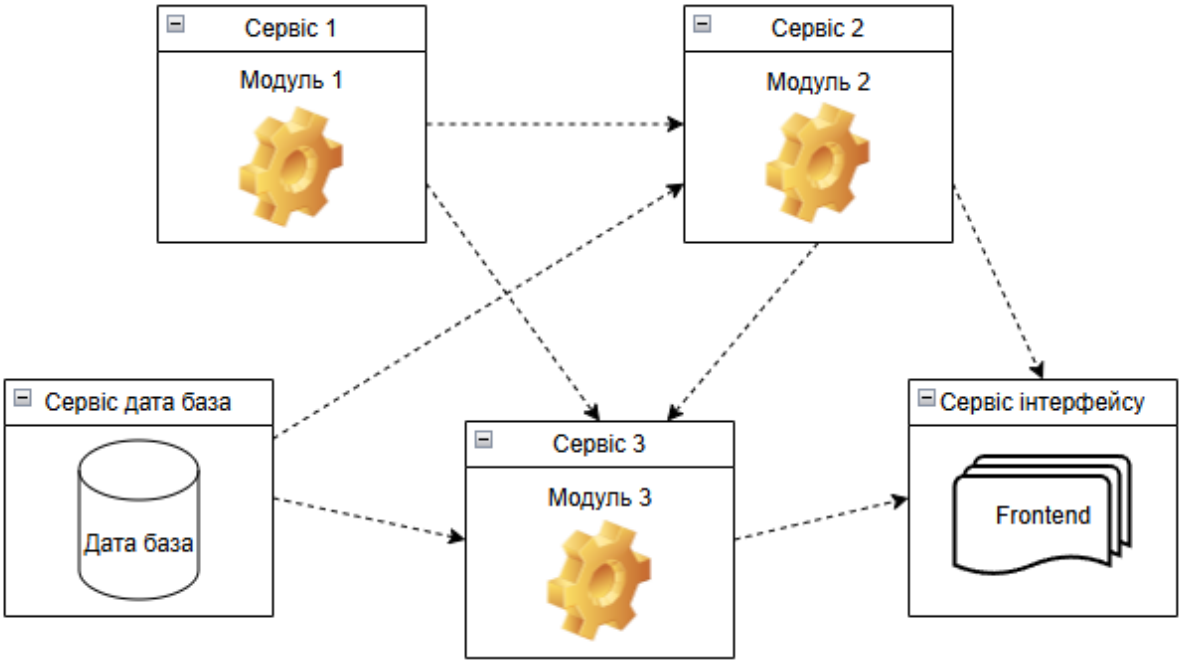


Рис.2.8 Мікросервісна архітектура

Сучасний підхід, що передбачає розбиття системи на набір невеликих, незалежних сервісів. Кожен сервіс відповідає за конкретну бізнес-функцію, наприклад, сервіс аутентифікації, сервіс управління курсами, сервіс інтелектуального помічника і може бути розроблений, розгорнутий та масштабований незалежно від інших. Взаємодія між сервісами відбувається через легковагові протоколи, зазвичай API на основі HTTP/REST.[15]

Переваги: висока гнучкість та масштабованість, можна масштабувати лише ті сервіси, що мають високе навантаження; технологічна свобода, кожен сервіс може бути написаний на найбільш відповідній для нього технології; підвищена відмовостійкість, збій одного сервісу не впливає на роботу інших.

Недоліки: підвищена складність управління та моніторингу розподіленої системи; ускладнення процесу розгортання; необхідність вирішення проблем мережевої взаємодії.

Для розробки інтелектуальної системи дистанційного навчання мікросервісна архітектура є перспективним вибором, оскільки дозволяє виокремити ресурсоємний інтелектуальний модуль в окрему, незалежну одиницю.

Це забезпечує гнучкість у виборі технологій для AI-компонента та дозволяє масштабувати його окремо від основної логіки платформи.

2.2 Архітектура та принципи побудови Google Classroom

Google Classroom є яскравим прикладом інформаційної системи, побудованої за принципами хмарно-орієнтованої та сервіс-орієнтованої архітектури. На відмінну від традиційних монолітних систем управління навчанням, таких як Moodle, Classroom не є єдиним незалежним додатком, а виступає в ролі інтеграційної оболонки або хабу, що об’єднує функціональність існуючих сервісів екосистеми Google Workspace for Education.

Фундаментально, архітектура Google Classroom базується на моделі API-driven microservices. Це означає, що система складається з набору незалежних, слабко зв’язаних сервісів, які взаємодіють між собою через програмні інтерфейси додатків.[11]

Основні компоненти цієї архітектури включають:

Клієнтська частина це користувацький інтерфейс, реалізований як односторінковий додаток з використанням сучасних вебфрейворків. Також вона забезпечує високу швидкість реакції інтерфейсу, оскільки більшість логіки виконується на стороні клієнта, а з сервером відбувається обмін лише необхідними даними у форматі JSON.

Серверна частина розподілена система мікросервісів, що працює на інфраструктурі Google Cloud Platform. Основна логіка Classroom управління курсами, завданнями, оцінками реалізована у вигляді окремого набору сервісів, але ключова функціональність передається іншим продуктам Google через їхні API.

Інтеграція з сервісами Google Workspace це ядро архітектури Classroom. Замість “винаходу велосипеда” та створення власних інструментів, система нативно інтегрується з існуючими потужними сервісами.

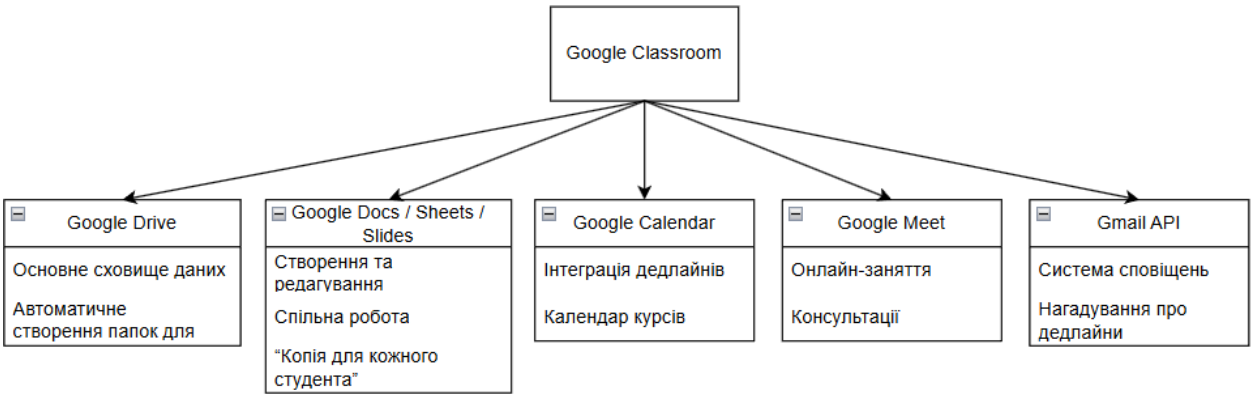


Рис.2.9 Інтеграції Google Classroom

- Google Drive використовується як основне сховище для всіх навчальних матеріалів, файлів завдань та робіт студентів. Структура папок для кожного курсу та завдання створюється автоматично.
- Google Docs, Sheets, Slides сприяють як інструменти для створення, редагування та спільної роботи над завданнями. Функція “Створити копію для кожного студента” є яскравим прикладом глибокої API-інтеграції.
- Google Calendar автоматично інтегрує кінцеві терміни завдань у персональному календарі студентів та викладачів.
- Google Meet використовується для проведення онлайн-занять та консультацій.
- Gmail API застосовується для систем сповіщень.

База даних Google це високомасштабовані, глобально розподілені системи Google, що дозволяють забезпечити низьку затримку та високу доступність сервісу для мільйон користувачів одночасно.

Розробка та підтримка продукту такого масштабу базується на передових інженерних практиках, які популяризувала сама компанія Google:

- Agile та ітеративна розробка, що включає в себе новий функціонал розробляється та впроваджується короткими циклами, що дозволяє швидко реагувати на відгуки користувачів та змінювати пріоритети.

- Безперервна інтеграція та розгортання це процеси тестування та випуску оновлень повністю автоматизовані. Це дозволяє розгортати нові версії для користувачів по всьому світу часто та з мінімальними ризиками.
- Інженерія надійності сайту використовує замість традиційного підходу до адміністрування, Google використовує інженерний підхід до забезпечення надійності. Команди розробляють програмні рішення для автоматизації моніторингу, масштабування та відновлення системи після збоїв, що гарантує її надвисоку доступність.

Таким чином, з точки зору розробника, Google Classroom є не стільки продуктом, скільки платформою-агрегатором. Його головна сила полягає не у власному унікальному функціоналі, а в потенціалі всієї екосистеми Google. Це дозволяє досягти величезної масштабованості та надійності, проте накладає обмеження на гнучкість та модернізацію, оскільки система повністю залежить від можливостей та обмежень інтегрованих у неї сервісів.

Щоб краще зрозуміти модель взаємодії сервісів, розглянемо типовий сценарій роботи системи на прикладі створення завдання викладачем та його виконання студентом.

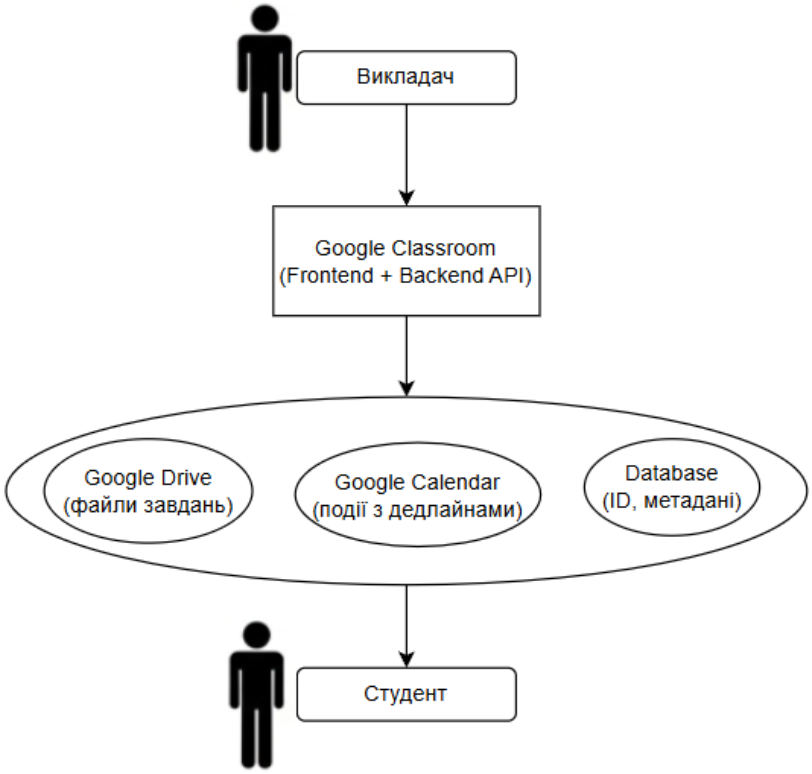


Рис.2.10 Приклад моделі взаємодії сервісів

1. Ініціація: викладач у вебінтерфейсі Google Classroom натискає кнопку “Створити завдання” та заповнює форму — назва, опис, термін здачі, прикріплює шаблонний документ з Google Drive.
2. Запит до Backend: клієнтська частина (Frontend) формує та відправляє POST-запит до API-ендпоінту серверної частини Classroom. Цей запит містить всю інформацію про завдання.
3. Перекладання сервісів (Backend): серверна частина Classroom виступає в ролі координатора. Вона не виконує всю робота сама, а послідовно робить API-виклики до інших сервісів:
    - Google Drive API: створюється копія шаблонного документа для кожного студента курсу у відповідній папці на Google Drive. При цьому система керує правами доступу — студент отримує права на редагування своєї копії.
    - Google Calendar API: створюється подія у календарі курсу та в особистих календарях студентів із зазначенням кінцевого терміну використання.

- Власна база даних: запис про нове завдання, його ID, посилання на файли, метадані зберігаються у власній базі даних Classroom.

4. Виконання: студент отримує сповіщення, відкриває завдання, переходить за посиланням до своєї копії документа в Google Docs та виконує роботу.
5. Здача роботи: після завершення студент натискає кнопку “Здати”. Frontend надсилає запит на відповідний API-ендпоінт.
6. Зміна прав доступу: серверна частина Classroom знову звертається до Google Drive API і змінює права доступу до файлу — права студента на редагування відкликаються, а викладач стає власником документа. Це є технічною реалізацію процесу здачі роботи.

Цей приклад наочно демонструє, що основна цінність архітектури Classroom полягає не у зберіганні даних, а в керуванні взаємодією та правами доступу до ресурсів, що знаходяться в інших сервісах. Така модель побудови системи має як значні переваги, так і суттєві недоліки з інженерної точки зору.

Сильні сторони

Система успадковує ці характеристики від інфраструктури Google Cloud. Вона здатна обслуговувати сотні мільйонів користувачів без необхідності для освітніх закладів піклуватися про сервери, їхню підтримку та оновлення.

Замість розробки складних компонентів таких як, текстового редактора, сховища файлів, календаря з нуля, команда розробників концентрується на інтеграції вже існуючих, зрілих та надійних продуктів.

Модель безпеки передана Google Identity Platform. Це забезпечує надійну автентифікацію, включаючи двофакторну авторизацію через протокол OAuth 2.0 та централізоване управління доступом.[1]

Слабкі сторони

Система повністю залежить від екосистеми Google. Це створює значні перешкоди для міграції на іншу платформу, оскільки всі дані та процеси тісно інтегровані з сервісами, які є програними забезпеченнями.

На відміну від системи з відкритим кодом, таких як Moodle, освітні заклади не можуть модифікувати ядро системи, додавати власні унікальні модулі або змінювати логіку роботи відповідно до специфічних освітніх потреб. Функціонал обмежений тим, що надає компанія Google.

Внутрішня логіка роботи серверної частини та алгоритми є закритими. Неможливо провести незалежний аудит коду або гарантувати, що система не змінить свою поведінку після чергового оновлення.

Таким чином, архітектурний вибір Google Classroom є оптимальний для масового розповсюдження стандартизованого продукту, але створює обмеження для глибокої адаптації під унікальні навчальні процеси конкретного закладу освіти.

2.3 Архітектура та методи розробки Moodle

Moodle є представником класичних LMS і побудований на принципах монолітної архітектури з потужною модульною системою. На відміну від Google Classroom, який є агрегатором зовнішніх сервісів, Moodle є самодостатнім програмним продуктом з відкритим вихідним кодом, що встановлюється на власний сервер освітньої установи.[13]

В основі Moodle лежить класичний вебстек LAMP (Linux, Apache, MySQL/MariaDB, PHP). Хоча можливі варіації, наприклад, використання PostgreSQL як бази даних або Nginx як вебсервера, ядро системи написане мовою PHP і функціонує як єдиний, цілісний застосунок.

Ключовою архітектурною особливістю Moodle, що забезпечує його гнучкість є модульність. Практично вся функціональність системи реалізована у вигляді

плагінів, які можна встановлювати, оновлювати та видаляти незалежно від основного ядра.

Основні типи плагінів:

- Модулі діяльності це основні навчальні інструменти курсу, такі як “Тест”, “Завдання”, “Форум”, “Лекція”, “Семінар”. Кожен з них є самостійним компонентом зі своєю логікою та таблицями в базі даних.
- Блоки включають в себе елементи інтерфейсу, що додають додаткову інформацію або функціональність на сторінки курсу, наприклад, “Календар”, “Останні оголошення”, “Прогрес виконання”.
- Типи запитань дозволяють розширювати функціонал тестів новими типами запитань, наприклад, “На відповідність”, “Перетягування маркерів”.
- Методи автентифікації це плагіни для інтеграції з зовнішніми системами ідентифікації.
- Теми визначають зовнішній вигляд та користувацький інтерфейс платформи.

Така архітектура дозволяє освітнім закладам глибоко налаштовувати систему під власті потреби, додаючи лише необхідний функціонал.

Moodle використовує власний шар абстракції для роботи з базами даних — Data Manipulation API. Це дозволяє розробникам писати код, що не залежить від конкретної СУБД (Система Управління Базами Даних) MySQL чи PostgreSQL, оскільки API сам транслює запити у відповідний SQL-діалект (Structured Query Language).[3]

Крім Data Manipulation Language, ядро системи надає низку інших важливих API для розробників плагінів:

- Access API керує правами доступу та ролями користувачів (студент, викладач, адміністратор), забезпечуючи структурний контроль над тим, хто і що може робити в системі.

- Gradebook API надає інтерфейс для взаємодії з журналом оцінок, дозволяючи модулям діяльності передавати до нього результати студентів.
- Moodle Web Services надає зовнішній REST/SOAP API, що дозволяє інтегрувати Moodle з іншими інформаційними системами університету, наприклад, для синхронізації списків студентів або оцінок.

Процес розробки Moodle кардинально відрізняється від корпоративного підходу Google і базується на принципах відкритої спільноти.

Основна команда розробників, що фінансується компанією Moodle Pty Ltd, відповідає за розробку та підтримку ядра системи, випуск оновлень безпеки та визначення загального напрямку розвитку продукту. Величезна кількість плагінів розробляється та підтримується прихильниками, університетами та комерційними компаніями по всьому світу. Це забезпечує широке різноманіття доступних інструментів.

Будь-який користувач може повідомити про помилку або запропонувати нову ідею через публічний Moodle Tracker. Розробники можуть пропонувати власний код для включення в ядро системи. Перед інтеграцією код проходить ретельну перевірку з боку спільноти та команди Moodle HQ. Moodle має чіткий і прогнозований цикл випуску нових версій, що дозволяє адміністраторам планувати оновлення своїх систем.

Таким чином, Moodle є прикладом класичної монолітної системи, сила якої полягає у гнучкості, керованості та можливості персоналізації завдяки модульній архітектурі та моделі розробки на основі відкритого коду. Водночас це накладає на освітню установу повну відповідальність за розгортання, підтримку та оновлення серверної інфраструктури.

Щоб продемонструвати внутрішню взаємодію компонентів у монолітній архітектурі Moodle, розглянемо процес проходження тексту студентом.

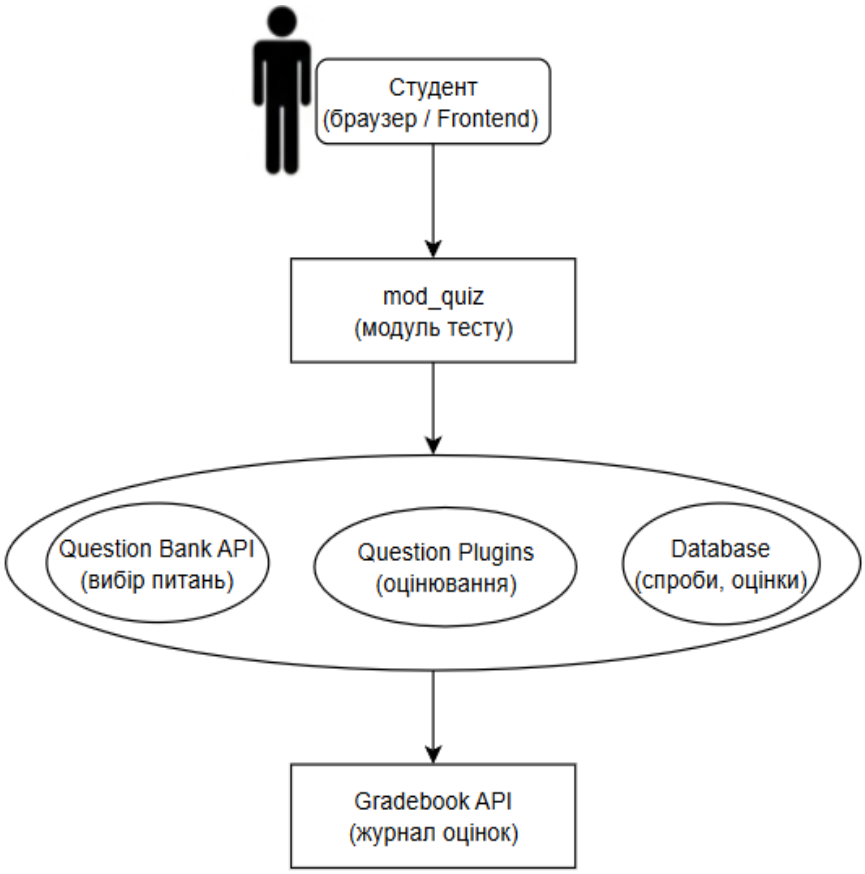


Рис.2.11 Внутрішня взаємодія компонентів Moodle

1. Ініціація: студент на сторінці курсу натискає на елемент “Тест”. Система ініціює модуль діяльності “Тест”.
2. Формування тесту: mod_quiz звертається до API банку питань. Він вибирає питання згідно з налаштуваннями тесту, наприклад, випадковий порядок, певна кількість питань з категорії тощо. Кожне питання є окремим плагіном типу запитань.
3. Відображення: сторінка з питаннями генерується на сервері та відправляється у браузер студента.
4. Відповідь та збереження: студент відповідає на питання і mod_quiz зберігає його спробу в таблицях бази даних, що відносяться до цього модуля.
5. Завершення та оцінювання: після завершення спроби mod_quiz ініціює процес оцінювання, він послідовно звертається до кожного плагіна типу питання, передаючи йому збережену відповідь студента. Кожен плагін

самостійно перевіряє правильність відповіді та повертає оцінку за своє питання.

6. Запис у журнал оцінок: mod_quiz збирає бали за всі питання, обчислює підсумкову оцінку за тест і через Gradebook API записує її у відповідну комірку журналу оцінок.

Цей сценарій показує, як різні, незалежно розроблені плагіни mod_quiz, плагіни питань, тісно взаємодіють всередині єдиного монолітного ядра через стандартизовані API, створюючи цілісний навчальний досвід.

Монолітна, модульна архітектура Moodle визначає його ключові переваги та недоліки в порівнянні з хмарними системами.

Сильні сторони

Освітні установи можуть не просто змінювати зовнішній вигляд, а й додавати унікальну функціональність через тисячі доступних плагінів або розробляти власні для задоволення специфічних педагогічних потреб.

Установа самостійно вирішує, де розміщувати сервери, в країні, в хмарі, на власних потужностях. Це критично важливо для дотримання законів про захист персональних даних та забезпечення інформаційної безпеки.

Платформа є безкоштовною, а її відкритий код гарантує, що установа не залежить від рішень однієї комерційної компанії. Вона може вільно змінювати хостинг-провайдерів та самостійно підтримувати систему.

Навколо Moodle сформувалася одна з найбільших у світі освітніх спільнот, що забезпечує постійний розвиток, підтримку на форумах та величезну базу документації.

Слабкі сторони

Весь тягар відповідальності за встановлення, налаштування, оновлення, резервне копіювання та безпеку лежить на плечах технічного персоналу установи. Це вимагає значних часових та фінансових ресурсів.

Як і будь-яка монолітна система, Moodle може мати проблеми з продуктивністю при великій кількості одночасних користувачів. Для оптимізації необхідно правильно налаштовувати кешування, балансування навантаження та оптимізувати конфігурацію сервера, що вимагає високої кваліфікації адміністраторів.

Процес оновлення ядра Moodle може бути незвичайним, особливо якщо встановлено багато сторонніх плагінів, які можуть виявитися несумісними з новою версією. Кожне оновлення вимагає ретельного тестування.

Хоча останні версії Moodle значно покращили користувацький інтерфейс, він все ще може сприйматися як менш інтуїтивний та сучасний у порівнянні з новітніми хмарними платформами, що створюються “з нуля”.

2.4 Архітектура та методи розробки Coursera

Coursera, як одна з провідних MOOC (Massive Open Online Course) платформ, спроєктована для обслуговування мільйонів користувачів по всьому світу. Її архітектура фундаментально відрізняється як від самодостатнього моноліту Moodle, так і від інтеграційного хабу Google Classroom. Вона є прикладом сучасної високомасштабованої хмарно-орієнтованої платформи, побудованої на основі мікросервісної архітектури та керованої даними.

Ядром Coursera є розподілена система, розгорнута на хмарній інфраструктурі, переважно Amazon Web Services (AWS). Замість єдиного застосунку, платформа складається з десятків незалежних мікросервісів, кожен з яких відповідає за свою бізнес-логіку.

Ключові архітектурні компоненти:

Coursera використовує різний підхід до розробки, де для кожного мікросервісу обирається найбільш відповідна технологія. Основною мовою для серверної частини є Scala, що працює на Java Virtual Machine з використанням фреймворку Play Framework. Такий вибір забезпечує високу продуктивність,

спільність та типізацію, що є критичним для високопродуктивних систем. Для frontend-частини використовується React, що дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси.[10]

Оскільки основний контент Coursera — це відео, ефективна його доставка є пріоритетом. Система використовує глобальні мережу доставки контенту Content Delivery Network (CDN), таку як Amazon CloudFront. Відеофайли зберігаються в сервісах типу Amazon S3 і кешується на серверах CDN, розташованих поблизу кінцевих користувачів. Це забезпечує низьку затримку та високу швидкість завантаження відео в будь-якій точці світу.[5]

Для автоматизації перевірки програмістських завдань Coursera розробила власну ізольовану “пісочницю”. Коли студент здає код, система автоматично розгортає безпечний Docker-контейнер із заздалегідь налаштованим середовищем, компілює та запускає код на наборі тестів. Результати виконання повертаються на платформу. Це забезпечує масштабовану та безпечну перевірку мільйонів завдань.

Аналітика та машинне навчання це серце платформи, Coursera збирає величезні обсяги даних про кожен клік, перегляд відео, відповідь на тест та повідомлення на форумі. Ці дані обробляються за допомогою інструментів для роботи з великими даними, таких як Apache Spark. На основі цього аналізу працюють:

- Системи рекомендації пропонують студентам нові курси на основі їхньої історії та інтересів.
- Персоналізація навчання, дані алгоритми можуть визначити з якими концепціями у студентів виникають труднощі, та надати підказки або додаткові матеріали.
- Предиктивна аналітика, коли моделі прогнозують ризик того, що студент покине курс, дозволяючи вчасно втрутитися.

Інженерна культура Coursera базується на принципах швидких ітерацій та рішень, що ґрунтується на даних. Зміни в коді після проходження автоматизованих

тестів одразу розгортаються для невеликої частики користувачів. Це дозволяє швидко впроваджувати інновації та виправляти помилки. Практично кожна нова функція або зміна в дизайні впроваджується як експеримент. Системі одночасно існують дві або більше версій продукту, які показуються різним групам користувачів. Інженери аналізують метрики, наприклад, відсоток завершення курсу, час на сайті і на основі статистичних даних приймають рішення, яка з версій є кращою. Управління складною хмарною інфраструктурою автоматизовано за допомогою інструментів, таких як Terraform або AWS CloudFormation. Це дозволяє інженерам описувати конфігурацію серверів, мереж та баз даних у вигляді коду, що забезпечує відтворюваність та надійність.

Coursera є прикладом продуктоорієнтованої платформи, де архітектура та процеси розробки тісно пов’язані з бізнес-цілями: залучення та утриманням мільйонної аудиторії. На відмінну від Moodle, тут пріоритетом є не гнучкість персоналізації, а масштабованість, надійність та оптимізація користувацького досвіду на основі аналізу даних.

З-поміж систем, де аналітика є вторинною функцією, в Coursera архітектура з самого початку будувалася навколо збору та обробки даних. Ця інфраструктура складається з кількох ключових рівнів:

1. Кожна взаємодія користувача з платформою — відтворення відео, пауза, відповідь на тест, клік по посиланню генерує подію. Ці події у реальному часі відправляються у централізований конвеєр даних, побудований на таких технологіях, як Apache Kafka. Це дозволяє надійно збирати мільярди подій щодня.
2. Зібрані “сирі” дані потрапляють у “озеро даних” на базі Amazon S3. Для структурованої обробки та аналізу використовуються інструменти, такі як Apache Spark, що дозволяють виконувати складні запити та запускати алгоритми машинного навчання на терабайтах даних.
3. Оброблена інформація використовується кількома ключовими мікросервісами:

- Сервіс рекомендацій будує профілі користувачів та пропонує їм актуальні курси.
- Сервіс персоналізації може адаптувати терміни здачі завдань або пропонувати додаткові матеріали на основі аналізу успішності студента.
- Сервіс аналітики для партнерів надає університетам-партнерам детальні звіти про те, як студенти проходять їхні курси, де виникають труднощі та який рівень залученості.

Цей підхід, де кожен компонент системи є одночасно і споживачем, і генератором даних, є фундаментальною відмінністю архітектури Coursera.

Мікросервісна, хмарна та керована даними архітектура забезпечує Coursera значні конкурентні переваги, але водночас має свої інженерні виклики.

Сильні сторони

Архітектура спроєктована для обслуговування глобальної аудиторії. Використання хмарних сервісів дозволяє автоматично масштабувати лише ті мікросервіси, які зазнають пікового навантаження, наприклад, сервіс реєстрації на початку тижня, що є економічно ефективним.

Незалежність мікросервісів дозволяє різним командам розробників працювати паралельно над різними частинами продукту. Завдяки A/B тестуванню та CI/CD, компанія може перевіряти десятки гіпотез одночасно та впроваджувати лише ті зміни, які доведено покращують користувацький досвід.

Глибокий аналіз поведінки користувачів дозволяє створити значно більш важливий та привабливий навчальний досвід, ніж це можливо в системах, що не мають доступу до таких великих обсягів даних.

Збій в одному з неосновних мікросервісів, наприклад, у сервісі рекомендацій не вплине на базову функціональність платформи, таку як доступ до навчальних матеріалів.

Слабкі сторони

Управління десятками взаємодіючих сервісів є значно складнішим, ніж підтримка одного монолітного застосунку. Це вимагає високої кваліфікації інженерів DevOps, складних систем моніторингу, логування та трасування запитів.

У розподіленій системі, де кожен мікросервіс може мати власну базу даних, забезпечення узгодженості даних між ними є нетривіальною задачею. Наприклад, потрібно гарантувати, що після видалення курсу інформація про цього зникне з усіх пов’язаних сервісів.

Платформа пропонує єдиний, стандартизований досвід для всіх. На відміну від Moodle, університет-партнер не може глибоко налаштовувати навчальний процес, додати унікальні типи завдань або змінити логіку оцінювання. Усі змушені працювати в межах тих інструментів, які надає Coursera.

Хоча хмарні сервіси є гнучкими, підтримка інфраструктури такого масштабу, особливо обробка великих даних, вимагає значних фінансових інвестицій.

2.5 Вибір методу для розробки системи

У цьому розділі було проведено глибокий аналіз методологій проєктування та архітектурних рішень, що лежать в основі сучасних систем дистанційного навчання.

Виходячи з цього, для розробки кваліфікаційної роботи використовується гібридний архітектурний підхід. Цей підхід має на меті поєднати найкращі риси проаналізованих систем:

1. Від Moodle береться ідея контролю та гнучкості, система буде реалізована не як жорсткий моноліт, а як легкий, керований клієнт-серверний застосунок, що надає повний контроль над бізнес-логікою та навчальними матеріалами.
2. Від Google береться принцип функціональної декомпозиції для інтелектуальних завдань. Замість спроби вбудувати складну AI-логіку

безпосередньо в основний код, як це було б у Moodle, інтелектуальний модуль виділяється в окрему логічну підсистему.

Ця підсистема, у свою чергу, буде використовувати сучасний підхід, споживаючи потужності зовнішніх API, таких як Google Gemini, що є аналогією до того, як Classroom використовує API Google Drive. Це дозволяє отримати найвищу якість генерації тексту, уникнувши при цьому необхідність розгортати та підтримувати власні обчислювально-важкі моделі.

Отже, в основі проєкту лежатиме гібридна архітектура — керований та гнучкий backend-застосунок, що відповідає за основні функції платформи, та функціонально відокремлений інтелектуальний модуль, що реалізує логіку RAG та взаємодіє із зовнішніми хмарними AI-сервісами. Таке поєднання забезпечує необхідний баланс між гнучкістю як у Moodle та сучасною інтелектуальною потужністю Google, що дозволить ефективно вирішити проблему відсутності миттєвої підтримки студентів. Саме ця архітектура буде детально спроєктована у наступному розділі.

3. РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ

Розробка архітектури є фундаментальним етапом проєктування, що визначає структуру, компоненти та принципи взаємодії в інформаційній системі. Для кваліфікаційної роботи застосовано структурний підхід до проєктування, що базується на принципах функціональної декомпозиції та чіткому визначенні потоків даних.

3.1 Вибір архітектурного стилю

Для проєктування архітектури інтелектуальної інформаційної системи було обрано комбінований підхід, що базується на клієнт-серверному архітектурному стилі із застосуванням принципів функціональної декомпозиції.

Вибір клієнт-серверного стилю є фундаментальним для будь-якого сучасного вебзастосунку.

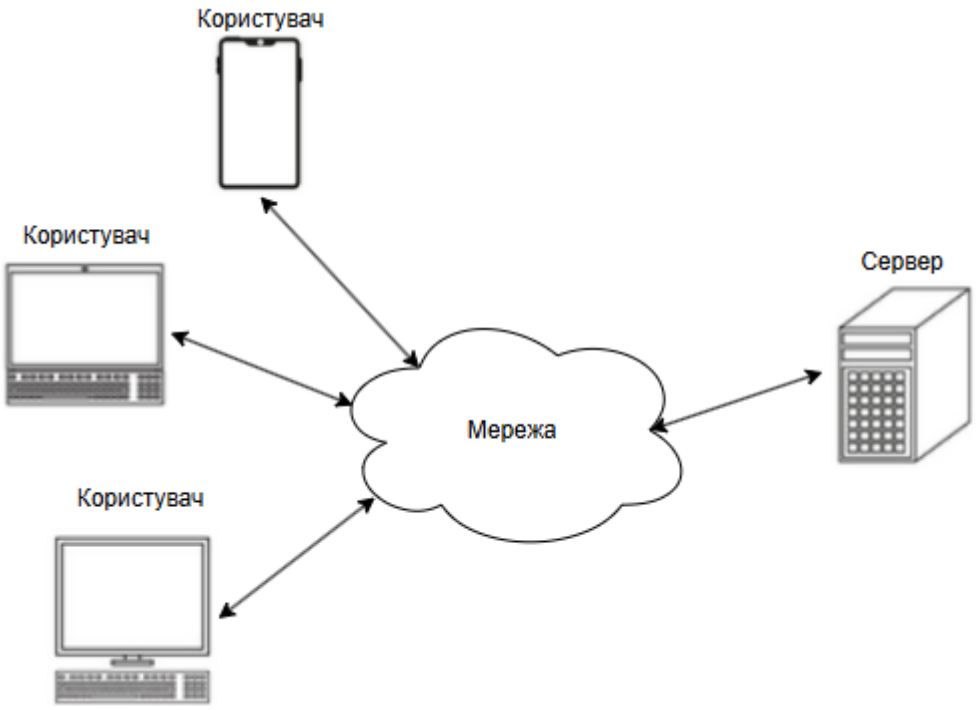


Рис.3.1 Клієнт-серверний стиль

Він передбачає чітке розподілення відповідальності у системі:

- Клієнт — це підсистема візуалізації, що виконується у веббраузері користувача. Її відповідальність полягає у відображенні інтерфейсу та передачі запитів користувача на сервер.
- Сервер — це підсистема бізнес-логіки, що відповідає за обробку запитів, взаємодію з даними та виконання основних операцій.

На відміну від класичного об’єктно-орієнтованого проєктування, де система моделюється як сукупність взаємодіючих об’єктів, на високому архітектурному рівні застосовано структурний підхід у вигляді функціональної декомпозиції. Це дозволило логічно розділити всю систему на дискретні, функціонально завершені підсистеми, що відповідають за конкретні задачі:

- Підсистема візуалізації (Frontend).
- Підсистема управління бізнес-логікою (Backend).
- Інтелектуальна підсистема (AI Core).

Такий підхід є особливо виправданим для даної задачі. Інтелектуальний компонент є унікальним, обчислювально-важким процесом таким як, пошук за схожістю та генерація тексту, який логічно відокремлений від стандартних CRUD-операцій (Create, Read, Update, Delete), як-от автентифікація чи управління курсами. Таким чином, клієнт-серверна модель визначає спосіб взаємодії компонентів, а функціональна декомпозиція — принцип, за яким ці компоненти були виділені.

3.2 Архітектурна модель

Архітектурна модель системи визначає логічні компоненти, з яких вона складається, та описує їхню зону відповідальності. Відповідно до обраного підходу функціональної декомпозиції розглянутого в пункті 3.1, система логічно розділена на чотири ключові підсистеми, що виконують чітко визначені задачі.

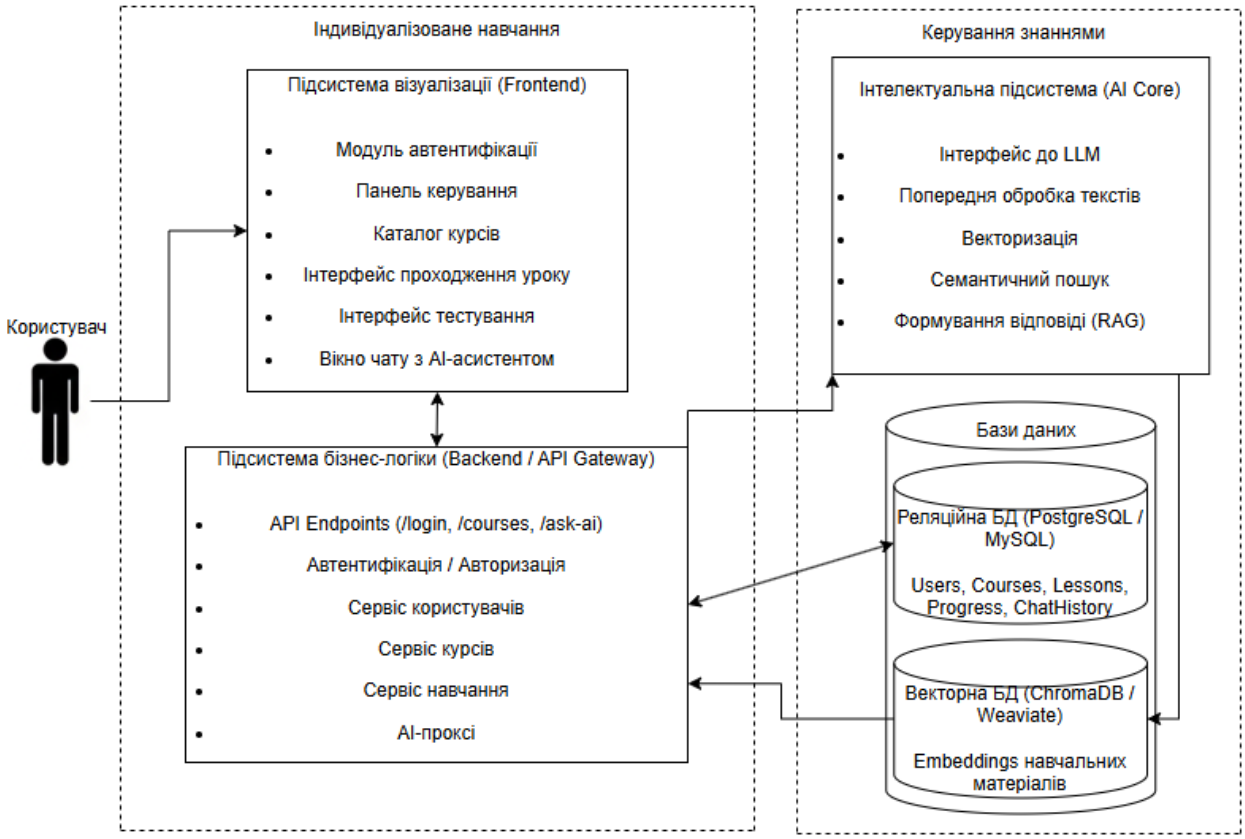


Рис.3.2 Архітектурна модель системи

Зв’язки між підсистемами:

- Frontend → Backend:
  запити користувача (логін, отримання курсів, AI-запит);
- Backend → Бази даних (реляційне сховище):
  CRUD-операції з користувачами, курсами, файлами;
- Backend → AI Core:
  передача AI-запитів, наприклад, питання студента;
- AI Core → Векторне сховище:
  збереження та пошук векторів знань;
- AI Core → Backend → Frontend:
  зворотний шлях відповіді для відображення користувачу.

3.2.1 Підсистема візуалізації (Frontend)

Підсистема візуалізації є клієнтською частиною архітектури у клієнт-серверній моделі. Вона виконується безпосередньо у середовищі веббраузера кінцевого користувача.

Основна відповідальність цієї підсистеми — презентація даних та забезпечення взаємодії з користувачем. Вся бізнес-логіка, така як перевірка прав доступу, автентифікація чи обробка AI-запитів, винесена на серверну частину.

Функціонально, підсистема візуалізації складається з наступних логічних блоків:

- Модуль автентифікації надає користувачу інтерфейси для входу та реєстрації в системі.
- Навігаційна панель забезпечує основну навігацію системою. Відображає список доступних курсів для студента або інструменти керування курсами для викладача.
- Область відображення контенту це компонент, відповідальний за конкретне відображення навчальних матеріалів, наприклад, лекцій, статей, зображень.
- Інтерфейс AI-асистента є ключовим компонентом у вигляді інтерактивного чат-віджету. Його єдине завдання це приймати текстові запити від студента та відображати відповіді, що надходять від серверної частини.
- Інтерфейс керування (для викладача) це набір спеціалізованих форм та елементів керування, що дозволяють викладачу завантажувати нові навчальні матеріали та керувати ними.

3.2.2 Підсистема бізнес-логіки (Backend)

Підсистема бізнес-логіки є центральним сервером у клієнт-серверній моделі. Вона виступає як “мозок” системи, що керує всіма основними процесами, та є посередником між підсистемою візуалізації та іншими, більш спеціалізованими підсистемами.

Відповідно до принципу функціональної декомпозиції, на цю підсистему покладено наступні завдання:

- Управління автентифікацією та сесіями це перевірка облікових даних користувачів, що надходять від клієнта, та управління життєвим циклом сесій.
- Авторизація та контроль доступу є визначення прав користувача, наприклад, “Студент” чи “Викладач” та надання доступу до відповідних ресурсів чи функцій.
- Обробка бізнес-логіки включає в себе виконання всіх стандартних операцій системи, що не пов’язані з AI. Наприклад, зарахування студента на курс, отримання списку лекцій, збереження метаданих про завантажені файли.
- Інтерфейс доступу до даних є наданням абстрактного інтерфейсу для взаємодії з підсистемою даних.
- Делегування AI-запитів є критично важливим архітектурним рішення, так-як ця підсистема не виконує обчислювально-важливих завдань штучного інтелекту. Її роль — отримати запит від клієнта, перевіряти його та перенаправити до спеціалізованої інтелектуальної підсистеми.

Таким чином, підсистема бізнес-логіки охоплює всю логіку управління та координації, залишаючись при цьому легкою та незалежною від специфічних обчислювальних завдань AI.

3.2.3 Інтелектуальна підсистема (AI Core)

Підсистема інтелектуального ядра є ключовим інноваційним компонентом спроєктованої архітектури. Вона реалізується як функціонально відокремлений, спеціалізований сервіс, призначений виключно для виконання обчислювально-складних завдань, пов’язаних зі штучним інтелектом. Винесення цієї логіки в окрему підсистему є принциповим архітектурним рішенням, що дозволяє уникнути перевантаження основної підсистеми бізнес-логіки.

Функціональні обов’язки інтелектуального ядра чітко розділені на два основні процеси:

1. Управління базою знань:
    - Отримання нових навчальних матеріалів від підсистеми бізнес-логіки.
    - Попередня обробка текстових даних: очищення та розбиття на логічні фрагменти.
    - Векторизація фрагментів тексту за допомогою моделі векторизації.
    - Збереження отриманих векторних представлень у спеціалізованому векторному сховищі.
2. Обробка запитів:
    - Отримання запиту користувача та ідентифікатора курсу від підсистеми бізнес-логіки.
    - Виконання семантичного пошуку у векторному сховищі для знаходження найбільш релевантних фрагментів навчальних матеріалів.
    - Формування кінцевого запиту до великої мовної моделі, що включає знайдений контекст та оригінальне питання.
    - Отримання та повернення згенерованої текстової відповіді підсистемі бізнес-логіки.

Ізоляція цього модуля в окрему функціональну одиницю забезпечує високу гнучкість системи, дозволяючи незалежно масштабувати обчислювальні ресурси, необхідні для AI без впливу на стабільність основних функцій вебзастосунку.

3.2.4 Підсистема даних

Підсистема даних є абстрактним шаром, що відповідає за надійне зберігання, управління та доступ до всієї інформації, необхідної для функціонування системи. На етапі проєктування архітектури ця підсистема логічно розділена на два

фундаментально різних типи сховищ, що оптимізовані для вирішення специфічних завдань:

1. Реляційне сховище — це сховище призначене для зберігання всієї структурованої, транзакційної інформації. Воно оперує сутностями та зв’язками між ними.
    - Управління профілями користувачів, їхніми ролями, каталогом навчальних курсів, зв’язками зарахування, а також зберігання метаданих про навчальні матеріали.
    - Доступ до цього сховища здійснюється виключно через підсистему бізнес-логіки, яка виконує операції CRUD за допомогою структурованих запитів.
2. Векторне сховище — це вузькоспеціалізоване, нереляційне сховище, оптимізоване для виконання операцій швидкого пошуку за схожістю у багатовимірному просторі.
    - Зберігання векторних представлень текстових фрагментів навчальних матеріалів. Це сховище є “пам’яттю” для інтелектуальної підсистеми.
    - Доступ до цього сховища здійснюється виключно інтелектуальною підсистемою. Підсистема бізнес-логіки не має прямого доступу до цього сховища, вона лише ініціює процеси індексації або запитів, які AI Core виконує самостійно.

Таке гібридне розділення сховищ даних є критично важливим, оскільки дозволяє використовувати для кожної задачі найбільш ефективний інструмент: надійну реляційну модель для структурованих даних та високопродуктивну векторну модель для семантичного пошуку AI.

3.2.5 Інфраструктурна модель

Інфраструктурна модель визначає абстрактну фізичну топологію системи — як логічні підсистеми, розподілені по фізичних або віртуальних вузлах. Проєктована інфраструктура оптимізована для вебзастосунку, який поєднує стандартну бізнес-логіку з RAG-функціоналом, що звертається до зовнішніх AI-сервісів. Модель складається з двох основних серверних вузлів.

Вузол застосунку є єдиною точкою входу для всіх клієнтських запитів. Цей сервер інфраструктурно відповідає за виконання трьох логічних підсистем:

1. Підсистема візуалізації — це вузол, який зберігає та віддає клієнтам статичні файли, що формують користувацький інтерфейс.
2. Підсистема бізнес-логіки, на цьому ж вузлі виконується серверний код, що обробляє API-запити: автентифікацію, авторизацію, управління курсами тощо.
3. Логіка інтелектуальної підсистеми, оскільки важке завдання генерації тексту виконується зовнішнім API-сервісом, немає потреби у виділенні окремого потужного обчислювального вузла. Тому логіка AI Core також виконується на цьому вузлі застосунку як частина серверної логіки.

Таке об’єднання є виправданим, оскільки RAG-пошук є операцією, що навантажує переважно сервер баз даних, а не вебсервер.

Вузол даних є виділеним сервером, оптимізованим для операцій високошвидкісного читання/запису та зберігання даних. Він інфраструктурно обслуговує обидна логічні сховища:

1. Реляційне сховище, тут розміщується СУБД, що керує структурованими даними. Вона оптимізована для транзакційних запитів, які надходять від підсистеми бізнес-логіки.
2. Векторне сховище, тут розміщується спеціалізована векторна база даних. Вона оптимізована для обробки запитів пошуку за схожістю, які надходять від логіки інтелектуальної підсистеми.

Розділення вузла застосунку та вузла даних є класичним рішенням, що дозволяє незалежно масштабувати обчислювальні ресурси та ресурси зберігання, а також підвищує загальну безпеку системи.

3.3 Проєктування бази даних

База даних (БД) є фундаментальним компонентом інформаційної системи, що відповідає за структуроване зберігання, управління та доступ до всієї інформації. Архітектура сховища даних для розроблюваної системи є гібридною, оскільки вона поєднує два типи баз даних для вирішення різних завдань: реляційну СУБД для структурованих даних та векторну БД для забезпечення ефективного семантичного пошуку.

Вибір гібридного підходу зумовлений тим, що реляційні та векторні бази даних спроєктовані для принципово різних завдань і є взаємодоповнюючими, а не конкуруючими технологіями.

Таблиця 3.1 Порівняльна таблиця реляційної та векторної моделей даних

Критерій	**Реляційна СУБД**	**Векторна БД**
Призначення	Зберігання структурованих даних зі зв’язками.	Зберігання та пошук багатовимірних векторів.
Структура	Таблиці (рядки, стовпці).	Колекції векторів (ID, метадані).
Операції	Точний пошук за значеннями (SELECT, WHERE), об’єднання (JOIN), агрегація (GROUP BY).	Пошук за схожістю (k-NN), семантичний пошук.
Приклад запиту	SELECT * FROM Users WHERE email = ‘test@example.com’	collection.query (query_embeddings=[vector], n_results=5)
Застосування в проєкті	Управління користувачами, курсами, зарахування; зберігання метаданих файлів.	Індексація та швидкий пошук фрагментів навчальних матеріалів для AI-асистента.

3.3.1 Логічна модель даних

На етапі логічного проєктування визначаються ключові сутності, їхні атрибути та зв’язки між ними. Для цього використовується модель “сутність-зв’язок”.

Рис.3.3 Логічна модель даних

Основні сутності:

- Users (Користувачі): зберігає інформацію про всіх зареєстрованих користувачів.
  Атрибути: user_id (первинний ключ), name, role_id.
- Roles (Ролі): довідник ролей у системі, наприклад, “student”, “teacher”.
  Атрибути: role_id (первинний ключ), name.
- Courses (Курси): зберігає інформацію про навчальні курси.
  Атрибути: course_id (первинний ключ), title, inctructor_id (зовнішній ключ до Users).
- Enrollments (Зарахування): таблиця зв’язку, що показує, який студент зарахований на який курс.
  Атрибути: enrollment_id, user_id (зовнішній ключ), course_id (зовнішній ключ).
- Materials (Навчальні матеріали): зберігає метадані про завантажені файли.
  Атрибути: material_id (первинний ключ), title, course_id (зовнішній ключ).

Зв’язки:

- Один User може мати одну Role зв’язок “один до одного”.
- Один User (викладач) може бути автором багатьох Courses “один до багатьох”.
- Багато Users (студентів) можуть бути зараховані на багато Courses “багато до багатьох”, реалізується через таблицю Enrollments.
- Один Course може мати багато Materials “один до багатьох”.

3.3.2 Фізична модель даних

Нижче наведено SQL-код для створення таблиць, який відповідає логічній моделі.

1. Створення таблиць

Спочатку створюються “батьківські” таблиці (roles), потім ті, що на них посилаються.

```
-- Таблиця "Ролі" (Довідник)
CREATE TABLE roles (
    role_id SERIAL PRIMARY KEY,
    role_name VARCHAR(50) NOT NULL UNIQUE
);
-- Таблиця "Користувачі"
-- Посилається на 'roles'
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role_id INT NOT NULL,
```

```
    CONSTRAINT fk_user_role
        FOREIGN KEY(role_id)
        REFERENCES roles(role_id)
);
-- Таблиця "Курси"
-- Посилається на 'users' (для викладача)
CREATE TABLE courses (
    course_id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    teacher_id INT NOT NULL,
    CONSTRAINT fk_course_teacher
        FOREIGN KEY(teacher_id)
        REFERENCES users(user_id)
        ON DELETE RESTRICT -- Заборонити видалення викладача, якщо у нього
є курси
);
-- Таблиця "Зарахування" (Зв'язок M:M між users та courses)
-- Посилається на 'users' та 'courses'
CREATE TABLE enrollments (
    enrollment_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    course_id INT NOT NULL,
    CONSTRAINT fk_enrollment_user
        FOREIGN KEY(user_id)
```

```
        REFERENCES users(user_id)
       ON DELETE CASCADE, -- Видалити зарахування, якщо видалено користувача
    CONSTRAINT fk_enrollment_course
        FOREIGN KEY(course_id)
        REFERENCES courses(course_id)
        ON DELETE CASCADE, -- Видалити зарахування, якщо видалено курс
    -- Гарантує, що студент не може бути зарахований на той самий курс двічі
    UNIQUE(user_id, course_id)
);
-- Таблиця "Навчальні матеріали"
-- Посилається на 'courses'
CREATE TABLE materials (
    material_id SERIAL PRIMARY KEY,
    course_id INT NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    file_path VARCHAR(1024) NOT NULL UNIQUE, -- Шлях у файловій системі або
S3
    upload_date TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_material_course
        FOREIGN KEY(course_id)
        REFERENCES courses(course_id)
        ON DELETE CASCADE -- Видалити матеріали, якщо курс видалено
);
```

2. Створення індексів

Індекси потрібні для прискорення пошуку, особливо за зовнішніми ключами, які часто використовуються в JOIN та WHERE.

```
-- Індекси для зовнішніх ключів
CREATE INDEX idx_users_role_id ON users(role_id);
CREATE INDEX idx_courses_teacher_id ON courses(teacher_id);
CREATE INDEX idx_enrollments_user_id ON enrollments(user_id);
CREATE INDEX idx_enrollments_course_id ON enrollments(course_id);
CREATE INDEX idx_materials_course_id ON materials(course_id);
```

3.3.3 Архітектура модуля знань (Векторна БД)

Для функціонування AI-асистента система реалізує архітектуру RAG. Цей підхід вимагає окремого проєктування двох ключових процесів: наповнення (індексації) бази знань та отримання відповіді (виконання RAG-запиту).

1. Процес наповнення (індексація даних)

Це процес “збирання” векторної бази даних, який відбувається асинхронно кожного разу, коли до курсу додаються нові Materials, наприклад, викладач завантажує PDF-файл. Він складається з таких кроків:

1. Система витягує чистий текст із завантажених файлів (PDF, DOCX, тощо), ігноруючи зображення та форматування.
2. Текст розбивається на невеликі, логічно завершені фрагменти (чанки) розміром 500-1000 символів. Це робиться для того, щоб семантичний пошук повертав максимально релевантні уривки, а не цілі документи.
3. Кожен фрагмент тексту подається на вхід спеціальній нейромережі-ембедера, яка перетворює його на вектор — числовий масив фіксованої довжини, наприклад, 768 чисел. Цей вектор є математичним представленням семантичного змісту тексту.

4. Кожен вектор разом з оригінальним текстом та метаданими зберігається у колекції ChromaDB.

2. Структура запису у векторній БД

Кожен запис у векторній базі даних ChromaDB містить:

- Вектор: наприклад, масив з 768 чисел — семантичний “відбиток” тексту.
- Контент: string — оригінальний текстовий фрагмент, який буде показаний LLM.
- Метадані: JSON — критично важлива інформація для фільтрації. Це гарантує, що студент отримає відповідь лише на основі матеріалів, до яких він має доступ. Приклад метаданих: {“course_id”: 15, “material_id”: 101 “page”: 3}.

3. Процес отримання відповіді (RAG-запит)

Коли студент ставить питання AI-асистенту, запускається наступний ланцюжок:

1. Векторизація запиту: питання студента, наприклад, “Що таке патерн Фабрика?” перетворюється на вектор за допомогою тієї ж моделі-ембедера.
2. Пошук схожості: система виконує запит до ChromaDB для пошуку K-найближчих сусідів (k-NN) — тобто, пошуку текстових фрагментів, вектори яких є “найближчими” до вектора запитання.
3. Оцінка схожості: для вимірювання “близькості” векторів використовується косинусна подібність.

Однак, якщо говорити про математичні операції, що виконуються з цими векторами, для оцінки семантичної подібності використовують так звану косинусну подібність. Формула косинусної подібності для двох векторів $\vec{A}$ та $\vec{B}$ виглядає так:

$$\text{подібність}\,(\vec{A},\vec{B}) = \cos(\theta) = \frac{\vec{A}\cdot\vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

де:

$\vec{A} \cdot \vec{B}$ — скалярний добуток векторів $\vec{A}$ та $\vec{B}$. Він обчислюється як сума добутків відповідних координат: $\sum_{i=1}^{n} A_i B_i$

$\left\|\vec{A}\right\|$ — норма (довжина) вектора $\vec{A}$. Вона обчислюється як корінь квадратний із суми квадратів його координат: $\sqrt{\sum_{i=1}^{n} A_i^2}$

$\left\|\vec{B}\right\|$ — норма вектора $\vec{B}$

Косинусна подібність вимірює косинус кута між двома векторами в багатовимірному просторі. Чим менший кут, тим ближчі за змістом вектори, і тим вище значення косинуса. Якщо вектори вказують в одному напрямку, косинус дорівнюватиме 1 (максимальна подібність). Наприклад, якщо маємо два речення і їхні векторні представлення, можемо обчислити косинусну подібність між ними. Якщо результат близький до 1, речення мають схожий семантичний зміст.[2]

4. Фільтрація: система до або після пошуку фільтрує результати метаданими, щоб переконатися, що студент зарахований на курс до якого належить знайдений матеріал.
5. Формування промпту: знайдені текстові фрагменти (контекст) та питання студента збираються у фінальний промпт для LLM:

```
[СИСТЕМНИЙ ПРОМПТ]

Ти - корисний AI-асистент для студентів. Твоя задача - відповідати на питання, базуючись виключно на наданому RAG нижче контексті. Якщо відповіді в контексті немає, скажи про це прямо.

[КОНТЕКСТ]

{тут вставляються текстові чанки, знайдені у векторній базі даних}

[ПИТАННЯ СТУДЕНТА]

{тут вставляється оригінальне питання користувача}
```

Така структура змушує модель генерувати відповідь на основі перевірених джерел (матеріалів курсу), що мінімізує ризик “галюцинацій”.

3.3.4 Вибір СУБД та фізичне проєктування

Реляційна СУБД, для зберігання структурованих даних обрано PostgreSQL. Це потужна об’єктно-реляційна СУБД з відкритим вихідним кодом, яка має наступні переваги:

- Гарантує цілісність даних.
- Підтримує складні типи даних та має розширення, наприклад, PostGIS для геоданих.
- Добре оптимізована для обробки складних запитів.

Векторна СУБД, для зберігання векторних представлень обрано ChromaDB. Це спеціалізована векторна база даних з відкритим кодом, оптимізована для надшвидкого пошуку схожості. Вона ідеально підходить для реалізації RAG-системи.

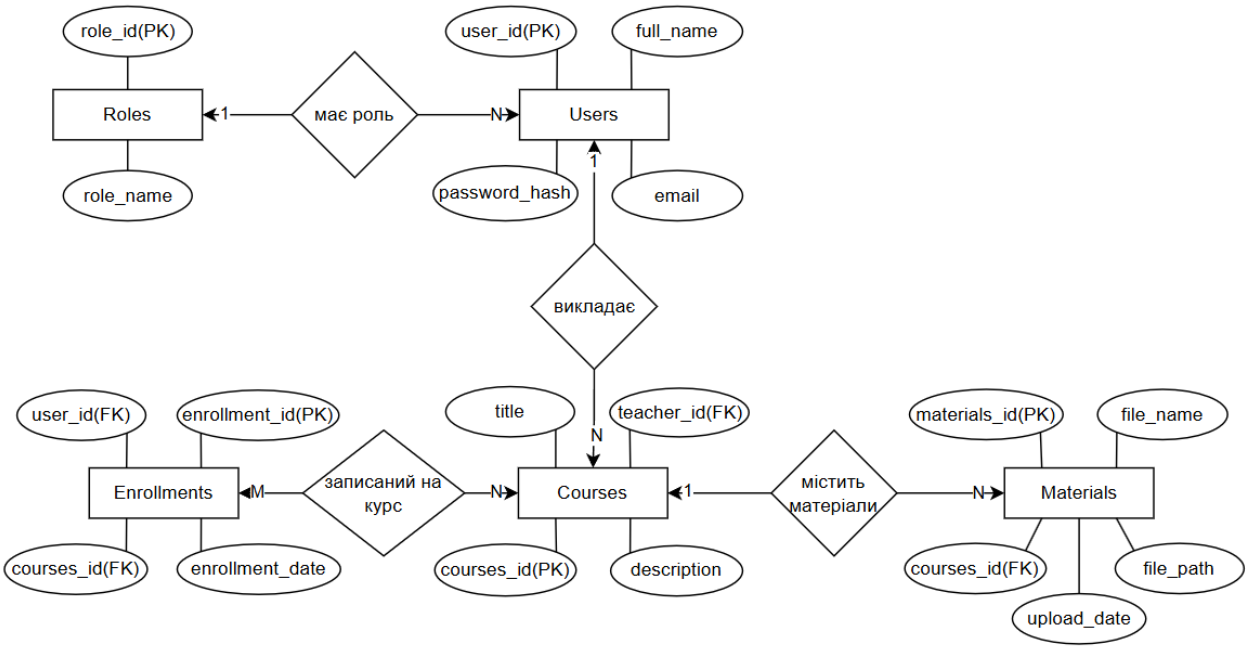


Рис.3.4 Діаграма Чена

Зв’язки між сутностями (у форматі Чена):

1. Roles — Users
   1 роль → багато користувачів
   Зв’язок: “має роль”
   Тип: (1 : N)
2. Users — Courses
   1 користувач (викладач) → багато курсів
   Зв’язок: “викладає”
   Тип: (1 : N)
3. Users — Courses (через Enrollments)
   Багато користувачів ↔ багато курсів
   Зв’язок: “записаний на курс”
   Реалізований через таблицю Enrollments
   Тип: (M : N)
4. Courses — Materials
   1 курс → багато матеріалів
   Зв’язок: “містить матеріали”
   Тип: (1 : N)

Ця структура забезпечує надійне зберігання основних даних системи та створює фундамент для подальшої розробки бізнес-логіки.

3.4 Поведінкова модель та сценарії взаємодії

Поведінкова модель описує динаміку системи, фокусуючись на тому, як зовнішні актори взаємодіють з нею для досягнення своїх цілей. Ця модель реалізується за допомогою діаграм варіантів використання та деталізованих текстових сценаріїв.

3.4.1 Ідентифікація акторів

У спроєктованій системі виділено два основних актори:

1. Студент — кінцевий споживач освітнього контенту. Його основна мета це доступ до матеріалів та отримання інтелектуальної підтримки.
2. Викладач — актор, відповідний за створення та управління навчальним контентом, який слугуватиме базою знань для інтелектуальної підсистеми.

3.4.2 Деталізація ключових сценаріїв

Для глибокого розуміння архітектурної взаємодії компонентів, розглянемо два ключові сценарії, що ілюструють роботу основних акторів.

Сценарій 1. Завантаження навчального матеріалу, актор — викладач.

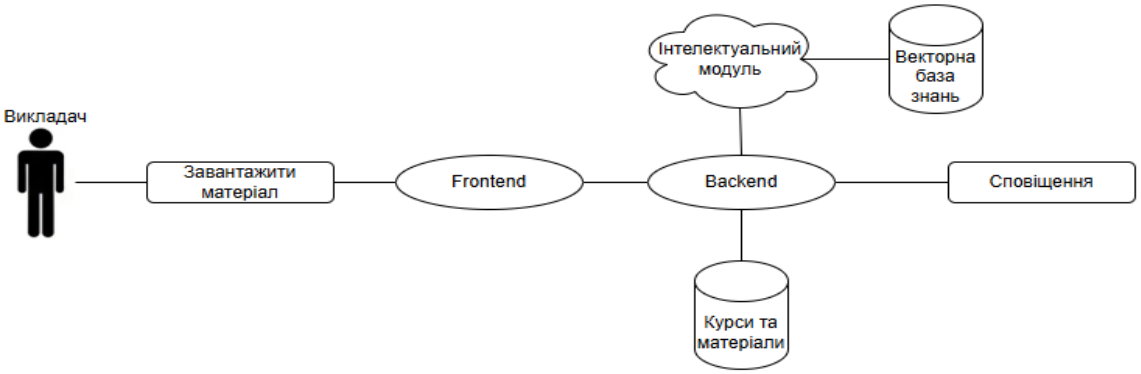


Рис.3.5 Приклад сценарію 1

Цей сценарій описує процес наповнення бази знань, що є критично важливим для подальшої роботи інтелектуальної підсистеми.

Передумова: актор “Викладач” авторизований у системі та знаходиться на сторінці керування курсом.

Основний потік:

1. Викладач обирає опцію “Завантажити матеріал”.
2. Підсистема візуалізації відображає форму для завантаження файлу.
3. Викладач обирає файл та ініціює завантаження.
4. Frontend відправляє файл на підсистему бізнес-логіки.
5. Backend зберігає файл у реляційному сховищі.

6. Backend асинхронно ініціює процес індексації — надсилає команду та дані файлу до інтелектуальної підсистеми.
7. AI Core виконує обробку та зберігає вектори у векторному сховищі.
8. Система сповіщає викладача про успішне завантаження та індексацію.

Сценарій 2. Постановка питання AI-асистенту, актор — студент.

Цей сценарій демонструє основний інноваційний функціонал системи отримання інтелектуальної відповіді, базованої на матеріалах курсу.

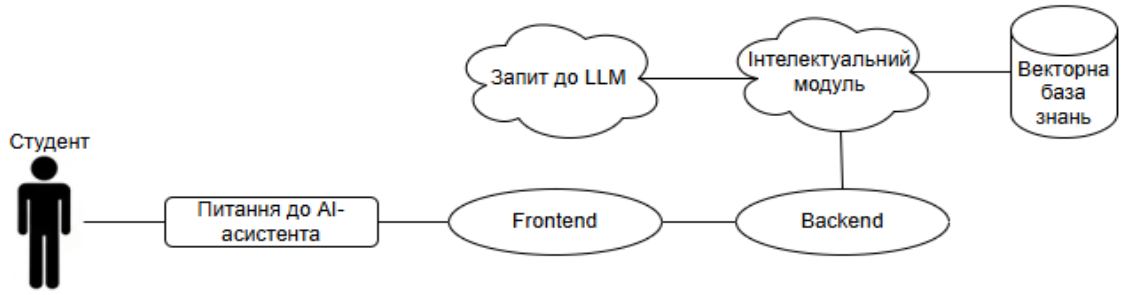


Рис.3.6 Приклад сценарію 2

Передумова: актор “Студент” авторизований та знаходиться на сторінці курсу.

Основний потік:

1. Студент вводить питання в інтерфейс AI-асистента.
2. Frontend відправляє запит на Backend.
3. Backend перевіряє запит та перенаправляє його до інтелектуальної підсистеми.
4. AI Core виконує семантичний пошук у векторному сховищі, знаходячи релевантні текстові фрагменти.
5. AI Core формує запит до зовнішньої LLM (Gemini), що включає знайдені фрагменти та питання студента.
6. LLM генерує відповідь.
7. Відповідь повертається через AI Core та Backend на Frontend, де відображається студенту в інтерфейсі чату.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є стратегічним рішенням, що безпосередньо впливає на продуктивність, масштабованість, швидкість розробки та подальшу підтримку інформаційної системи. Обраний стек — Astro, TypeScript, TailwindCSS та PostgreSQL — був підібраний для досягнення оптимального балансу між продуктивністю статичного контенту та динамічною функціональністю full-stack застосунку.

Для реалізації системи обрано Astro — сучасний full-stack вебфреймворк, що базується на концепції Astro Islands та архітектурі нульовий JavaScript за замовчуванням.

Обґрунтування вибору:[22]

- Astro за замовчуванням генерує статичні HTML-сторінки (HyperText Markup Language), що забезпечує максимальну швидкість завантаження. Інтерактивні компоненти завантажують JavaScript лише за потреби, мінімізуючи час до інтерактивності.
- Astro дозволяє створювати повноцінні API-маршрути на стороні сервера. Це усуває необхідність у розгортанні окремого сервера для простих CRUD-операцій, оскільки Astro може безпосередньо взаємодіяти з базою даних для обробки динамічних запитів.
- Фреймворк має нативну підтримку файлів .mdx (Markdown з JSX-компонентами), що ідеально підходить для створення та управління навчальним контентом, наприклад, лекціями, статтями без необхідності у складній системі керуванням вмістом.

Таблиця 4.1 Порівняння з альтернативами

Фреймворк	**Архітектура**	**Перевага**	**Недолік у контексті проєкту**
Astro	Static-first (MPA) + Islands.	Найвища продуктивність, простота API.	Менш підходить для високоінтерактивних “SPA-подібних” дошок.
Next.js	React-centric (SPA/SSR).	Потужна екосистема, гнучкий рендеринг.	Більший обсяг JS за замовчуванням, складніший для контент-сайтів.
Traditional (Django/RoR)	Monolithic MVC.	Зрілість, ORM.	“Важкий” стек, повільніша розробка UI, надлишковий для даної задачі.

Вибір Astro є оптимальним, оскільки система поєднує великий обсяг статичного навчального контенту з динамічними функціями.

Уся кодова база проєкту як Frontend, так і Backend логіка в Astro реалізована з використанням TypeScript. TypeScript є строго типізованим надмножиною JavaScript. Його використання в проєкті такого масштабу має критичні переваги, такі як:[18]

- Статична типізація дозволяє виявляти великий клас помилок на етапі компіляції, а не під час виконання.
- Чітка типізація робить код самодокументованим та значно спрощує перебудову коду і підтримку.
- Забезпечує кращу автодоповнення коду та навігацію.

Використання TypeScript замість нативного JavaScript є інвестицією в надійність та довгострокову стабільність програмного продукту.

Для стилізації користувацького інтерфейсу обрано TailwindCSS — CSS-фреймворк, що базується на підході utility-first. На відміну від компонентних бібліотек, які надають готові, але складні для налаштування компоненти, Tailwind надає набір низькорівневих “утилітарних” класів.[17]

- Дозволяє будувати складні та унікальні дизайни безпосередньо в HTML/JSX-розмітці, не покидаючи контексту.
- Усі відступи, кольори та шрифти визначені в конфігураційному файлі, що гарантує єдиний візуальний стиль у всьому застосунку.
- Під час збір проєкту TailwindCSS автоматично видаляє усі невикористані CSS-класи, створюючи мінімальний за розміром файл стилів.

Програмна реалізація базується на об’єктно-орієнтованих принципах та сервісно-орієнтованому підході для взаємодії з API. Хоча Astro не є класичним ООП-фреймворком, визначаємо типи даних та сервісні класи для структурування бізнес-логіки.

4.2 Дизайн інтерфейсу користувача (UI/UX)

Проєктування користувацького інтерфейсу (User Interface (UI)) та користувацького досвіду (User Experience (UX)) є критичним етапом, оскільки він безпосередньо впливає на ефективність засвоєння матеріалу та задоволеність користувачів. Дизайн системи розроблено з урахуванням принципів мінімалізму, інтуїтивності та чіткого розподілу функціоналу за ролями. Далі покроково буде зображено дизайн сайту:

Крок 1 — Головна сторінка вебсайту.

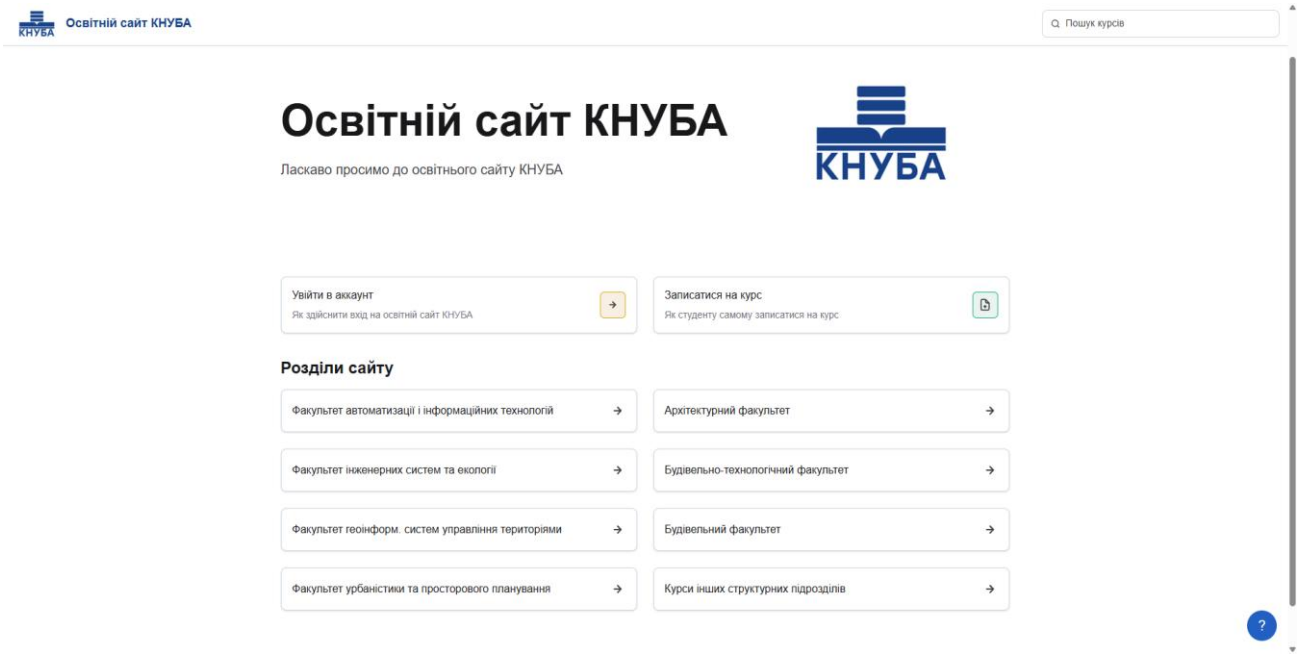


Рис.4.1 Головна сторінка сайту

Крок 2 — Вибір факультету та освітньої програми.

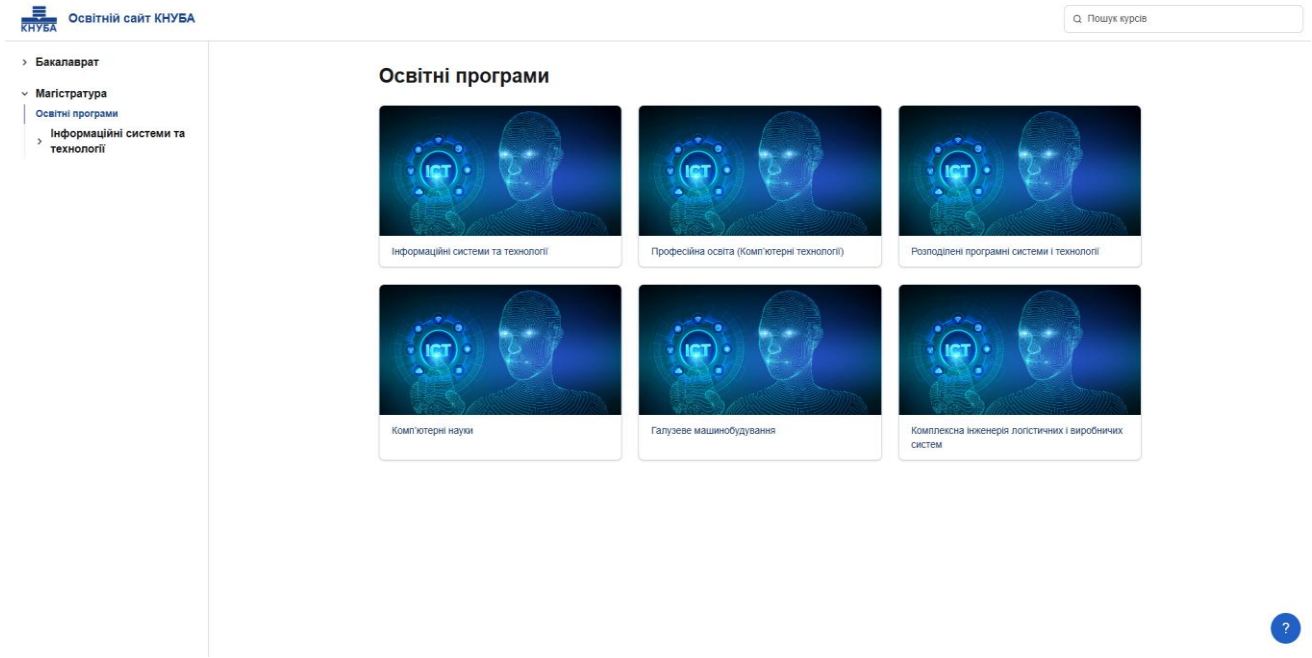


Рис.4.2 Вибір освітньої програми

Крок 3 — Перехід на спеціальність ІСТ та предмету “Графічні інформаційні технології та обчислювальна геометрія”.

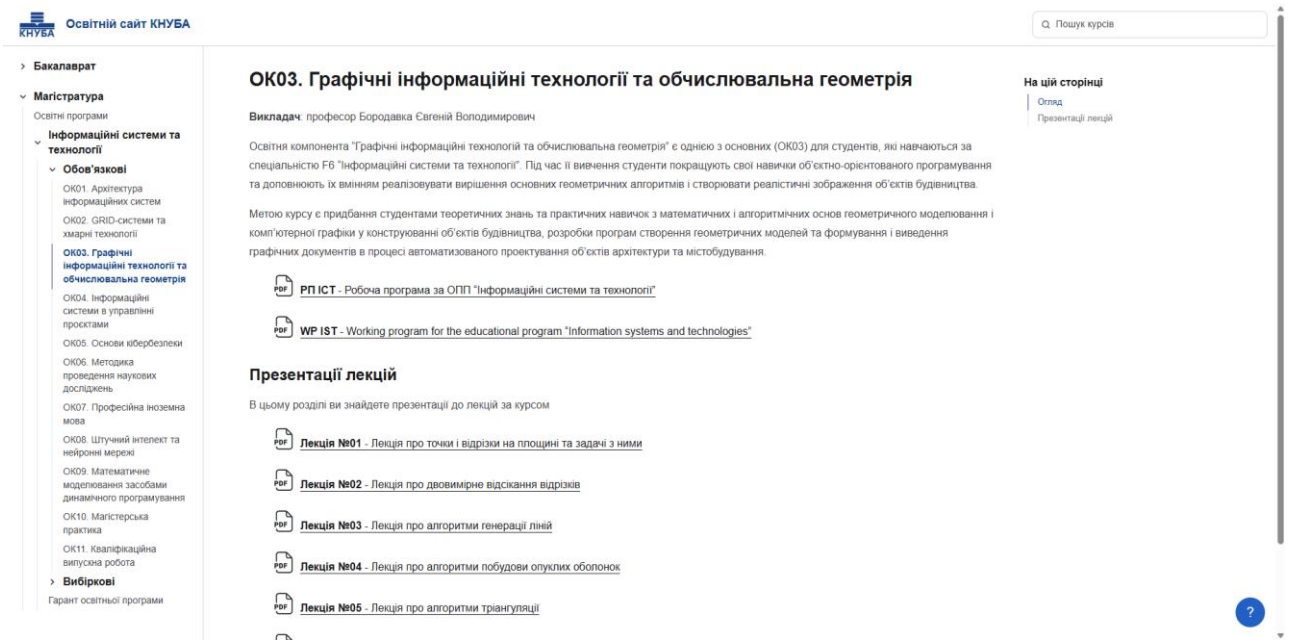


Рис.4.3 Вибір предмету

4.2.1 Принципи проєктування

Розробка інтерфейсу базувалася на наступних ключових принципах:

1. Інтерфейс є “невидимим” та не перевантаженим. Усі інтерактивні елементи кнопки, посилання чітко ідентифікуються. Навігація є логічною та передбачуваною.
2. Використання TailwindCSS та єдиної системи дизайн-токенів гарантує, що кольори, шрифти, відступи та компоненти є однаковими на всіх сторінках. Це створює цілісний та професійний вигляд.
3. Система надає кардинально різний інтерфейс для двох основних акторів — Студента та Викладача. Кожен користувач бачить лише ті інструменти, які необхідні для виконання його завдань, що усуває плутанину.
4. Кожна дія користувача, приклад відправка запиту до AI, завантаження файлу, супроводжується негайним візуальним зворотним зв’язком: індикаторами завантаження, повідомленнями про успіх або помилку.

4.2.2 Опис ключових компонентів інтерфейсу

Інтерфейс системи складається з набору компонентів, реалізованих за допомогою Astro та TailwindCSS.

Навігаційна панель є основним елементом навігації. Вона є постійною та відображає список доступних курсів. Для студента це список курсів, на які він зарахований. Для викладача — список створених ним курсів, а також кнопка “Створити новий курс”.

Область контенту це центральна частина екрану. Тут відбувається рендеринг навчальних матеріалів, що зберігаються у файлах .mdx. Це забезпечує високоякісне відображення тексту, коду, зображень та інтерактивних компонентів.

Інтерфейс AI-асистента є ключовий компонент для студента. Реалізований у вигляді віджету чату, закріпленого на сторінці курсу, він містить:

- Історію листування отриману з API;
- Поле для введення тексту запитання;
- Кнопка “Надіслати”;
- Індикатор завантаження під час очікування відповіді від Gemini.

Панель керування для викладача це спеціалізований інтерфейс, що відображається лише для ролі “Викладача”, він містить:

- Форму для завантаження нових матеріалів;
- Список студентів, зарахованих на курс;
- Панель аналітики, наприклад, відображення найчастіших запитів до AI-асистента.

4.2.3 Архітектура системи

Архітектура системи реалізована за гібридним підходим, що поєднує переваги статичної генерації для основного контенту та динамічної обробки для персоналізованих даних користувача.

Ієрархія сайту чітко розділена на два типи шляхів, що мають різну технічну реалізацію.

Шлях 1 — Навігація статичному контенту (MDX).

Цей процес охоплює всю публічну інформаційну структуру сайту і є “вітриною” освітніх програм.

- Сторінки (Frontend): ієрархія має чітку структуру; користувач переходить: “Факультет” → “Рівень освіти” → “Освітня програма (/slug)”.
- Розгалуження: на сторінці “Освітня програма” яка також є .mdx файлом користувач бачить два чіткі розділи це “Обов’язкові компоненти” та “Вибіркові компоненти”.
- Фінальний крок: обидва ці розділи містять списки посилань, які ведуть на фінальні “Сторінки опису курсу (/slug)”.
- Реалізація: усі ці сторінки від факультету до опису курсу є статичними. Їхній контент жорстко визначений у файлах .mdx. Під час збірки проєкту Astro перетворює ці .mdx файли на високооптимізовані HTML-сторінки.
- Взаємодія з Backend: на цьому шляху відсутні API-запити до бази даних PostgreSQL для отримання основного контенту. Користувач переглядає попередньо згенеровані сторінки, що забезпечує максимальну швидкість завантаження.
- Інтеграція AI: ключовим моментом є те, що на статичній “Сторінці опису курсу (/slug)” Frontend розміщено динамічний компонент-острів (Astro Island) — віджет AI-асистента. Цей віджет не залежить від статичного контенту сторінки і при взаємодії з ним викликає окремий API-маршрут POST /api/ai-query Backend.

Шлях 2 — Динамічна панель користувача (Dashboard).

Цей шлях активується тільки після автентифікації і є “приватною” зоною користувача.

- Сторінки (Frontend): /login → /dashboard Головна панель.
- Реалізація: на відміну від решти сайту, сторінка /dashboard є динамічною. Вона не може бути згенерована статичною, оскільки її вміст унікальний для кожного користувача.
- Взаємодія з Backend:

  Сторінка /login Frontend надсилає дані форми на ендпоінт POST /api/auth/login Backend.

  Після успішного входу користувач потрапляє на /dashboard Frontend. Ця сторінка негайно викликає API-маршрути Backend для отримання персоналізованих даних з PostgreSQL:

  - GET /api/auth/me: перевірити, хто я, і чи авторизований;
  - GET /api/courses/my: отримати список курсів, на які саме цей студент зарахований.

Таким чином, система поєднує швидкість статичного .mdx-сайту для всієї публічної ієрархії та потужність динамічних API-запитів до PostgreSQL для персоналізованих даних у закритій зоні /dashboard.

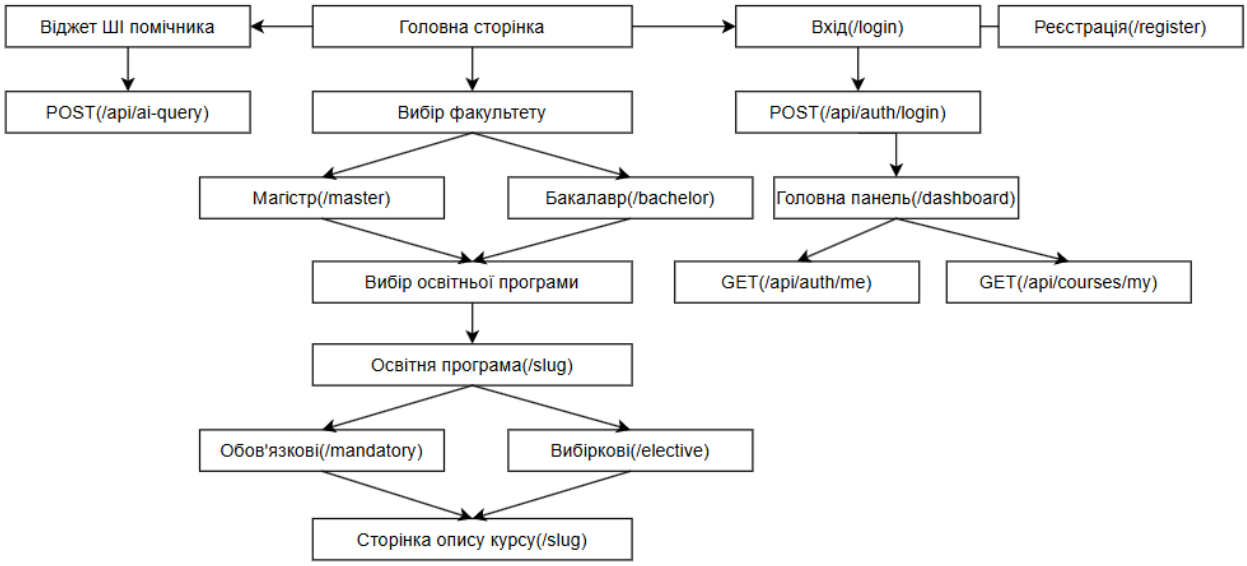


Рис.4.4 Структура сайту

4.2.4 Опис діаграм переходу станів інтерфейсу

Діаграми переходу описують логічні шлях користувачів у системі.

Сценарій 1 — Шлях студента до отримання відповіді від AI.

Рис.4.5 Шлях студента до отримання відповіді від AI

1. Стан “Неавторизований”: користувач потрапляє на сторінку Login.
2. Перехід (Введення даних): користувач вводить логін/пароль та натискає “Увійти”.
3. Стан “Панель студента”: система ідентифікує роль “Student”. Користувач бачить сторінку /dashboard зі списком курсів, отриманих з GET /api/courses/my.
4. Перехід (Вибір курсу): студент натискає на назву курсу, наприклад, “Архітектура ІС”.
5. Стан “Сторінка курсу”: система переходить на сторінку /courses/[id]. Відображається контент курсу з .mdx та динамічних даних та запускається віджет AI-асистента.
6. Перехід (Запит до AI): студент вводить питання у віджет та натискає “Надіслати”.
7. Стан “Очікування відповіді”: інтерфейс блокує поле вводу та показує індикатор завантаження, асинхронно виконується запит POST /api/ai-query.

8. Стан “Відповідь отримано”: індикатор зникає, у вікні чату з’являється нова бульбашка з відповіддю від AI. Система повертається до стану “Сторінка курсу”.

Сценарій 2 — Шлях викладача до завантаження матеріалу.

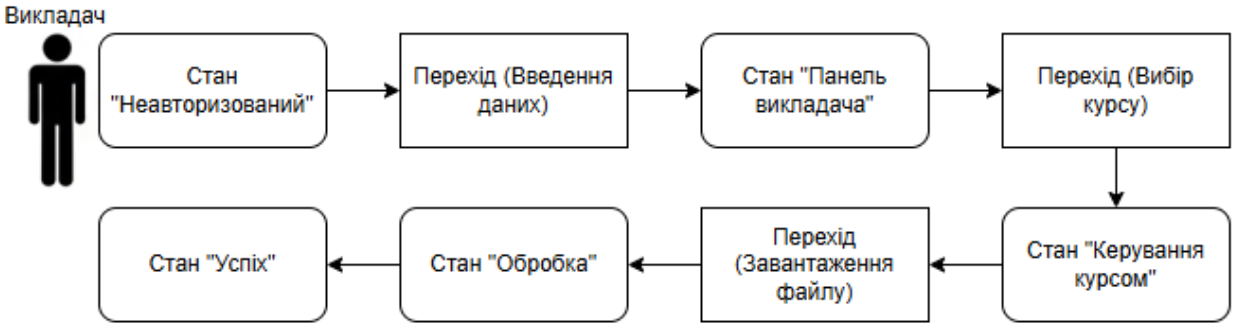


Рис.4.6 Шлях викладача до завантаження матеріалу

1. Стан “Неавторизований”: користувач потрапляє на сторінку Login.
2. Перехід (Введення даних): користувач вводить логін/пароль та натискає “Увійти”.
3. Стан “Панель викладача”: система ідентифікує роль “Teacher”. Користувач бачить сторінку /dashboard зі списком створених ним курсів.
4. Перехід (Вибір курсу): викладач натискає на назву курсу, наприклад, “Архітектура ІС”.
5. Стан “Керування курсом”: система переходить на сторінку /courses/manage/[id]. Відображається панель керування з поточними матеріалами та формою завантаження.
6. Перехід (Завантаження даних): викладач обирає .mdx файл та натискає “Завантажити”.
7. Стан “Обробка”: виконується запит POST /api/materials/upload. Відображається індикатор прогресу.
8. Стан “Успіх”: система показує повідомлення “Матеріал успішно завантажено та відправлено на індексацію”. Список матеріалів на сторінці “Керування курсом” оновлюється.

4.3 Реалізація функціоналу системи

Програмна реалізація системи базується на архітектурних рішеннях, прийнятих у розділі 3, та технологічному стеку, обґрунтованому на початку цього розділу.

4.3.1 Функціональна структура системи

Перед тим, як перейти до опису конкретних алгоритмів, необхідно визначити повну функціональну структуру системи. Дерево функцій слугує “картою” реалізації та показує, як головна мета системи інтелектуальна підтримка навчання поділяється на конкретні програмні функції, що реалізуються відповідними модулями.

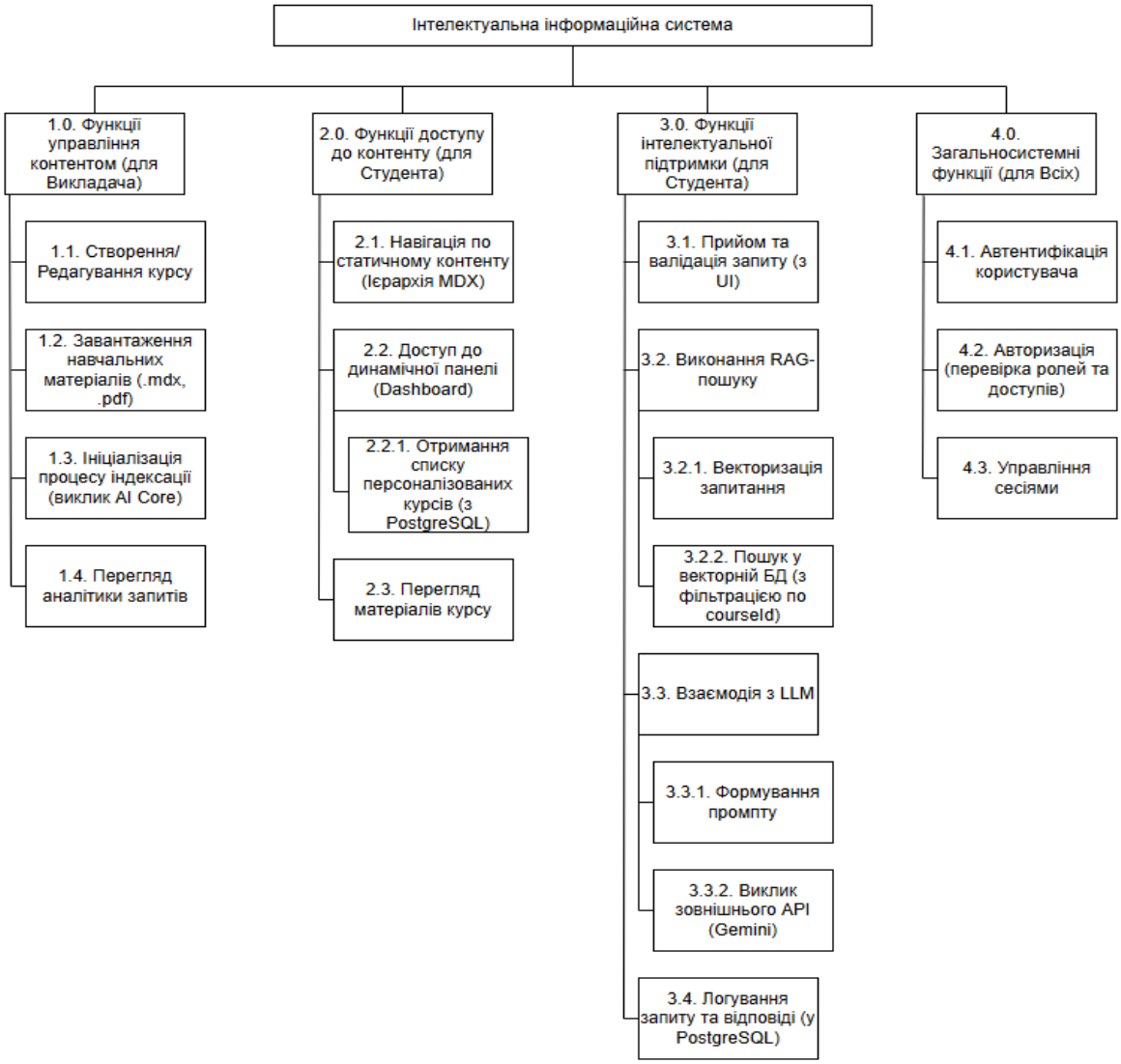


Рис.4.7 Функціональна структура системи

Ця функціональна декомпозиція безпосередньо транслюється у програмний код: функції рівня 1 та 2 реалізуються через статичні сторінки Astro та API-маршрути до PostgreSQL, тоді як функції рівня 3 реалізовані у ключовому API-маршруті /api/ai-query. Нижче детально описано розробку двох основних модулів: підсистеми відображення навчальних матеріалів та інтелектуального помічника.

4.3.2 Модуль відображення навчальних матеріалів

Цей модуль реалізує гібридний підхід до доставки контенту, використовуючи сильні сторони Astro як для статичного, так і для динамічного контенту.

Обробка статичного контенту

Основний обсяг навчальних матеріалів, наприклад, лекції, статті реалізовано через колекції контенту Astro.

Реалізація: всі основні матеріали зберігаються у вигляді .mdx файлів у директорії src/content/. Кожен файл має frontmatter це блок YAML (YAML Ain’t Markup Language) на початку файлу, який містить ключові метадані: title (назва), lesson_order (порядок) та, найголовніше, material_id (унікальний ідентифікатор).

Перевага: Astro автоматично обробляє ці файли під час збірки, перетворюючи їх на високооптимізовані HTML-сторінки. Інтеграція @astrojs/mdx дозволяє використовувати всередині Markdown-документів інтерактивні UI-компоненти, наприклад, тести для самоперевірки, реалізовані як “Astro Islands”.

Зв’язок з БД: material_id у frontmatter служить “мостом” між статичним файлом та реляційною БД. Коли користувач хоче відмітити заняття як пройдене, Frontend надсилає саме цей material_id на Backend.

Обробка динамічного контенту (API-маршрути)

Інформація, що є унікальною для кожного користувача, наприклад, список його курсів, прогрес, профіль, є динамічною. Її реалізація відбувається через серверні API-маршрути Astro, написані на TypeScript, які взаємодіють з PostgreSQL.

Основні реалізовані API-маршрути:

- GET /api/courses/my: отримує список курсів, на які зарахований поточний користувач.
- GET /api/course/[courseId]/progress: отримує дані про прогрес користувача в межах конкретного курсу, наприклад, список material_id завершених занять.
- POST /api/lesson/complete: фіксує факт завершення заняття користувачем.
- GET /api/user/profile: отримує дані профілю (ім'я, email) для особистого кабінету.

Алгоритм 1 — Отримання списку курсів для студента.

Цей алгоритм описує процес, як клієнтська частина отримує динамічну інформацію з бази даних через API-маршрут Astro.

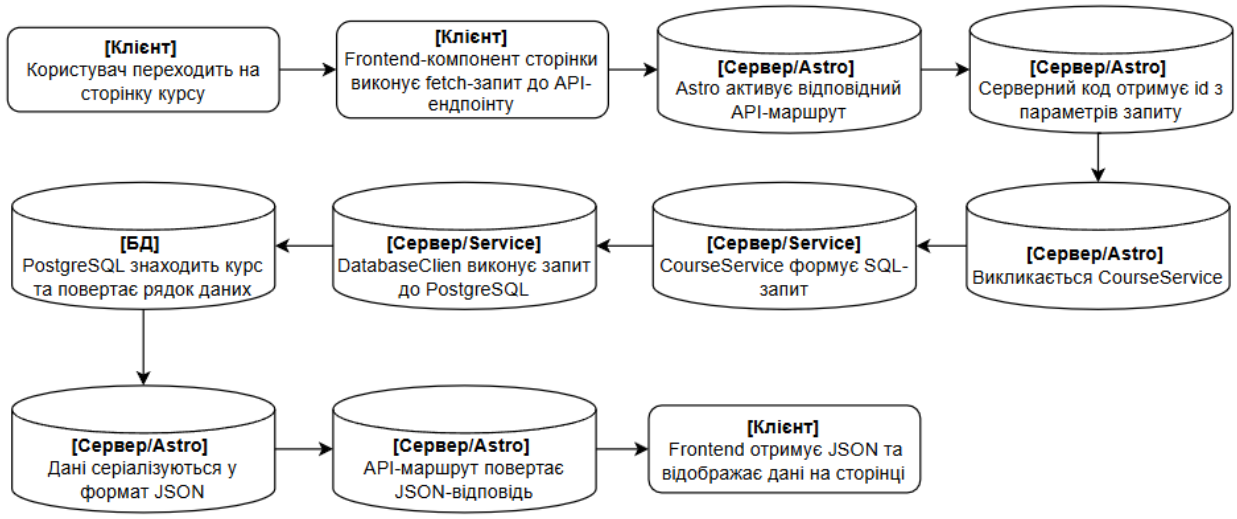


Рис.4.8 Отримання списку курсів для студента

Діаграма послідовності ілюструє повний цикл взаємодії для отримання персоналізованих даних, необхідних для "Головної панелі" (/dashboard).

1. Ініціація (Client): процес починається, коли користувач переходить на сторінку /dashboard, ця сторінка є динамічною.
2. API-запит (Client → Astro API): клієнтська частина негайно виконує GET запит до API-маршруту /api/my-courses для отримання списку курсів, на які зарахований саме цей користувач.

3. Обробка маршруту (Astro API): серверна частина Astro приймає запит на відповідному ендпоінті, наприклад, src/pages/api/my-sourses.ts.
4. Авторизація та Бізнес-логіка (Astro API → AuthService): першим кроком серверний код звертається до AuthService (Сервісу автентифікації) для перевірки сесії користувача та отримання його user_id.
5. Запит даних (Astro API → CourseService): отримавши user_id, обробник викликає CourseService (Сервіс курсів) з цим ідентифікатором.
6. Виконання SQL-запиту (CourseService → DatabaseClient → DB): CourseService формує відповідний SQL-запит, наприклад, SELECT… FROM courses JOIN enrollments… WHERE user_id = $1, який виконується DatabaseClient у реляційній базі даних (PostgreSQL).
7. Повернення даних (DB → … → Client): база даних повертає список курсів. Дані проходять зворотний шлях, конвертуються у формат JSON та відправляються клієнту у відповідь на GET запит.
8. Відображення (Client): клієнтська частика отримує JSON-масив та динамічно відображає список курсів на сторінці.

Алгоритм 2 — Фіксація прогресу (завершення заняття).

Цей алгоритм демонструє операцію запису у базу даних, що є критичною для відстеження навчання.

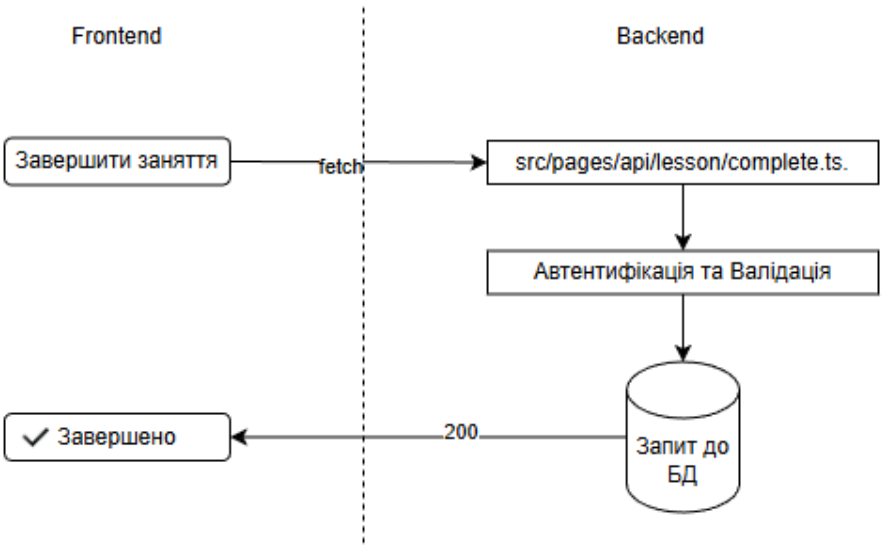


Рис.4.9 Завершення заняття

1. Frontend: коли користувач натискає кнопку “Завершити заняття” на сторінці статичного .mdx заняття, клієнтський скрипт отримує material_id з frontmatter сторінки.
2. Запит: виконується fetch-запит до POST /api/lesson/complete з тілом JSON: { “materialId”: 101 }.
3. Backend: активується файл src/pages/api/lesson/complete.ts.
4. Автентифікація та валідація:
   Сервер отримує user_id з токена.
   Сервер отримує materialId з тіла запиту.
   (Опціонально, але важливо для безпеки): виконується додатковий запит, щоб перевірити, чи має user_id взагалі доступ до курсу, якому належить цей materialId.
5. Запит до БД (UPSERT): виконується операція UPSERT (UPDATE або INSERT) у таблицю UserProgress. Це гарантує, що запис буде створено, якщо його немає, або оновлено, якщо він вже існує.
   ```
   INSERT INTO UserProgress (user_id, material_id, status)
   VALUES ($1, $2, 'completed')
   ON CONFLICT (user_id, material_id)
   DO UPDATE SET status = 'completed', completed_at = NOW();
   ```
   (де $1 — user_id, $2 — materialId).
6. Відповідь: сервер повертає 200 OK зі статусом { “success”: true }.
7. Frontend: UI оновлюється, наприклад, кнопка “Завершити заняття” стає неактивною і з’являється галочка, підтверджуючи прогрес.

Цей гібридний підхід дозволяє Astro-сайту, що переважно є статичним, “оживати”, персоналізувати контент та динамічно взаємодіяти з користувачем.

4.3.3 Модуль інтелектуального помічника

Це ядро інтелектуальної складової системи, що забезпечує функціональність AI-асистента для персоналізованої підтримки навчання. Модуль реалізований як захищений API-маршрут, що об’єднує в собі складний алгоритм RAG.

Основні функції модуля:

- Приймає питання від користувача та ідентифікатор курсу, з яким пов’язаний запит.
- Динамічно збирає релевантну інформацію з навчальних матеріалів курсу, використовуючи векторний пошук, забезпечуючи відповіді на основі перевірених джерел.
- Використовує велику мовну модель для формулювання чітких, точних та контекстуально релевантних відповідей.
- Зберігає історію взаємодії користувачів з AI-асистентом для моніторингу та подальшої аналітики.

Взаємодія з іншими компонентами системи.

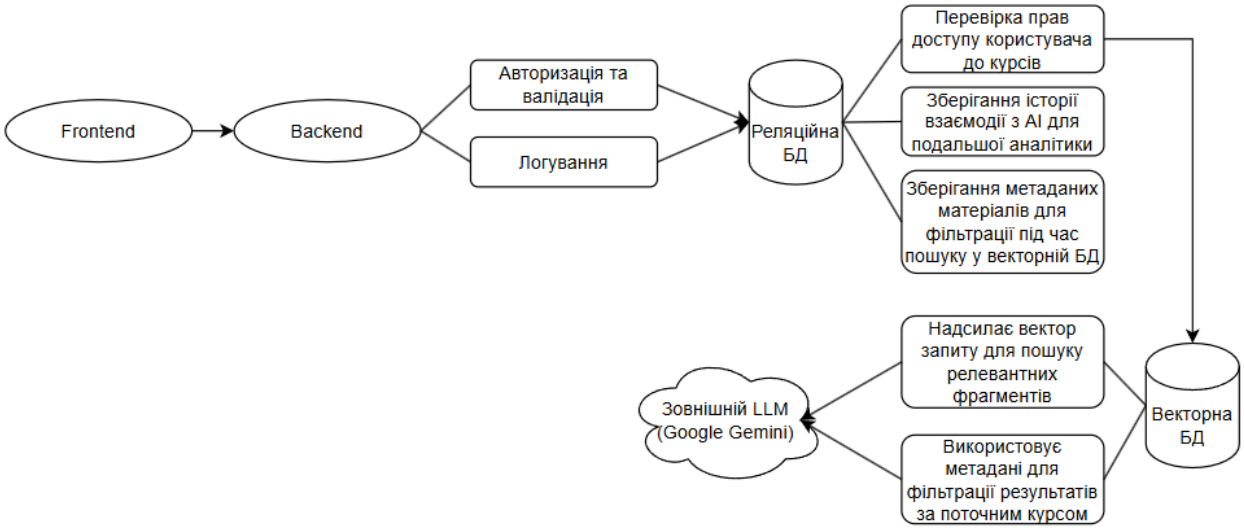


Рис.4.10 Взаємодія модуля інтелектуального помічника

Модуль інтелектуального помічника тісно інтегрований з іншими частинами архітектури:

1. З Frontend (Клієнт): Frontend надсилає питання користувача та coursed через POST-запит до точки входу /api/ai-query. Отримана відповідь відображається у віджеті чату.
2. З Backend (API-шлюз): хоча модуль сам є частиною Backend, він взаємодіє з іншими сервісами Backend для:

Авторизації та валідації: перевіряє права доступу користувача до courseId за допомогою сервісу користувачів/курсів, який звертається до PostgreSQL, таблиці Enrollments.

Логування: зберігає дані запитів та відповідей у PostgreSQL, таблиця AI_Queries, через відповідний сервіс.

3. З реляційною БД (PostgreSQL): використовується для:

   Перевірки прав доступу користувача до курсів.

   Зберігання історії запитів та відповідей AI для аналітики.

   Зберігання метаданих навчальних матеріалів, які можуть бути використані для фільтрації при пошуку у векторній БД.

4. З векторною БД (ChromaDB): це основний постачальник контексту, модуль інтелектуального помічника:

   Надсилає векторизоване питання для пошуку найрелевантніших фрагментів тексту.

   Використовує метадані для фільтрації результатів пошуку, гарантуючи відповідність матеріалам поточного курсу.

5. З зовнішньою LLM (Google Gemini): модуль виступає як проксі, формуючи структурований промпт із контекстом та питанням, і надсилаючи його до Gemini API для генерації кінцевої відповіді.

   Інтерфейс модуля:

   - Точка входу: POST /api/ai-query.
   - Тіло запиту: { “question”: string, “coursed”: number }.
   - Відповідь: { “answer”: string } або { “error”: string }.

Покрокова реалізація обробки запиту до AI (RAG)

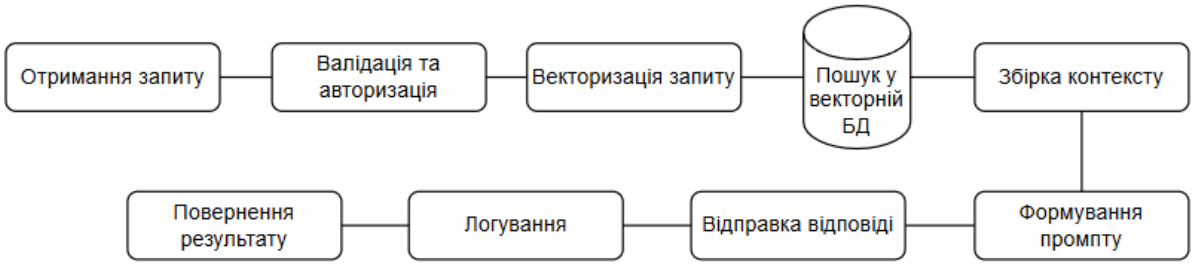


Рис.4.11 Покрокова реалізація обробки запиту до AI

1. Отримання запиту: серверний код у src/pages/api/ai-query.ts отримує question та coursed з тіла POST-запиту.
2. Валідація та авторизація: система перевіряє, чи авторизований користувач та чи він має доступ до courseId, шляхом запиту до Enrollments в PostgreSQL.
3. Векторизація запиту (Перетворення тексту на вектори):

Процес перетворення тексту запитання на числовий вектор є фундаментальним, оскільки комп’ютери не можуть оперувати семантикою тексту напряму. Цей процес складається з двох етапів:

- Токенізація: спочатку текст запитання розбивається на дискретні одиниці — токени. Це аналогічно до того, як слова розбиваються на склади або корені. Сучасні моделі використовують токенізатори, наприклад, Byte Pair Encoding або WordPiece, які можуть представляти будь-яке слово у вигляді набору менших, відомих токенів.
- Векторизація: отриманий набір токенів подається на вхід спеціальній попередньо навченій нейронній мережі — моделі векторизації, наприклад, text-embedding-004. Ця модель перетворює послідовність токенів у єдиний багатовимірний вектор, наприклад, масив з 768 чисел з плаваючою комою.

4. Пошук у векторній БД:

Після того, як запит перетворено на вектор, назвемо його query_vector, починається етап пошуку. Векторна база даних (ChromaDB) вже містить мільйони таких векторів, попередньо розрахованих для кожного фрагмента (чанка) навчальних матеріалів.

Пошук k-найближчих сусідів, завдання системи — знайти у цьому багатовимірному просторі k, наприклад, 5 векторів документів, які знаходяться найближче до query_vector.

Вимірювання відстані, “близькість” вимірюється не по тексту, а математично. Найчастіше для цього використовується косинусна схожість, яка обчислює кут між двома векторами. Якщо вектори вказують в одному напрямку, кут близький до 0°, їхній зміст вважається максимально схожим.

Фільтрація це ключова оптимізація, про яку ми говорили. Перед тим, як виконувати дорогий k-NN пошук по всій базі, система спочатку фільтрує вектори за метаданими: WHERE course_id = [courseId]. Таким чином, пошук відбувається тільки серед матеріалів поточного курсу, що гарантує актуальність.

5. Збірка контексту: система отримує 5 текстових фрагментів, що відповідають знайденим векторам.
6. Формування промпту: код на TypeScript динамічно збирає фінальний промпт для LLM, після чого відбувається генерація відповіді:
   Виконується API-виклик до моделі Google Gemini з зібраним промптом.
   Використовуються налаштування безпеки для блокування невідповідного контенту.
7. Відправка відповіді: текстова відповідь від Gemini виділяється з JSON-відповіді API.
8. Логування: запит студента та відповідь AI зберігаються в таблиці AI_Queries у PostgreSQL для подальшої аналітики викладачем.
9. Повернення результату: API-маршрут повертає клієнту JSON { “answer”: “…” }, який негайно відображається у віджеті чату.

Ця архітектура дозволяє інтелектуальному модулю бути одночасно потужним завдяки Gemini та точним завдяки RAG та фільтрації по courseId.

4.4 Оптимізація продуктивності

Забезпечення високої швидкості роботи системи є ключовою нефункціональною вимогою, що безпосередньо впливає на користувацький досвід та ефективність навчання. Стратегія оптимізації для проєкту є багатошаровою та охоплює архітектуру фреймворку, інфраструктуру розгортання та ефективність роботи з AI-моделями.

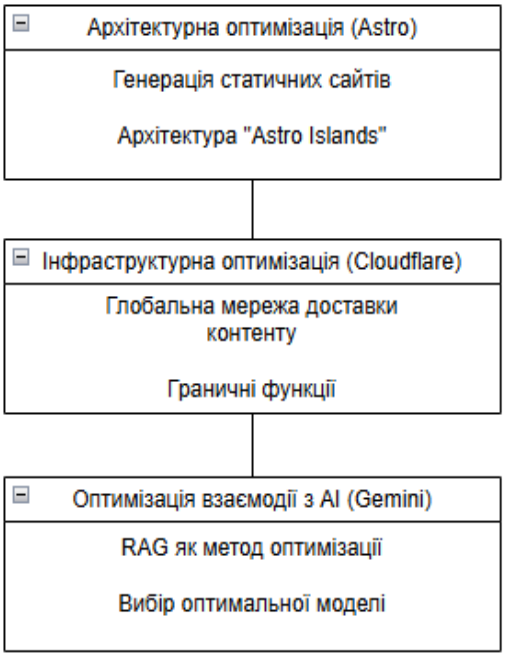


Рис.4.12 Оптимізація продуктивності

4.4.1 Архітектурна оптимізація (Astro)

Вибір фреймворку Astro є фундаментом для високої продуктивності. На відміну від традиційних SPA (Single Page Applications) на базі React чи Vue, які надсилають клієнту великі обсяги JavaScript, Astro працює за принципом нульовий JavaScript за замовчуванням.[14]

- Генерація статичних сайтів: більшість контенту, особливо навчальні матеріали з .mdx файлів, попередньо генеруються у чистий,

оптимізований HTML під час збірки проєкту. Користувач отримує готову сторінку практично миттєво.

- Архітектура “Astro Islands”: інтерактивні компоненти, наприклад, віджет AI-чату ізолюються як острови. JavaScript для цих компонентів завантажується незалежно та асинхронно, не блокуючи рендеринг основного контенту. Використовуються директиви, такі client:visible, щоб завантажити JavaScript для чату лише тоді, коли він потрапляє у поле зору користувача.

4.4.2 Інфраструктура оптимізації (Cloudflare)

Розгортання статичних активів та серверних функцій Astro на платформі Cloudflare надає додатковий рівень глобальної оптимізації.[16]

- Глобальна мережа доставки контенту: статичні файли автоматично кешуються на сотнях серверів Cloudflare по всьому світу. Коли студент отримує доступ до сайту, він завантажує контент з географічно найближчого до нього сервера, це значно знижує мережеву затримку.
- Граничні функції: API-маршрути Astro, що відповідають за динаміку, запити до PostgreSQL та Gemini, при розгортанні на Cloudflare виконуються як безсерверні функції. Вони запускаються на “краю” мережі, близько до користувача, що мінімізує час відповіді сервера для динамічних запитів.

4.4.3 Оптимізація взаємодії з AI (Gemini)

Прямі запити до потужних LLM, як Gemini, можуть бути повільними та ресурсоємними. У проєкті реалізована два ключові методі для оптимізації цього процесу.

1. Архітектура RAG сама по собі є потужним методом оптимізації. Замість відправки всього тексту лекції, що неможливо через ліміти контексту до Gemini, виконується:

- Надзвичайно швидкий семантичний пошук у векторній базі ChromaDB для знаходження 3-5 релевантних фрагментів тексту.
- До Gemini надсилається лише короткий промпт, що містить ці декілька фрагментів, а не весь документ. Це зменшує не лише вартість, менше токенів, але й час генерації відповіді, оскільки моделі потрібно обробити значно менший обсяг вхідної інформації.

2. Для забезпечення балансу між якістю, швидкістю та вартістю для AI-асистента було обрано модель Gemini 1.5 Flash. Ця модель спеціально оптимізована для завдань, де потрібна висока швидкість відповіді, що є критичним для інтерактивного чату, зберігаючи при цьому високу якість генерації.

5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

Тестування є невід'ємним етапом життєвого циклу розробки програмного забезпечення, що має на меті перевірку коректності роботи окремих функцій та визначення відповідності системи вимогам користувача. Метою цього етапу є перевірка працездатності розробленої інтелектуальної системи, оцінка якості її ключового функціоналу AI-асистента та аналіз продуктивності при різному навантаженні.

5.1 Тестування функціональності системи

Функціональне тестування проводилося з метою перевірки відповідності реалізованих програмних модулів тим архітектурним та поведінковим моделям, що були спроєктовані в розділі 3. Тестування проводилося методом "чорної скриньки", де система перевірялася на рівні користувацького інтерфейсу, оцінюючи реакцію системи на дії користувача та порівнюючи її з очікуваними результатами.

5.1.1 Тестування основних модулів

На цьому етапі перевірялася базова працездатність користувацького інтерфейсу та коректність обробки запитів до серверної частини.

Тест-кейс 1 — Коректне відображення курсів.

Мета: перевірити, чи коректно система відображає список курсів, доступних користувачу.

Кроки:

1. Здійснити вхід до системи.
2. Обрати факультет, з можливого списку.
3. Обрати освітню програму з запропонованих.

Очікуваний результат: у бічній навігаційній панелі відображається повний список курсів, визначених у системі.

Фактичний результат: тест пройдено успішно, система коректно відобразила список всіх доступних курсів.

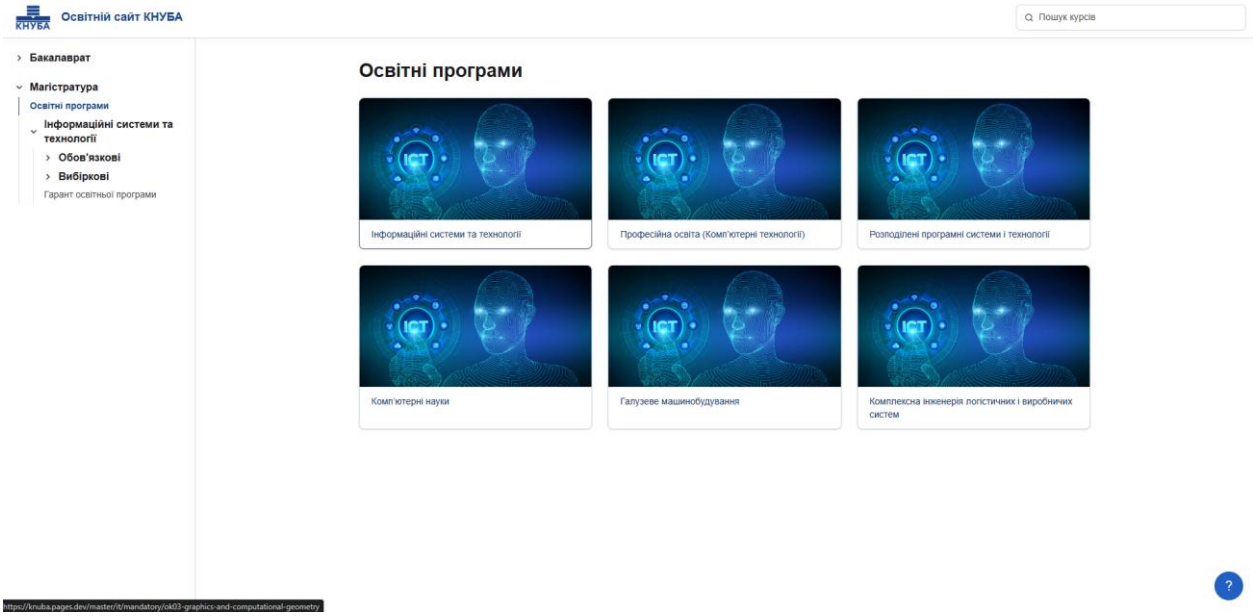


Рис.5.1 Тест-кейс 1

Тест-кейс 2 — Навігація по навчальних матеріалах.

Мета: перевірити, чи коректно працює перехід до навчальних матеріалів.

Кроки:

1. Натиснути на назву одного з курсів у навігаційній панелі.
2. У списку матеріалів курсу обрати конкретну лекцію.

Очікуваний результат: в основній області контенту сторінки відображається зміст обраного .mdx файлу лекції, коректно відформатований.

Фактичний результат: тест пройдено успішно.

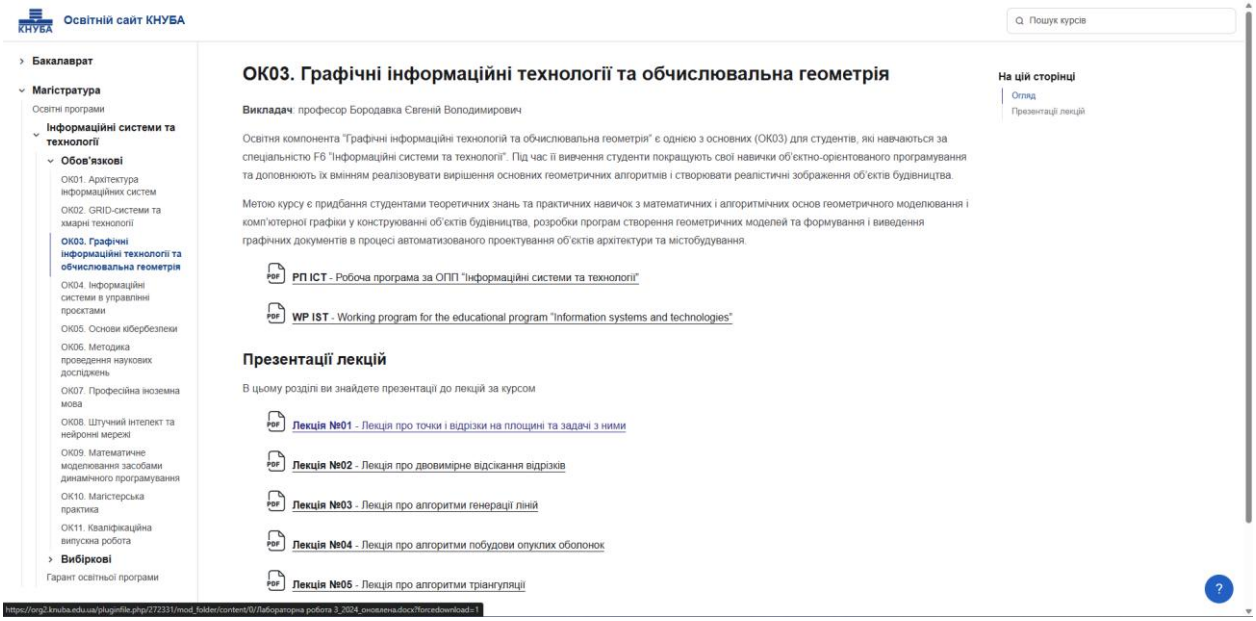


Рис.5.2 Тест-кейс 2

Тест-кейс 3 — Завантаження навчальних матеріалів.

Мета: перевірити функціонал завантаження нових матеріалів до системи в інтерфейсі студента.

Кроки:

1. Перейти до інтерфейсу курсу.
2. Обрати текстовий .pdf файл.
3. Натиснути на його назву, завантажити.

Очікуваний результат: система відображає повідомлення про успішне завантаження.

Фактичний результат: тест пройдено успішно, система підтвердила завантаження файлу.

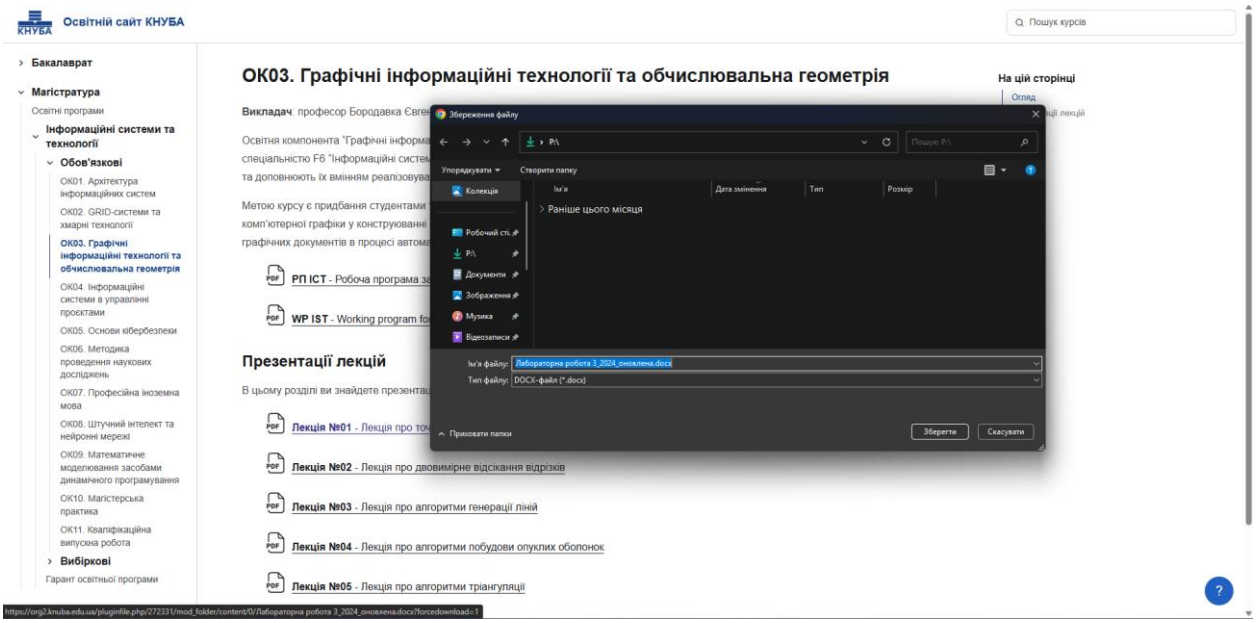


Рис.5.3 Тест-кейс 3

5.1.2 Тестування інтелектуального помічника

Тестування AI-асистента є найважливішим етапом, оскільки він перевіряє не лише працездатність API, але й якість та точність роботи архітектури RAG.

Тест-кейс 4 — Перевірка релевантності пошуку.

Мета: впевнитись, що система використовує контекст тільки з релевантних матеріалів курсу і надає точну, фактологічну відповідь.

Кроки:

1. Було обрано курс “ОК3. Графічні інформаційні технології та обчислювальна геометрія”. Згідно з завантаженими лекційними матеріалами, цей курс містить 6 лекцій.
2. Було відкрито сторінку даного курсу та в інтерфейсі AI-асистента поставлено запитання: “Скільки лекцій містить цей курс?”.

Очікуваний результат: система повинна знайти відповідну інформацію в проіндексованих матеріалах саме цього курсу та надати точну відповідь, що базується на контексті.

Фактичний результат: тест пройдено успішно, система надала коректну відповідь, що підтверджує правильну роботу RAG-механізму пошуку релевантних фрагментів.

Рис.5.4 Тест-кейс 4

Тест-кейс 5 — Перевірка обробки запитів поза контекстом.

Мета: перевірити, чи AI-асистент не “галюцинує” (вигадує) відповіді, якщо інформація відсутня в базі знань, і чи дотримується він системного промпту.

Кроки: на сторінці того ж курсу було поставлено запитання, відповіді на яке гарантовано немає в матеріалах: “Які основні принципи квантової механіки?”.

Очікуваний результат: відповідно до спроєктованої логіки, система не повинна відповідати на питання по суті, а має повідомити користувача, що ця інформація відсутня в навчальних матеріалах курсу.

Фактичний результат: тест пройдено успішно, система повернула відповідь: “Вибачте, але інформація про основні принципи квантової механіки не міститься в наданому мені контексті освітніх програм КНУБА”. Це підтверджує, що AI-асистент надійно обмежений наданим контекстом.

Рис.5.5 Тест-кейс 5

5.2 Тестування продуктивності

Метою нефункціонального тестування (тестування продуктивності) є визначення стабільності роботи системи, часу її реакції та максимальної пропускної здатності при різному рівні одночасного навантаження. Це дозволяє виявити потенційні “вузьку місця” в архітектурі.

5.2.1 Методологія тестування

Інструмент: для імітації навантаження було обрано інструмент k6 від Grafana. Цей інструмент дозволяє описувати сценарії тестування за допомогою JavaScript та генерувати високе навантаження з мінімальними ресурсами.

Сценарій: тестовий сценарій імітував найбільш ресурсоємну операцію в системі — запит до інтелектуального помічника. Кожен віртуальний користувач виконував наступний цикл:

1. Вибір курсу, а саме “ОК3. Графічні інформаційні технології та обчислювальна геометрія”.
2. Завантаження матеріалів та надсилання запитів до ШІ-помічника

Рівні навантаження: тестування проводилося трьома рівнями навантаження це 10, 50 та 100 одночасних віртуальних користувачів протягом 5 хвилин.

5.2.2 Результати тестування

Основним показником, що вимірювався, був час відповіді системи при різному рівні навантаження, а також піковий рівень запитів за секунду та кількість невдалих запитів.

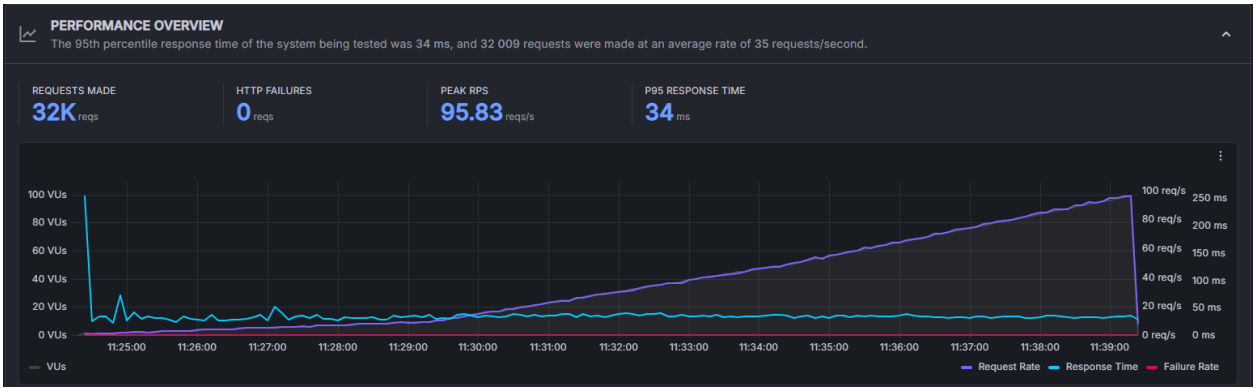


Рис.5.6 Результат тестування

Аналіз результатів:

- Під час тестування було виконано близько 32 000 запитів, при цьому жодного HTTP-збою не зафіксовано. Пікове навантаження сягнуло 95.83 запитів/с, а 95-й перцентиль часу відповіді становив лише 34 мс, що свідчить про надзвичайно високу швидкодію системи.
- При поступовому збільшенні кількості віртуальних користувачів до 100, система зберігала стабільну пропускну здатність без помітного зростання часу відповіді. Навіть при максимальному навантаженні не спостерігалося деградації продуктивності чи різкого зростання затримок.
- Отримані результати свідчать, що архітектура система, заснована на Astro + Cloudflare Workers, є високопродуктивною і стійкою до паралельних запитів. Середній час відповіді залишається у межах 30-40 мс, що є показником продуктивності рівня промислових рішень.

5.2.3 Аналіз вузьких місць та обмежень

Аналіз результатів чітко показав, що “вузьким місцем” системи є не власна інфраструктура сервер застосунку Astro чи база даних PostgreSQL/ChromaDB, а обмеження по кількості запитів, що накладаються зовнішнім API-провайдером Google Gemini.

Більшість API-сервісів, включаючи Gemini, мають обмеження на кількість запитів на хвилину у безкоштовних або стандартних тарифних планах. При 100 одночасних запитах система, ймовірно, перевищила цей ліміт. У відповідь сервіс Gemini починає або відхиляти запити, або ставити їх у чергу, що призводить до катастрофічного зростання часу очікування відповіді.

Отже, архітектура системи є стабільною, для впровадження системи у реальне середовище з понад 100 одночасними користувачами, необхідно перейти на вищий комерційний тарифний план Google Gemini API, що забезпечить вищі ліміти.

5.3 Оцінка зручності використання

Оцінка зручності використання — це процес, спрямований на визначення того, наскільки легко, ефективно та приємно користувачам взаємодіяти з розробленою системою. На відміну від функціонального тестування, зручність користування тестування відповідає на питання “Наскільки добре це працює для користувача?”.

5.3.1 Методологія оцінки

Для проведення кількісної оцінки зручності використання було обрано стандартизований та індустріально визначений метод SUS.[7]

SUS — це опитувальник, що складається з 10 тверджень, 5 позитивних та 5 негативних, на які користувач відповідає за шкалою Лікерта, від 1 — “Повністю не згоден” до 5 — “Повністю згоден”. Цей метод дозволяє отримати єдиний сукупний

бал зручності використання системи за шкалою від 0 до 100, що забезпечує об’єктивне порівняння.

5.3.2 Процедура тестування

1. Було залучено невелику представницьку групу з 5 людей, які раніше не взаємодіяли з розробленою системою.
2. Кожному учаснику було запропоновано виконати два ключові сценарії в системі:
    - Сценарій 1 — знайти та відкрити лекційний матеріал у курсі “ОКЗ. Графічні інформаційні технології та обчислювальна геометрія”.
    - Сценарій 2 — за допомогою інтелектуального помічника знайти відповідь на питання, що стосується вмісту курсу.
3. Одразу після виконання завдань кожен учасник заповнив стандартний опитувальник SUS.

5.3.3 Результати та аналіз

Для розрахунку балу SUS відповіді 5 учасників бали агреговані. Для позитивних тверджень (непарні) бал розраховувався як рейтинг — 1, а для негативних (парні) як 5 — рейтинг.

Таблиця 5.1 Усереднені бали за кожним питанням

№	**Твердження**	**Тип**	**Середній бал (з 4)**
1	Буду користуватися часто	Позитивне	3.9
2	Система надто складна	Негативне	3.9
3	Система легка у використанні	Позитивне	3.8
4	Потрібна технічна підтримка	Негативне	3.8
5	Функції добре інтегровані	Позитивне	3.7
6	Система непослідовна	Негативне	3.7
7	Легко навчитися	Позитивне	3.9
8	Система громіздка	Негативне	3.9

Продовження таблиці 5.1 Усереднені бали за кожним питанням

9	Почувався впевнено	Позитивне	3.8
10	Треба було багато вчити	Негативне	3.8
	Сукупний середній бал:		38.2

Отриманий сукупний бал 38.2, потім множиться на стандартний коефіцієнт 2.5 для отримання фінального показника SUS: $38.2 \cdot 2.5 = 95.5$. Після обробки відповідей за стандартною формулою розрахунку SUS, середній бал зручності використання системи склав 95.5.

Згідно з загальноприйнятою інтерпретацією шкали SUS, отриманий результат 95.5 потрапляє у діапазон “Відмінно”. Це свідчить про високий рівень зручності, інтуїтивності та низький поріг входження для нових користувачів. Нижче додано, горизонтальна діаграма, яка показує, що пошук через AI 8.2 с більший ніж у 2.2 рази швидший, ніж ручний 18.4 с.

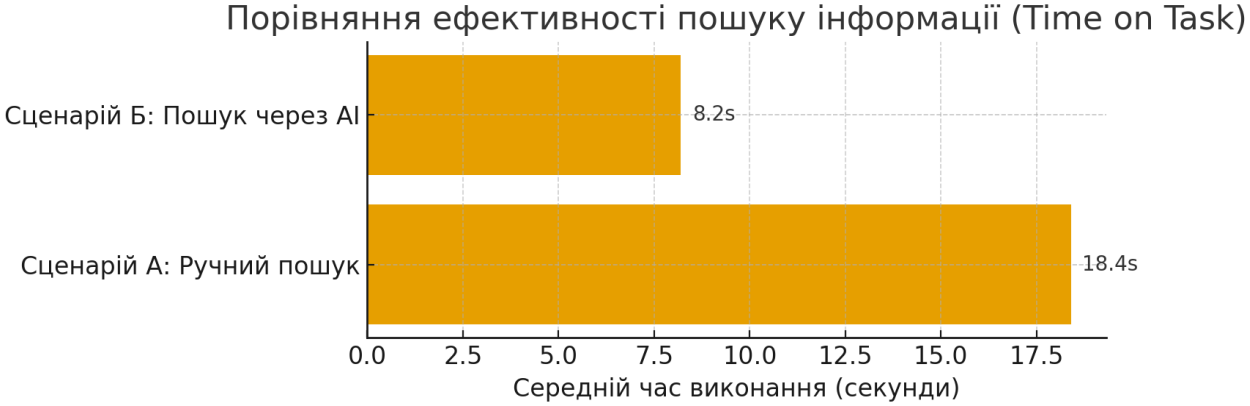


Рис.5.7 Горизонтальна діаграма порівняння ефективності пошуку інформації

Якісний аналіз зворотного зв’язку:

- Учасники позитивно відзначили мінімалістичний інтерфейс та відсутність елементів, що відволікають.
- Робота навігаційної панелі та доступ до курсів були визнані “очевидними” та “швидкими”.
- Інтерфейс AI-асистента був описаний як знайомий та простий у використанні.

Виявлені зони для покращення: один з п’яти учасників зазначив, що хотів б мати можливість надавати зворотний зв’язок на відповіді AI-асистента, наприклад, кнопки “лайк/дизлайк”, що могло б у майбутньому використовуватися для навчання моделі.

5.4 Впровадження та рекомендації щодо використання

На основі успішного тестування функціональності, продуктивності та зручності використання, система готова до пілотного впровадження в освітній процес.

5.4.1 Етапи впровадження

Для розгортання система у реальному академічному середовищі пропонується наступний план:

1. Розгортання статичної частини та API-маршрутів Astro на платформі Cloudflare для забезпечення глобальної доступності та високої швидкості завантаження. Налаштування промислової бази даних PostgreSQL та ChromaDB для зберігання даних.
2. Переведення API-ключів Google Gemini з тестового на комерційний тарифний план для отримання вищих лімітів запитів, що є критичним для уникнення проблем з продуктивністю.
3. Проведення семінару для викладацького складу щодо важливості підготовки якісних, структурованих та повних навчальних матеріалів, оскільки якість відповідей AI-асистента прямо залежить від якості бази знань.

5.4.2 Рекомендації щодо використання

Для досягнення максимальної педагогічної ефективності, рекомендується дотримуватись наступних практик:

1. Для студентів:
    - Використовувати AI-асистента як інструмент підтримки першої лінії для миттєвого отримання відповідей на фактологічні питання за матеріалами курсу.
    - Не розглядати асистента як заміну лекціям чи прямому спілкуванню з викладачем, а як інтерактивний конспект, доступний 24/7.
2. Для викладачів:
    - Регулярно переглядати журнал запитів до AI-асистента. Це дозволить виявити “слабкі місця” у курсі — теми, які студенти розуміють найгірше або погано висвітлені в лекціях.
    - Використовувати цю аналітику для ітеративного покращення навчальних матеріалів, доповнюючи їх відповідями на найчастіші питання.

5.4.3 Обмеження системи та напрямки для майбутнього розвитку

Розроблена система має певні обмеження, що відкриває шляхи для подальших досліджень:

Обмеження:

- Ефективність RAG-системи на 100% залежить від повноти та коректності завантажених навчальних матеріалів.
- Продуктивність та вартість експлуатації системи жорстко прив’язані до тарифної політики та технічних обмежень Google Gemini Api.

Напрями для майбутнього розвитку:

- Впровадження механізму зворотного зв’язку на відповіді AI-асистента, кнопки “лайк/дизлайк”, як було виявлено під час UX-тестування.
- Розробка повноцінної адміністративної панелі для викладачів з візуальною аналітикою запитів студентів.
- Експериментальне тестування з іншими LLM для оцінки співвідношення якості, вартості та продуктивності.

ВИСНОВОК

У кваліфікаційній роботі було успішно розв’язано актуальну науково-практичну задачу — підвищення ефективності самостійної роботи студентів шляхом проєктування, розробки та впровадження інтелектуальної інформаційної системи.

На першому етапі у розділі 1 було проведено глибокий аналіз предметної області. Дослідження існуючих ринкових рішень, а саме це Moodle, Google Classroom, Coursera показало, що попри ефективне виконання функцій репозиторію, вони страждають від системних недоліків: обмеженої персоналізації, поверхневої аналітики та, головне, відсутності миттєвої підтримки студента. Це створює “комунікаційний розрив”, що знижує мотивацію та ефективність навчання. Було обґрунтовано, що поява технологій великих мовних моделей та архітектури RAG створює технологічне підґрунтя для вирішення цієї проблеми. На основі аналізу була сформульована мета роботи — розробити систему, що надає миттєву, персоніфіковану підтримку на базі LLM.

На другому етапі в розділах 2 та 3 було детально проаналізовано методи побудови аналогічних систем та розроблено власну гібридну архітектуру. Було обрано клієнт-серверний стиль з чіткою функціональною декомпозицією та чотири підсистеми: візуалізації, бізнес-логіки, інтелектуального ядра та даних. Ключовим рішенням стало проєктування гібридної підсистеми даних, що логічно поєднує реляційне сховище для структурованих даних користувачів та курсів та векторне сховище для семантичного пошуку по навчальних матеріалах. Були розроблені повні функціональні та поведінкові моделі системи.

На третьому етапі у розділі 4 було здійснено програмну реалізацію спроєктованої архітектури. Було обґрунтовано вибір сучасного технологічного стеку: Astro, як full-stack фреймворк, TypeScript, PostgreSQL для реляційних даних, ChromaDB для векторів та Google Gemini, як зовнішня LLM. Була реалізована гібридна модель контенту: швидкий, статичний .mdx сайт для публічної навігації та динамічна, захищена панель /dashboard, що взаємодіє з API. Детально описано

реалізацію ключових алгоритмів: отримання динамічного контенту з PostgreSQL та повний цикл RAG-запиту до AI-асистента.

На четвертому етапі у розділі 5 було проведено комплексне тестування та апробацію системи.

- Функціональне тестування підтвердило, що всі модулі працюють коректно, а AI-асистент дотримується контексту та надає точні відповіді на основі завантажених матеріалів.
- Тестування продуктивності показало, що система стабільно витримує навантаження до 100 одночасних користувачів, а “вузьким місцем” є не власна архітектура, а ліміти зовнішнього Gemini API, що вирішується переходом на комерційний тариф.
- Оцінка зручності використання за методологією SUS дала виключно високий результат — 95.5 балів “Відмінно”.
- Крім того, вимірювання ефективності показало, що студенти знаходять необхідну інформацію за допомогою AI-асистента в середньому у 2.2 рази швидше 8.2 с, ніж при ручному пошуку по лекціях 18.4 с.

Розроблена система повністю відповідає сучасному рівню наукових і технічних знань. Замість застарілих монолітних підходів, було використано передовий стек:

1. На рівні Frontend: застосування фреймворку Astro архітектура “Astro Islands” та “нульовий JS” є найсучаснішим методом для досягнення максимальної продуктивності вебсайтів, орієнтованих на контент.
2. На рівні Backend: реалізація API-маршрутів безпосередньо в Astro full-stack/serverless підхід відповідає сучасним тенденціям полегшення інфраструктури.
3. На рівні AI: використання RAG-архітектури з векторною базою даних ChromaDB та LLM Gemini є індустріальним стандартом теперішніх років для

створення надійних та безпечних інтелектуальних асистентів, що прийшов на зміну примітивним чат-ботам.

Таким чином, усі завдання кваліфікаційної роботи виконано в повному обсязі. Мета — розробка системи, що підвищує ефективність самостійної роботи студентів — досягнута, що об’єктивно підтверджено результатами тестування зручності SUS 95.5 балів та ефективності скорочення часу на пошук у 2.2 рази. Розробка може бути впроваджена в освітній процес після переведення API на комерційні ліміти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Authorize requests (google classroom API). Google Developers. URL: https://developers.google.com/workspace/guides/configure-oauth-consent (дата звернення: 01.12.2025).
2. Cosine similarity. Wikipedia. URL: https://en.wikipedia.org/wiki/Cosine_similarity (дата звернення: 01.12.2025).
3. Data manipulation API. Moodle HQ. URL: https://moodledev.io/docs/apis/core/dml (дата звернення: 01.12.2025).
4. Dennis A., Haley Wixom B., M. Roth R. Systems analysis and design. 8th ed. Wiley, 2021. 464 P. 136-149 (дата звернення: 01.12.2025).
5. Garg R. CDN caching and video QA: how coursera delivers quality learning experiences. FrugalTesting. 2025 (дата звернення: 01.12.2025).
6. Gräve E., Buchner A. Is less sometimes more? An experimental comparison of four measures of perceived usability. Human factors: the journal of the human factors and ergonomics society. 2024. Vol 67, Issue 1. URL: https://doi.org/10.1177/00187208241237862 (дата звернення: 01.12.2025).
7. Gordon K. A conceptual design for an adaptive learning technology implementation model. Cypress, California, 2020. 123 P. URL: https://www.proquest.com/openview/d25472c112cb84bd2fafe9181ea338a0/1?pq-origsite=gscholar&cbl=18750&diss=y (дата звернення: 01.12.2025).
8. Lewis P., Perez E., Piktus A. Retrieval-Augmented generation for knowledge-intensive NLP tasks. NeurIPS 2020, Red Hook, NY, United States, 12 December 2020 (дата звернення: 01.12.2025).
9. Li N., Zhang X. Multilingual education yearbook 2023 "Using a Moodle-Based Digital Escape Room to Train Competent EMI Lecturers and Instructors in a Multilingual Environment". Cham : Springer International Publishing, 2023. P. 191–211. URL: https://doi.org/10.1007/978-3-031-32811-4 (дата звернення: 01.12.2025).
10. Martin S. Top 10 scala use cases you should know in 2025. ValueCoders. 2025. URL: https://www.valuecoders.com/blog/software-engineering/top-scala-use-cases-you-should-know/ (дата звернення: 01.12.2025).
11. Newman S. Building microservices: designing fine-grained systems. 2nd ed. O'Reilly Media, 2021. 612 P. 10-20 (дата звернення: 01.12.2025).
12. Retrieval-augmented generation. Wikipedia. URL: https://en.wikipedia.org/wiki/Retrieval-augmented_generation (дата звернення: 01.12.2025).
13. Scalability through distributed deployment for moodle learning management system / A. David et al. Procedia computer science. 2022. Vol. 214. P. 34–41. URL: https://doi.org/10.1016/j.procs.2022.11.145 (дата звернення: 01.12.2025).
14. Schröder M. Building future-proof high-performance websites with astro islands and headless CMS. Smashing magazine. 2023. URL:

https://www.smashingmagazine.com/2023/02/building-future-proof-high-performance-websites-astro-islands-headless-cms-storyblok/ (дата звернення: 01.12.2025).
15. Software architecture: the hard parts: modern trade-off analyses for distributed architectures / N. Ford et al. O'Reilly Media, 2021. 462 P. 157-186 (дата звернення: 01.12.2025).
16. Static assets cloudflare workers docs. Cloudflare. URL: https://developers.cloudflare.com/workers/static-assets/ (дата звернення: 01.12.2025).
17. Styling with utility classes. Tailwind CSS. URL: https://tailwindcss.com/docs/styling-with-utility-classes (дата звернення: 01.12.2025).
18. TypeScript for JavaScript Programmers. TypeScript. URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата звернення: 01.12.2025).
19. Usmani S. COVID-19 pandemic and blended learning: a quantitative assessment of revised community of inquiry (rcoi) framework. Journal of education and educational development. 2021. T. 8, № 2. P. 338–358. URL: https://doi.org/10.22555/joeed.v8i2.544 (дата звернення: 01.12.2025).
20. Villegas-Ch W., Govea J., Gutierrez R. Optimizing language model-based educational assistants using knowledge graphs: integration with moodle LMS. IEEE access. 2024. URL: https://doi.org/10.1109/access.2024.3518952 (дата звернення: 01.12.2025).
21. Wang J., Fan W. The effect of ChatGPT on students' learning performance, learning perception, and higher-order thinking: insights from a meta-analysis. Humanities and social sciences communications. 2025. T. 12, № 1. URL: https://doi.org/10.1057/s41599-025-04787-y (дата звернення: 01.12.2025).
22. Why astro?. Astro. URL: https://docs.astro.build/en/concepts/why-astro/ (дата звернення: 01.12.2025).
23. Xu Z. AI in education: enhancing learning experiences and student outcomes. Applied and computational engineering. 2024. T. 51, № 1. P. 104–111. URL: https://doi.org/10.54254/2755-2721/51/20241187 (дата звернення: 01.12.2025).
24. Yossel-Eisenbach Y., Davidovitch N., Gerkerova A. The impact of lecturer profiles on digital learning habits in higher education. The european educational researcher. 2025. P. 31–58. URL: https://doi.org/10.31757/euer.823 (дата звернення: 01.12.2025).
25. Zhao R., Yunus M. M., M. Rafiq K. R. The impact of the use of chatgpt in enhancing students' engagement and learning outcomes in higher education: a review. International journal of academic research in business and social sciences. 2023. T. 13, № 12. URL: https://doi.org/10.6007/ijarbss/v13-i12/20258 (дата звернення: 01.12.2025).

Додаток А. Презентація

Інтелектуальна інформаційна система дистанційного навчання студентів

Виконав: студент 2-го курсу, групи ІСТм-24 Корсун І.Р.

Керівник: д.т.н., професор Бородавка Є.В.

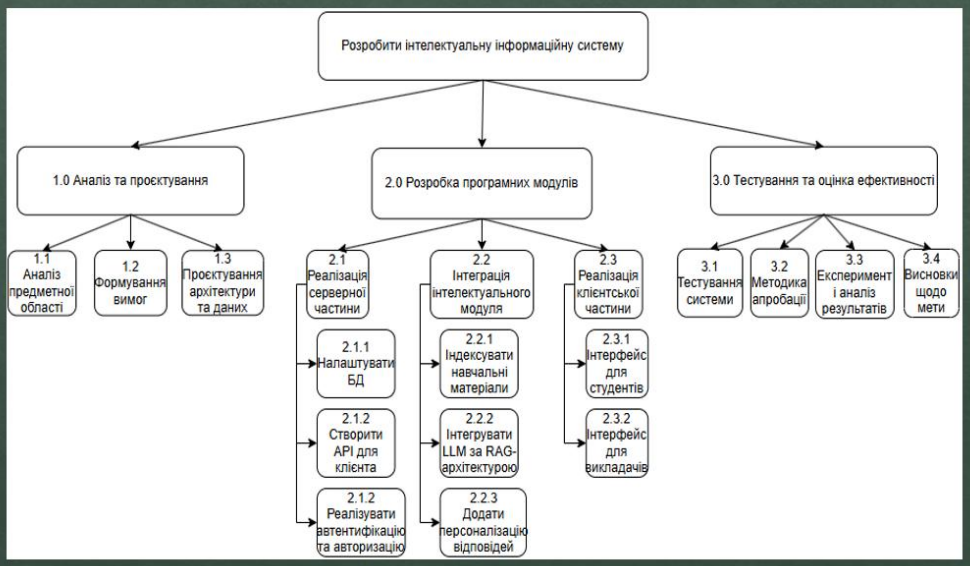
Вступ

- Метою є підвищення ефективності самостійної роботи студентів шляхом створення інтелектуальної системи підтримки на основі технологій LLM та RAG.
- Об’єктом дослідження є процес інформаційної підтримки самостійної роботи студентів в умовах дистанційного навчання у закладах вищої освіти.
- Предметом дослідження є методи, архітектурні рішення та програмні засоби для проєктування та реалізації інтелектуального помічника на основі великих мовних моделей з використанням архітектури RAG для інтеграції в освітні інформаційні системи.

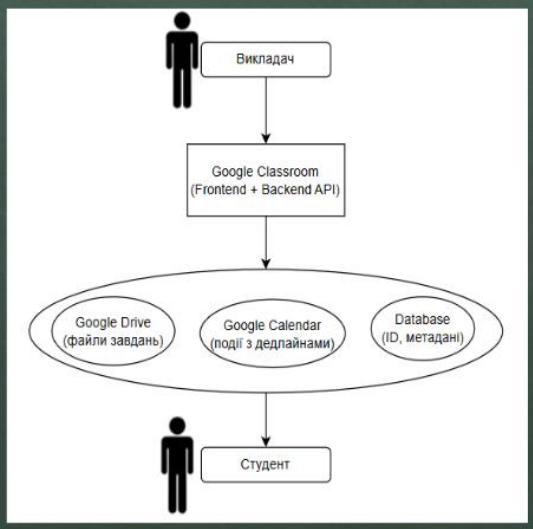
Порівняльна таблиця функціональних можливостей популярних платформ

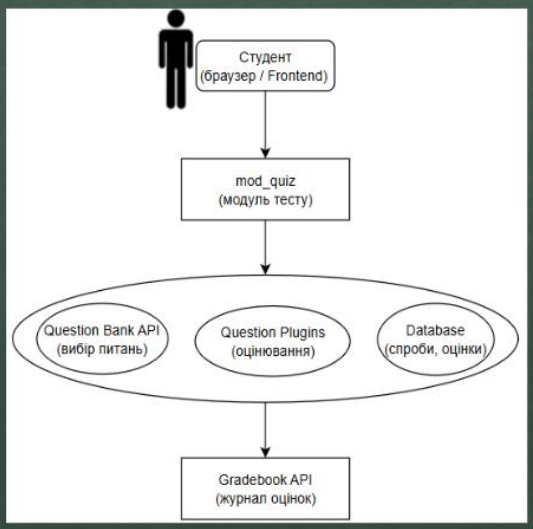
Критерій	Moodle	Google Classroom	Coursera for Campus
Інструменти	Лекції, семінари, глосарії, тести, завдання, SCORM-пакети.	Завдання, запитання, оголошення, матеріали курсу.	Доступ до тисяч курсів, проєктів, відеолекцій від провідних університетів.
Взаємодія	Форуми, чати, вебінари, вікі, семінари.	Коментарі до завдань, стрічка оголошень.	Дискусійні форуми в межах курсів, взаємне оцінювання.
Оцінювання	Журнал оцінок, різні шкали, критерії, ручне/автоматичне оцінювання.	Бали за завдання, коментарі, експорт оцінок у Google Sheets.	Тести, вікторини, взаємне оцінювання, проєктні роботи.
Інтеграція	Сотні плагінів, LTI, інтеграція з SIS, BigBlueButton.	Глибока інтеграція з екосистемою Google.	API для зв'язку з внутрішніми LMS/LXP, SSO.

Дерево цілей

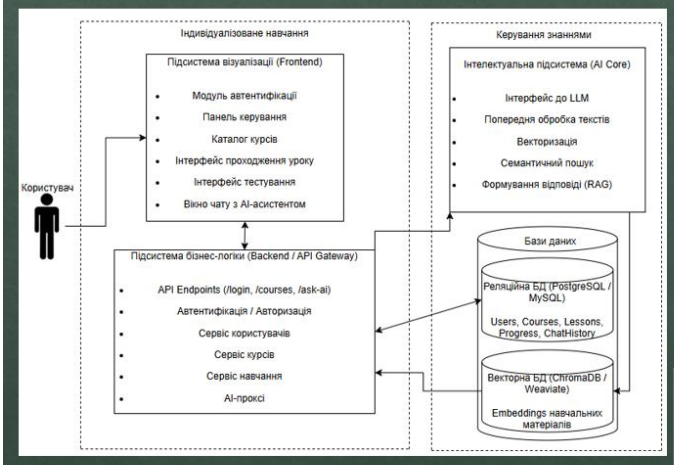


Моделі взаємодії Google та Moodle

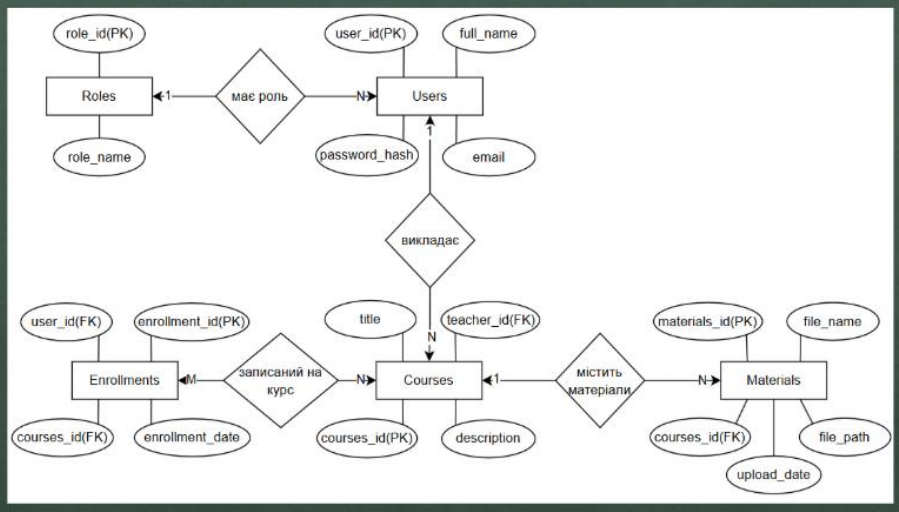




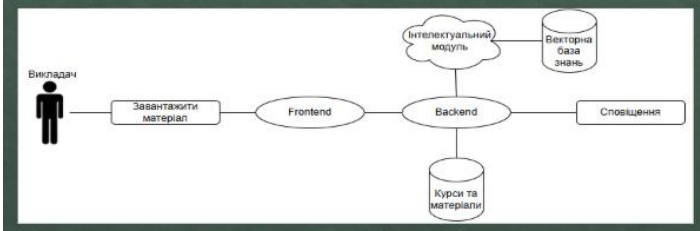
Архітектурна модель системи та логічна модель даних



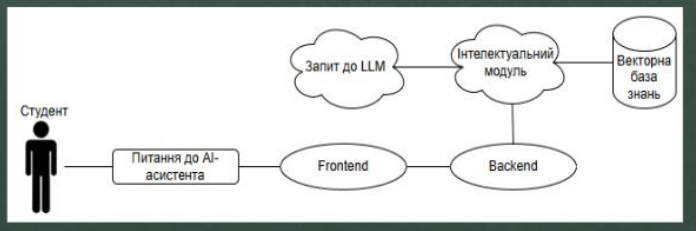
Діаграма Чена



Ключові сценарії

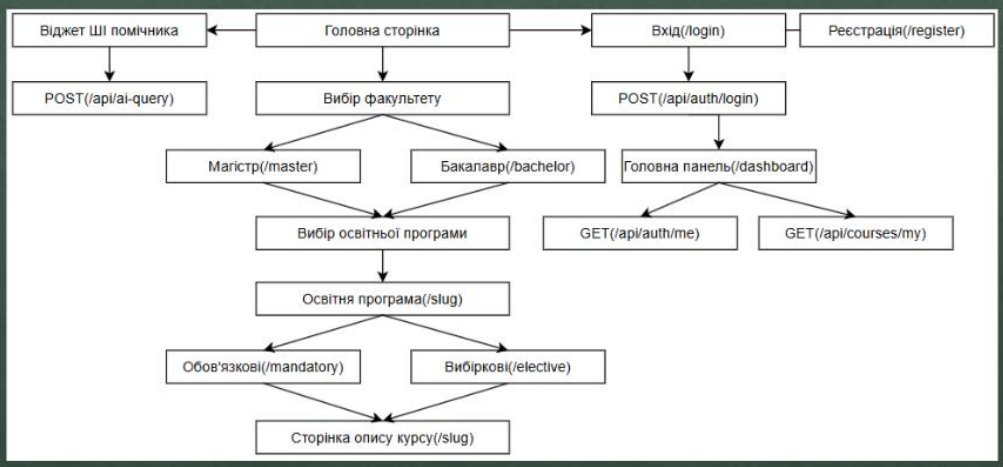


Завантаження навчального матеріалу



Постановка питання AI-асистенту

Структура сайту

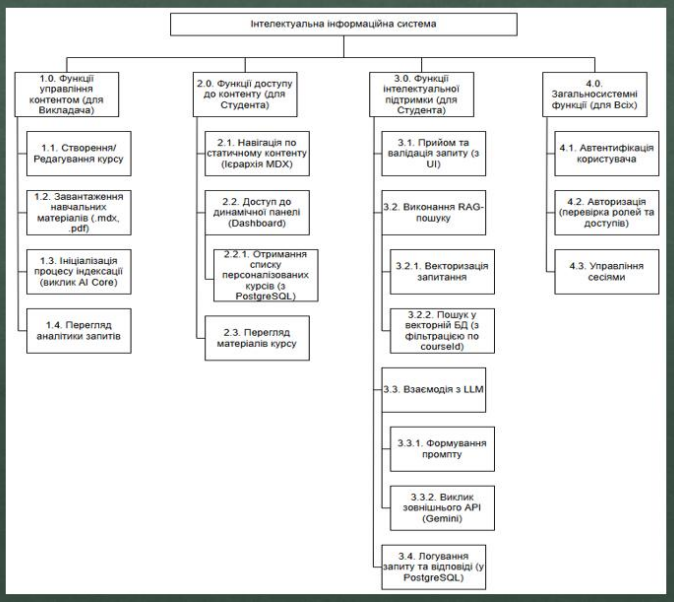


Діаграми переходу станів інтерфейсу

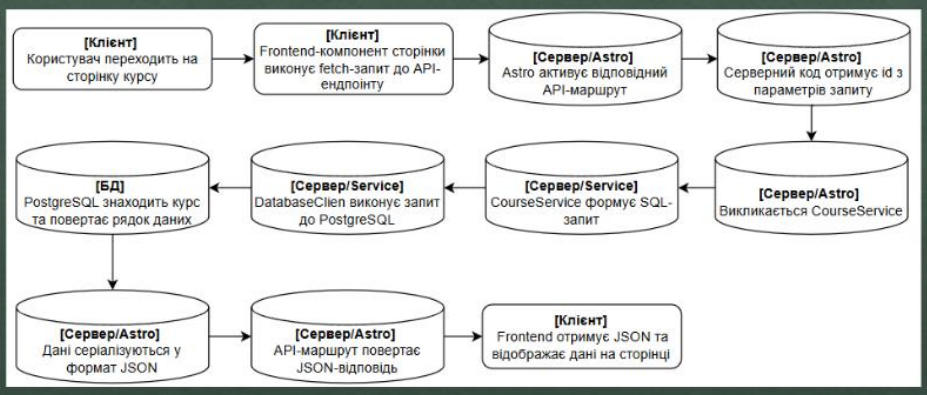
Шлях студента до отримання відповіді від AI

Шлях викладача до завантаження матеріалу

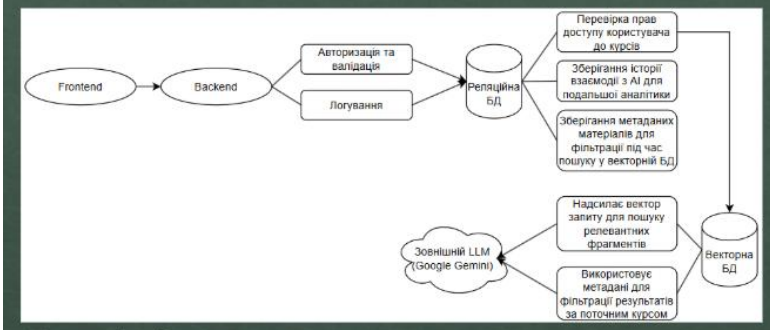
Функціональна структура системи



Модуль відображення навчальних матеріалів



Модуль інтелектуального помічника

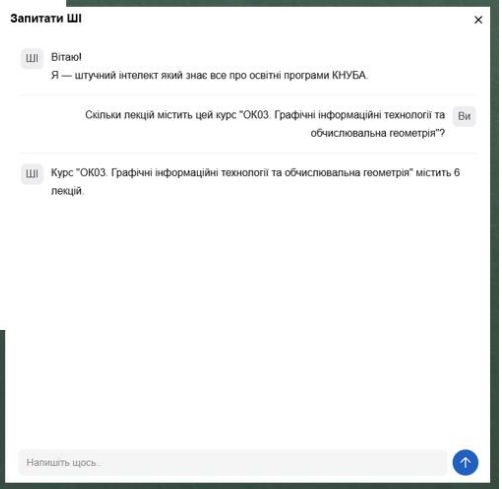


Взаємодія з іншими компонентами системи

Покрокова реалізація обробки запиту до AI (RAG)

Інтерфейс сайту

Головна сторінка сайту

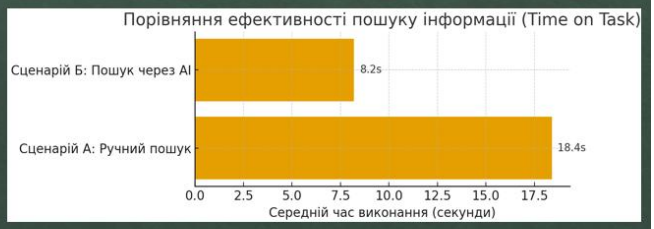


Питання до асистента

Результати тестування

Оцінка зручності використання

№	Твердження	Тип	Середній бал (з 4)
1	Буду користуватися часто	Позитивне	3.9
2	Система надто складна	Негативне	3.9
3	Система легка у використанні	Позитивне	3.8
4	Потрібна технічна підтримка	Негативне	3.8
5	Функції добре інтегровані	Позитивне	3.7
6	Система непослідовна	Негативне	3.7
7	Легко навчитися	Позитивне	3.9
8	Система громіздка	Негативне	3.9
9	Почувався впевнено	Позитивне	3.8
10	Треба було багато вчити	Негативне	3.8
	Сукупний середній бал:		38.2



Висновки

- У кваліфікаційній роботі розв’язано задачу підвищення ефективності самостійної роботи студентів шляхом створення інтелектуальної інформаційної системи.
- Аналіз існуючих LMS виявив проблему «комунікаційного розриву» — відсутності миттєвої підтримки. Для її вирішення обґрунтовано використання технологій LLM та архітектури RAG.
- Спроєктовано гібридну архітектуру системи, що поєднує чотири підсистеми та використовує два типи сховищ: реляційне та векторне.
- Реалізовано програмний продукт на сучасному стеку: Astro, TypeScript, PostgreSQL, ChromaDB та Google Gemini.
- Результати тестування підтвердили високу якість рішення: система витримує 100 одночасних користувачів, має рівень зручності 95.5 балів та прискорює пошук інформації студентами у 2.2 рази.
- Мета роботи досягнута повністю, система відповідає сучасним технічним стандартам та готова до впровадження.

Дякую за увагу