Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

«Допущено до захисту»
Завідувач кафедри
комп’ютерних наук,
доктор технічних наук,
професор
_____А.П. Бондарчук
(підпис)
« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «Магістр»
Спеціальність 122 Комп’ютерні науки
Освітня програма 122.00.02 Інформаційно-аналітичні системи

Тема роботи: «Педагогічні AI агенти: проектування та імплементація»

Виконав
студент групи ІАСм-1-24-1.4д
Остапенко Д.А.
_____
(підпис)

Науковий керівник
доктор педагогічних наук,
професор
Смирнова-Трибульська Є.М.

_____
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп'ютерних наук, кандидат технічних наук, доцент
_____ Машкіна І.В.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

«Педагогічні AI агенти: проектування та імплементація»

Виконавець: студент групи ІАСм-1-24-1.4д
Остапенко Дмитро Андрійович

1. Вихідні дані: *Наукові публікації про проектування педагогічних AI-агентів та інтелектуальних тьюторських систем; технічна документація на мову програмування Python, фреймворк Django, систему управління базами даних PostgreSQL/SQLite та технології фронтенду (JavaScript/AJAX); документація API для інтеграції великих мовних моделей (Ollama, Groq), матеріали з проєктування архітектури MVC, а також аналіз існуючих платформ адаптивного навчання.*

2. Основні завдання: *Здійснити огляд наукових джерел та практичних рішень у сфері проєктування педагогічних AI-агентів та інтелектуальних тьюторських систем. Обґрунтувати мету, об'єкт, предмет та психолого-педагогічні основи взаємодії «людина-AI». Розробити завдання, структуру та зміст дипломної роботи. Проаналізувати сучасні підходи до інтеграції великих мовних моделей (LLM) та обрати оптимальний технологічний стек. Спроєктувати модульну архітектуру системи, включаючи сервісний шар AI, серверний Web API на Django, базу даних та клієнтський інтерфейс. Реалізувати програмний продукт: уніфікований інтерфейс для роботи з локальними (Ollama) та хмарними (Groq) моделями, API для генерації пояснень, механізми кешування відповідей та панель викладача для модерації контенту. Розробити та реалізувати алгоритми аналізу помилок студентів та генерації персоналізованих рекомендацій. Провести функціональне та користувацьке тестування, проаналізувати результати та сформувати висновки. Оформити*

дипломну роботу згідно з методичними рекомендаціями кафедри, підготувати наукову публікацію (або тези) та презентацію для захисту .

3. Пояснювальна записка: *Обсяг – до 60 сторінок формату А4 комп'ютерного набору з дотриманням вимог стандарту та методичних рекомендацій кафедри; містить теоретичний аналіз, опис архітектури та реалізації системи Педагогічного AI агента, методику та результати експериментів, висновки та список використаних джерел.*

4. Графічні матеріали: *Структурна схема модульної архітектури педагогічного AI-агента (взаємодія Presentation – Controllers – AI Services – Data), схема ендпоінта генерації пояснень, алгоритм генерації AI-пояснень помилок, ER-діаграма бази даних навчальної платформи, діаграма критичних точок логування системи, екрани веб-інтерфейсу (кабінет студента з рекомендаціями, проходження тесту, панель викладача), а також діаграми порівняльної продуктивності локального та хмарного AI-сервісів.*

5. Додатки: Додаток А, Додаток Б, Додаток В, Додаток Г, Додаток Д, Додаток Е.

6. Строк подання роботи на кафедру: *«1 » грудня 2025 р.*

Науковий керівник
д.п.н., професор:
Смирнова-Трибульська Є.М.
_____
дата

Виконавець:
Остапенко Д.А.
_____
дата

РЕФЕРАТ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

Дипломна робота: 100с., 10 рис., 20 таб., 6 додатків, 40 посилань.

Актуальність: обумовлена стрімкою інтеграцією технологій штучного інтелекту в освітню сферу та зростаючою потребою в адаптивних навчальних системах. Традиційні моделі тестування часто обмежуються лише констатацією результату, не надаючи пояснень. Розробка педагогічних AI-агентів дозволяє вирішити цю проблему, забезпечуючи студентам миттєвий персоналізований зворотний зв'язок, що імітує взаємодію з реальним тьютором, та автоматизує рутинну роботу викладача .

Об'єкт дослідження: процеси проектування, імплементації та функціонування інформаційних технологій навчання на основі педагогічних AI агентів.

Предмет дослідження: моделі, методи та програмні засоби для проектування й реалізації педагогічних AI агентів, а також принципи їхньої адаптивної взаємодії з користувачем в освітньому середовищі.

Мета роботи: розробка та програмна імплементація архітектури педагогічного AI агента, здатного до ефективної та адаптивної взаємодії в навчальному процесі, та дослідження його функціональності шляхом інтеграції у веб-систему тестування.

Завдання роботи:

1. Провести аналітичний огляд існуючих підходів та інструментів (ITS, чат-боти, LLM) для створення педагогічних агентів.
2. Розробити концепцію та модульну архітектуру AI-агента, включаючи вимоги до ПЗ та інтерфейсу.
3. Здійснити програмну реалізацію компонентів агента (інтеграція локальних та хмарних LLM, RAG, кешування).
4. Розробити методику та провести комплексне функціональне й користувацьке тестування системи.

5. Проаналізувати результати тестування та сформулювати рекомендації щодо вдосконалення системи .

Методи дослідження: У роботі використано методи системного аналізу та синтезу (для проектування архітектури), методи теорії алгоритмів та об’єктно-орієнтованого проектування (для програмної реалізації), методи математичної статистики (для обробки результатів) та емпіричні методи (тестування, анкетування) для оцінки ефективності .

Практичне значення дослідження: полягає у створенні програмного модуля педагогічного AI-агента на базі Python/Django, який інтегрується в LMS. Розроблена система замикає цикл «тестування-аналіз-рекомендація»: вона аналізує помилки студента, автоматично генерує структуровані пояснення («чому невірно», «як мислити», фрагмент лекції) та надає інструменти модерації контенту для викладача. Це посилює компонент самостійної роботи студента та підвищує ефективність навчання .

Ключові слова: *штучний інтелект, педагогічний агент, AI-агент, інтелектуальна система навчання, адаптивне навчання, персоналізація, система тестування, зворотний зв'язок, комп'ютерні науки*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ І СПЕЦІАЛЬНИХ ТЕРМІНІВ

AI – Штучний інтелект (Artificial Intelligence)

БД – База даних (Database)

ВММ – Велика мовна модель (Large Language Model, LLM)

ОПМ – Обробка природної мови (Natural Language Processing, NLP)

СУН – Система управління навчанням (Learning Management System, LMS)

ШІ – Штучний інтелект (Artificial Intelligence)

API – Інтерфейс прикладного програмування (Application Programming Interface)

BERT – Bidirectional Encoder Representations from Transformers

CLT – Теорія когнітивного навантаження (Cognitive Load Theory)

DF – Google Dialogflow

EPA – Embodied Pedagogical Agents (Втілені педагогічні агенти)

GPT – Generative Pre-trained Transformer

HTTP – Протокол передачі гіпертексту (Hypertext Transfer Protocol)

ITS – Інтелектуальна тьюторська система (Intelligent Tutoring System)

JSON – Нотація об'єктів JavaScript (JavaScript Object Notation)

LUIS – Language Understanding Intelligent Service

MBF – Microsoft Bot Framework

ML – Машинне навчання (Machine Learning)

NLU – Розуміння природної мови (Natural Language Understanding)

NPS – Індекс чистої підтримки (Net Promoter Score)

ORM – Об'єктно-реляційне відображення (Object-Relational Mapping)

OWL – Web Ontology Language

Q&A – Питання та відповіді (Questions and Answers)

RAG – Retrieval-Augmented Generation (Генерація з доповненим пошуком)

RDF – Resource Description Framework

REST – Передача репрезентативного стану (Representational State Transfer)

SDT – Теорія самодетермінації (Self-Determination Theory)

SQL – Мова структурованих запитів (Structured Query Language)

SUS – Шкала зручності використання системи (System Usability Scale)

UI – Користувацький інтерфейс (User Interface)

UX – Користувацький досвід (User Experience)

XP – Experience Points (Бали досвіду)

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційного суспільства характеризується стрімкою інтеграцією технологій штучного інтелекту (Artificial Intelligence, AI) в усі сфери людської діяльності, зокрема й в освіту. Традиційні моделі навчання дедалі частіше доповнюються цифровими інструментами, що відкриває нові горизонти для персоналізації, адаптації та підвищення доступності освітнього процесу.

Особливої актуальності набуває розробка педагогічних AI-агентів - інтелектуальних програмних систем, здатних імітувати роль викладача, тьютора або партнера у навчанні, надаючи студентам миттєвий зворотний зв’язок та підтримку. Це зумовлено зростаючою потребою в адаптивних навчальних системах, які можуть підлаштовуватися під індивідуальний темп, стиль навчання та рівень знань кожного здобувача освіти. Педагогічні AI-агенти мають значний потенціал для вирішення таких викликів, як перевантаженість викладачів, необхідність диференційованого підходу та забезпечення безперервності навчання поза межами аудиторії. Водночас проєктування та імплементація ефективних агентів є складним науково-технічним завданням, що вимагає поєднання потужних алгоритмів AI з глибоким розумінням психолого-педагогічних аспектів взаємодії «людина–комп’ютер», що й визначає актуальність обраної теми дослідження.

Об’єктом дослідження є процеси проєктування, імплементації та функціонування інформаційних технологій навчання на основі педагогічних AI-агентів.

Предметом дослідження є моделі, методи та програмні засоби для проєктування й реалізації педагогічних AI-агентів, а також принципи їхньої адаптивної взаємодії з користувачем в освітньому середовищі.

Метою роботи є розробка та програмна імплементація архітектури педагогічного AI-агента, здатного до ефективної та адаптивної взаємодії в навчальному процесі, а також дослідження його функціональності.

Завдання роботи. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналітичний огляд існуючих підходів, класифікацій та інструментальних засобів для створення педагогічних AI-агентів, а також дослідити психолого-педагогічні основи їхньої взаємодії з людиною.
2. Розробити концепцію та архітектуру педагогічного AI-агента, що включає його комп'ютерну модель, вимоги до програмного забезпечення, користувацького інтерфейсу та навчального контенту.
3. Здійснити програмну реалізацію ключових компонентів розробленого педагогічного AI-агента.
4. Розробити методику та провести комплексне функціональне й користувацьке тестування для оцінювання ефективності роботи створеної системи.
5. Проаналізувати отримані результати тестування, виявити напрями покращення та сформулювати рекомендації щодо подальшого вдосконалення системи.

Методи дослідження. Для вирішення поставлених завдань у роботі використано комплекс загальнонаукових та спеціальних методів:

- *методи системного аналізу та синтезу* для огляду предметної області та визначення архітектури системи;
- *методи теорії алгоритмів та об'єктно-орієнтованого проєктування* - для розробки комп'ютерної моделі та програмної реалізації агента;
- *методи математичної статистики* – для обробки результатів тестування;
- *методи емпіричного дослідження (тестування та анкетування)* – для оцінки ефективності системи та користувацького досвіду.

Практичне значення одержаних результатів полягає у розробці програмного модуля педагогічного AI-агента, призначеного для інтеграції в існуючі системи

тестування та управління навчанням (LMS). На відміну від стандартних систем, що обмежуються констатацією результату, розроблений агент надає студенту миттєвий, персоналізований та змістовний зворотний зв'язок. Система аналізує відповіді користувача, ідентифікує слабкі теми та автоматично генерує педагогічно обґрунтовані рекомендації з посиланням на навчальні матеріали. Це дозволяє замкнути цикл «тестування-аналіз-рекомендація», посилює компонент самостійної роботи студента та автоматизує частину рутинної роботи викладача. Розроблена архітектура може бути використана як шаблон для модернізації існуючих освітніх платформ .

Структура роботи: Дипломна робота складається зі вступу, трьох розділів основної частини, висновків, списку використаних джерел та додатків. У першому розділі подано аналіз теоретичних основ проектування педагогічних AI-агентів та інтелектуальних тьюторських систем, досліджено психолого-педагогічні аспекти взаємодії «людина–AI», а також проведено огляд існуючих рішень і вибір технологічного стеку (Python, LLM, RAG) для реалізації системи. Другий розділ присвячений обґрунтуванню модульної архітектури агента, опису взаємодії компонентів (Django Backend, AI Services, PostgreSQL) та інтерфейсу користувача, а також програмній реалізації алгоритмів аналізу помилок і генерації персоналізованих пояснень. У третьому розділі наведено методику оцінювання ефективності, результати функціонального, навантажувального та UX-тестування для локального (Ollama) та хмарного (Groq) режимів роботи, а також аналіз отриманих показників продуктивності й рекомендації щодо вдосконалення системи

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД

Розділ закладає теоретичний фундамент для проектування педагогічного AI агента, що є основною метою кваліфікаційної роботи. Проведено аналіз предметної області, визначено сутність, класифікацію, сучасні підходи до створення агентів та психолого-педагогічні основи взаємодії «людина-AI». Виконано огляд існуючих програмних продуктів та інструментальних засобів для обґрунтування вибору архітектури проектованої системи.

1.1. Поняття та класифікація педагогічних AI агентів

Педагогічний AI-агент - це автономна програмна система на основі штучного інтелекту, яка здатна самостійно виконувати освітні завдання, приймати рішення та здійснювати дії для досягнення педагогічних цілей з мінімальним втручанням людини. На відміну від традиційних AI-систем, які реагують на запити користувача, педагогічні AI-агенти діють проактивно, аналізуючи освітнє середовище, плануючи послідовність дій і адаптуючись до потреб учнів у реальному часі [7].

Педагогічний AI-агент характеризується наступними ключовими властивостями [8]:

- автономність - здатність функціонувати незалежно, приймаючи рішення без постійного нагляду людини
- цілеорієнтованість - робота спрямована на досягнення конкретних освітніх результатів
- з датність до міркування - використання складних когнітивних процесів для розуміння проблем і розробки рішень
- адаптивність - коригування стратегій навчання при зіткненні з новими ситуаціями або перешкодами

- безперервне навчання - вдосконалення власної роботи через досвід і зворотний зв'язок.

Термін «педагогічний агент» історично використовувався для опису віртуальних персонажів в освітньому програмному забезпеченні. Однак сучасні педагогічні AI-агенти виходять за межі цього визначення, інтегруючи можливості великих мовних моделей (LLM) та технології генеративного AI для виконання складних багатокрокових освітніх процесів [9].

Інтелектуальна тьюторська система (Intelligent Tutoring System, ITS) - це комп'ютерна система, яка імітує людського репетитора і призначена для надання індивідуалізованої навчальної підтримки та зворотного зв'язку учням, зазвичай без втручання людини-викладача. ITS використовує технології штучного інтелекту для створення персоналізованого освітнього досвіду, адаптуючись до індивідуального рівня знань і темпу навчання кожного студента [9].

Класична архітектура ITS складається з чотирьох основних компонентів:

- М одель предметної області (Domain Model) містить експертні знання з навчального предмету
- Модель учня (Student Model) відстежує прогрес, знання, помилки та когнітивний стан учня
- Педагогічна модель (Tutor Model) визначає навчальні стратегії та методи втручання
- Інтерфейс користувача (User Interface) забезпечує взаємодію між учнем і системою

Порівняльний аналіз представлено в таблиці 1.1

Таблиця 1.1

Порівняння AI-агента та ITS

Характеристика	**Педагогічний AI-агент**	**Інтелектуальна тьюторська система (ITS)**
Ступінь автономності	Висока; може самостійно ініціювати дії та приймати	Середня; працює в межах запрограмованої архітектури та педагогічних правил

Характеристика	**Педагогічний AI-агент**	**Інтелектуальна тьюторська система (ITS)**
	рішення без попередньо визначених сценаріїв	
Архітектура	Гнучка, модульна; може інтегруватися з різними зовнішніми інструментами та API	Структурована чотирикомпонентна архітектура (Domain Model, Student Model, Tutor Model, Interface)
Область застосування	Широка; може виконувати адміністративні, навчальні та аналітичні завдання	Сфокусована на індивідуалізованому навчанні та репетиторстві
Адаптивність	Динамічна; навчається на основі досвіду та змінює стратегії в реальному часі	Адаптується на основі попередньо визначених педагогічних моделей і правил
Здатність до планування	Може розбивати складні завдання на підзадачі та виконувати багатокрокове планування	Слідує педагогічним стратегіям, визначеним у Tutor Model
Взаємодія з середовищем	Проактивна; може ініціювати втручання та взаємодіяти з зовнішніми системами	Реактивна; відповідає на дії учня в межах навчального середовища

Основна відмінність полягає в тому, що ITS є структурованою системою з чітко визначеними компонентами і педагогічними моделями, тоді як педагогічний AI-агент представляє більш автономну, цілеорієнтовану сутність, здатну до багатокрокового міркування і самостійного прийняття рішень. При цьому педагогічний агент може бути реалізований як компонент ITS (наприклад, як інтерфейс або частина Tutor Model), але також може функціонувати як самостійна система з ширшим спектром можливостей [9].

Освітній чат-бот - це AI-система на основі обробки природної мови, призначена для ведення текстових діалогів з користувачами в освітньому контексті. Чат-боти відповідають на запити студентів, надають навчальну інформацію, допомагають з адміністративними питаннями та забезпечують підтримку 24/7.

Таблиця 1.2

Порівняльний наналіз AI-агента та чат-бота

Характеристика	**Педагогічний AI-агент**	**Освітній чат-бот**
Основна функція	Автономне виконання складних освітніх завдань і прийняття рішень	Ведення діалогу та надання відповідей на запити користувачів
Стиль взаємодії	Цілеорієнтована діяльність, часто з мінімальною взаємодією	Розмовний інтерфейс на основі текстових запитань і відповідей
Автономність	Висока; може самостійно визначати хід дій і виконувати багатокрокові процеси	Обмежена; реагує на запити користувача та працює в межах діалогових сценаріїв
Здатність до виконання завдань	Може виконувати дії в різних системах, аналізувати дані, генерувати контент і автоматизувати процеси	Обмежується наданням інформації та простими транзакціями в межах чату
Планування	Здатний до багатокрокового планування та декомпозиції складних завдань	Слідує попередньо визначеним діалоговим потокам або дає відповіді на основі запитів
Навчання та адаптація	Безперервне навчання з досвіду, динамічна адаптація стратегій	Обмежене машинне навчання в межах специфічної області; складність адаптації до нових ситуацій
Приклади застосування	Персоналізовані траєкторії навчання, автоматизоване оцінювання, предиктивна аналітика	Відповіді на питання, допомога з домашніми завданнями, надання ресурсів

Ключова відмінність полягає в тому, що освітні чат-боти зосереджені на конверсаційній взаємодії і надають інформацію в діалоговому форматі, тоді як педагогічні AI-агенти є виконавчими системами, здатними до автономного виконання комплексних освітніх процесів. Чат-бот очікує на запит користувача, щоб надати відповідь, в той час як AI-агент може самостійно ініціювати дії для досягнення освітніх цілей.

Деякі сучасні розвинені освітні чат-боти можуть демонструвати агентні властивості (наприклад, проактивне надання рекомендацій), але вони не досягають рівня автономності та складності прийняття рішень, характерних для повноцінних AI-агентів. Водночас чат-боти можуть

використовуватися як інтерфейс для взаємодії з педагогічними AI-агентами.

У контексті освітніх технологій ці три поняття можна розташувати в порядку зростання складності та складності (освітній чат-бот - інтелектуальна тьюторська система - педагогічний AI-агент), де

- Чат-боти забезпечують конверсаційний інтерфейс і базову підтримку;
- ITS надають структуроване персоналізоване навчання на основі педагогічних моделей;
- Педагогічні AI-агенти представляють автономні цілеорієнтовані системи з найвищим рівнем самостійності прийняття рішень.

Важливо зазначити, що ці технології не є взаємовиключними, оскільки педагогічний агент може бути компонентом ITS, а чат-бот може служити інтерфейсом для обох систем.

Далі розглянемо детальну класифікаціу педагогічних AI агентів за їхньою педагогічною роллю.

1. Тьютор/Експерт (Tutor/Expert)

Педагогічний агент-тьютор або експерт виконує функцію віртуального наставника, який пояснює навчальний матеріал, дає поради щодо засвоєння знань, відповідає на запитання учня та допомагає знаходити оптимальні шляхи досягнення навчальних цілей. Такий агент надає структуровану підтримку та активно керує освітньою траєкторією користувача.

Особливості:

- ініціює навчальні кроки;
- пояснює нові теми;
- проводить діагностику знань;
- пропонує вправи та корегує помилки.

2. Ментор (Mentor)

Педагогічний агент-ментор супроводжує учня в питаннях особистісного розвитку, допомагає визначати сильні сторони та освітні потреби, сприяє

формуванню цілей і мотивує до саморозвитку. Ментор наголошує не лише на академічних знаннях, а й на розвитку «м’яких навичок», професійному самовизначенні, формуванні освітньої траєкторії.

Особливості:

- дає поради і рекомендації з урахуванням майбутньої кар’єри;
- підтримує у вирішенні неакадемічних питань;
- мотивує та надихає до самонавчання і самовдосконалення.

3. Асистент (Assistant)

Педагогічний агент-асистент виконує допоміжну роль, підтримуючи учня у виконанні рутинних або технічних завдань: надсилає нагадування, організовує графік, надає ресурси, відповідає на базові питання та полегшує взаємодію учня з навчальним середовищем.

Особливості:

- сумісний з функціями секретаря чи помічника;
- автоматично організовує навчальний процес;
- відповідає на часті запитання;
- оперативно шукає довідкову інформацію.

4. Партнер по навчанню/Ко-учень (Learning Partner/Peer/Co-Learner)

Такий агент моделює поведінку «рівного», супроводжує учня у навчанні, допомагає долати труднощі, співпрацює в групових завданнях, стимулюючи елементи колаборації та розвиваючи навички командної роботи.

Особливості:

- бере участь у спільних обговореннях;
- підказує, але не «веде» учня;
- сприяє розвитку критичного мислення шляхом рівноправного діалогу;
- стимулює відповідальність за результат через рівне партнерство.

Насправді, сучасний педагогічний AI-агент може комбінувати декілька педагогічних ролей, динамічно їх змінюючи залежно від освітньої ситуації, контексту і рівня самостійності учня.

Також існує класифікація педагогічних AI агентів за способом втілення (embodiment) та модальністю взаємодії [7; 10; 11]:

1. Текстові (Text-based agents) - взаємодіють з користувачем за допомогою текстового чату та повідомлень. Вони не мають візуального образу, працюють у вигляді чат-бота чи текстового тьютора на освітніх платформах.

2. 2D аватари (2D avatars) - використовують графічне двовимірне зображення персонажа, аватара або іконки - для унаочнення присутності агента у навчальному середовищі. Такі агенти можуть відображати мінімальні жести чи анімацію, додатково до тексту .

3. 3D аватари (3D avatars, Embodied agents, EPA) - мають тривимірну модель, можуть демонструвати складні візуальні жести, рухи, вирази обличчя, мовну артикуляцію. Часто використовуються в ігрових, VR- або AR-освітніх середовищах - агенти набувають форми повного «віртуального репетитора» чи «персонажа», здатного озвучувати фрази та взаємодіяти з учнем майже як реальна людина.

Більшість сучасних агентів можуть поєднувати кілька способів втілення та змішану модальність взаємодії: наприклад, 3D аватар із голосовим озвученням, текстовий агент із мінімальною анімацією і т.д. Вибір embodiment та modality залежить від цілей освітнього проєкту й технологічних можливостей навчального середовища.

Класифікація педагогічних агентів за технологічною архітектурою наступна:

1. Агенти на основі правил (Rule-based agents) рацюють згідно із заздалегідь прописаними правилами (зразок «if-then»):

- всі сценарії поведінки задаються розробниками
- такі агенти не здатні до навчання, діють лише в межах окресленої логіки
- використовуються для простих чат-ботів, систем автоматичного оцінювання, тестування

Наприклад, чат-бот з жорстко визначеним сценарієм відповіді на запитання, особливостями якого є прозорість роботи, передбачуваність, обмежена гнучкість.

2. Агенти на основі машинного навчання (Machine learning-based agents), що використовують алгоритми машинного навчання:

- здатні аналізувати дані, виявляти закономірності
- навчаються на освітніх даних, прогнозують успіх/помилки учнів
- можуть адаптувати свою поведінку та рекомендації на основі накопиченого досвіду

Наприклад, агент, який адаптує навчальний контент відповідно до оцінок учня або автоматично виявляє прогалини у знаннях

3. Агенти на базі великих мовних моделей (LLM-based agents), які працюють за допомогою великих (глибоких) нейронних мовних моделей (GPT, BERT і подібних):

- надзвичайна гнучкість у генерації, розумінні тексту, діалогів та контекстів;
- можуть планувати складні багатокрокові дії, створювати персоналізований освітній досвід;
- виконують роль автономних тьюторів або асистентів, інтегруються з зовнішніми освітніми ресурсами;

Наприклад, сучасні педагогічні AI-агенти, що інтегрують навички планування, генерації навчальних матеріалів, моделювання діалогу

Вибір архітектури визначає рівень автономності, адаптивності і складності освітнього агента, а також можливості щодо персоналізації навчання.

1.2. Аналіз сучасних підходів до створення педагогічних агентів

Інтелектуальні тьюторські системи (ITS) є важливою складовою сучасної освітньої технології. Класична архітектура ITS складається з декількох ключових модулів, які забезпечують адаптивність та індивідуалізацію навчального процесу. Завдяки структурованій організації ITS стають потужним інструментом персоналізованого навчання [9].

До основних компонентів входять:

- Модуль знань (Domain Model / Expert Model)
- Модуль студента (Student Model)
- Педагогічний модуль (Tutoring/Pedagogical Model)
- Модуль інтерфейсу (User Interface Module)

Опишемо детально взаємодію між цими компонентами. На рисунку 1.1. зображено класичну архітектуру ITS, яка складається з чотирьох основних модулів. Модуль знань містить інформацію про навчальну предметну область, модуль студента збирає дані про успішність та особливості учня, педагогічний модуль визначає освітню стратегію, а інтерфейс забезпечує безпосередню взаємодію з користувачем.

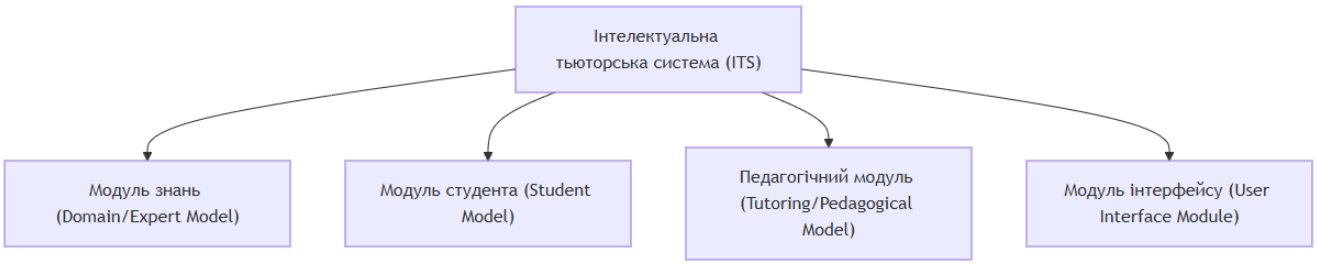


Рисунок 1.1 - Ключові компоненти класичної ITS

Взаємодія всіх складових ITS дозволяє створити гнучке, адаптивне і максимально ефективне освітнє середовище для кожного студента, підтримуючи якість, мотивацію та індивідуальний прогрес.

Сучасні підходи до моделювання знань для педагогічних агентів демонструють як спадкоємність із класичними методами представлення знань, так і активне впровадження новітніх технологій на базі векторних та гібридних структур. Існує два типи підходів - традиційні та новітні. Розглянемо традиційні підходи до Knowledge Representation:

1. Онтології (Ontologies) - це формальні, ієрархічно організовані структури, що описують концепти предметної області та зв’язки між ними. Використовують мови типу OWL, RDF. Із переваг - формальна строгость, прозорість, можливість

автоматичного виведення знань. Серед недоліків - складність розробки та супроводу, погано масштабуються для нефіксованих, дуже великих знань [15].

2. Семантичні мережі (Semantic networks) - це графи, у яких вузли є об’єктами/поняттями, а ребра - це семантичні відношення. Застосовуються для візуалізації, пояснення і пошуку зв’язків між поняттями. Переваги - наочність, зручність інтеграції з іншими методами. Недолііки - фрагментарність, відсутність суворої формалізації складних залежностей.

3. Бази правил (Rule-based systems) - це знання представляються у вигляді набору логічних правил («якщо – то»). Їх переваги в простоті, ефективності для вузьких предметних областей. Недоліки - слабка масштабованість, обмежена можливість самонавчання.

Також розглянемо новітні підходи до представлення знань, які представляються двома типами систем:

1. Векторні бази даних (Vector Databases, Embedding Stores) для RAG (Retrieval-Augmented Generation) [16]:

- знання кодуються у вигляді векторних представлень (ембеддингів); пошук базується на схожості у багатовимірному просторі;
- агент отримує запит, перетворює його у вектор, знаходить найближчі фрагменти знань (документи, факти, навчальні матеріали) та надає релевантну відповідь;

Переваги - це чудова масштабованість, гнучкість при роботі з неструктурованими і великими обсягами даних, продуктивність у RAG-підходах до генерації відповідей LLM-агентом. Недоліки - менш наочне для людини структурування, обмежена explainability.

2. Гібридні нейросимвольні системи [17]:

- комбінують класичну символічну (онтологічну, графову) структуру з можливостями векторних моделей;
- дозволяють як робити пошук за подібністю, так і використовувати строгі логічні зв’язки та обґрунтування;

- найбільш перспективні для персоналізованих та адаптивних педагогічних агентів у контексті LLM та RAG.

В таблиці 1.3 представлено порівняння традиційних та новітніх підходів до представлення знань.

Таблиця 1.3

Порівняння підходів до представлення знань

Підхід	**Пояснюваність**	**Гнучкість та масштабованість**	**Інтеграція з LLM/RAG**	**Дедукція/Інференція**
Онтології, сем. мережі	Висока	Середня/низька	Обмежена, потребує додаткових шарів	Сильна
Бази правил	Висока	Низька	Погана	Легка/локальна
Векторні бази для RAG	Низька	Висока	Пряма	Слабка (без графу)
Гібридні нейросимволічні	Середня	Висока	Відмінна	Комбінована

Сучасні педагогічні агенти поступово переходять від класичних експертних ієрархій та логічних правил до гібридних моделей, у яких великі мовні моделі використовують як структуровані (онтології, знань-ґрафи), так і неструктуровані (векторні бази, embedding stores) джерела знань для реалізації Retrieval-Augmented Generation та інших просунутих алгоритмів інтерактивного навчання.

Сучасні AI-агенти застосовують багаторівневі методи моделювання студента для персоналізації навчання, створення індивідуальних маршрутів й ефективного подолання прогалин у знаннях. Системи аналізують відповіді, динаміку успішності, зібрані дані про дії та помилки учнів, автоматично реалізуючи адаптивну логіку навчання.

Рисунок 1.2 показує, як AI-агент збирає дані про дії учня, формує актуальну модель його знань, аналізує прогалини на основі виявлених помилок і забезпечує адаптивне навчання через рекомендації та коригування освітньої траєкторії.

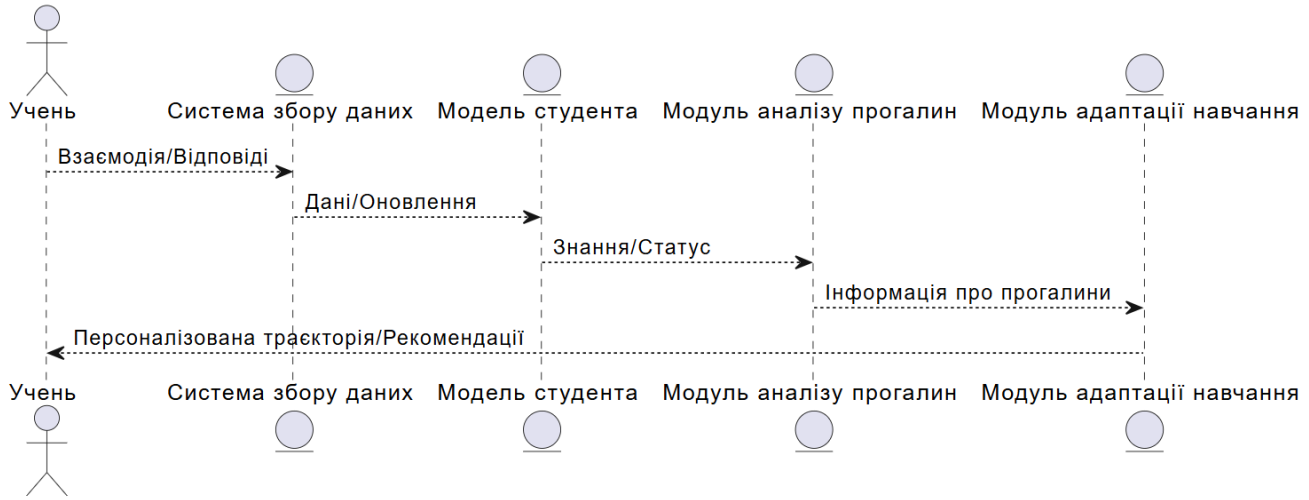


Рисунок 1.2 - Адаптивне моделювання студента в AI-агенті

Цей підхід дозволяє освітнім AI-агентам ефективно персоналізувати навчання, допомагати учню долати складнощі та формувати оптимальний шлях до освітнього результату.

Сучасні AI-агенти для навчання інтегрують різноманітні педагогічні стратегії, щоб зробити діалог з учнем продуктивним, підтримуючим і мотивуючим. Вони не лише надають знання, а й сприяють розвитку критичного мислення, самостійності та соціальних навичок через персоналізовані сценарії взаємодії (рис. 1.3).

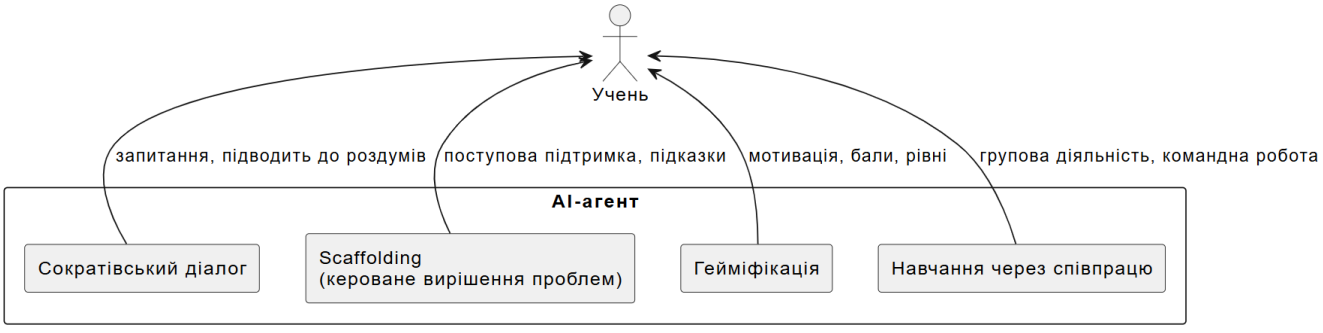


Рисунок 1.3 - Педагогічні стратегії у діалогах AI-агентів

Схема ілюструє провідні стратегії взаємодії: Сократівський діалог для розвитку критичного мислення, кероване вирішення проблем (scaffolding) для поступової допомоги, гейміфікацію для мотивації та навчання через співпрацю для соціальних навичок.

Впровадження цих стратегій забезпечує гнучку підтримку та допомагає учням розвиватися не лише інтелектуально, а й особистісно у процесі навчального діалогу з AI-агентом.

Великі мовні моделі (LLM) і генеративний AI кардинально змінили можливості сучасних педагогічних агентів. Вони дозволяють генерувати осмислені, багатокрокові діалоги та відповіді, розуміти контекст і адаптуватися до індивідуальних потреб користувача у реальному часі. RAG-архітектура та інжиніринг промптів - ключові технології для створення таких агентів (рис. 1.4).

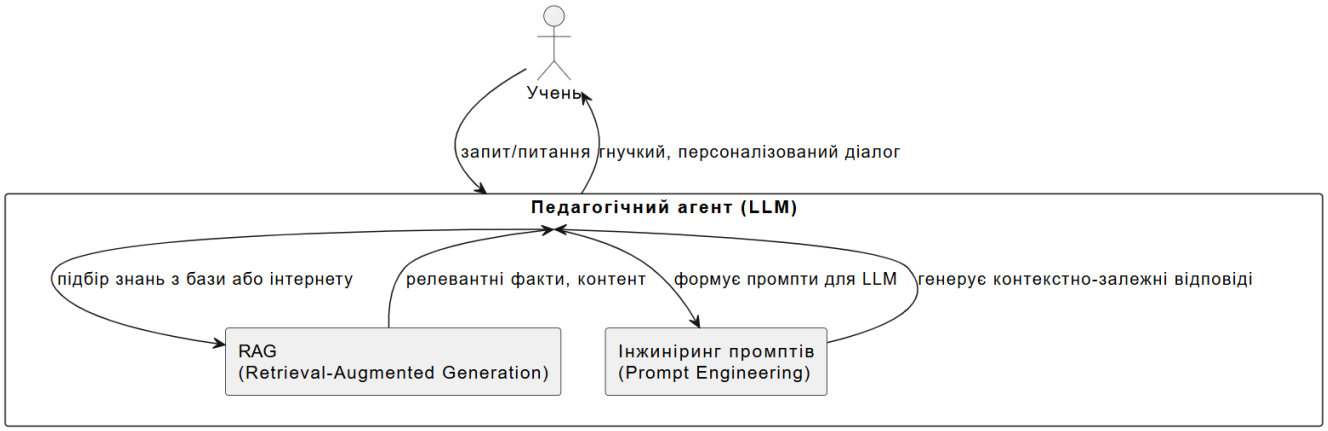


Рисунок 1.4 - RAG та інжиніринг промптів у сучасних педагогічних агентів

Учень звертається до агенту, який через RAG-архітектуру знаходить релевантну інформацію, а за допомогою інжинірингу промптів створює відповіді, враховуючи контекст і цілі навчання. Усе це забезпечує високу гнучкість і персоналізацію освітньої взаємодії.

Інтеграція LLM разом з RAG та продуманим інжинірингом промптів дає педагогічним агентам потужний інструментарій для підтримки індивідуалізованого та контекстного навчання на сучасному рівні.

Отже, впровадження сучасних педагогічних AI агентів супроводжується низкою серйозних викликів і обмежень:

- великі мовні моделі схильні до «галюцинацій» - генерації неправдивої або некоректної інформації, що може вводити учня в оману;

- оцінка реального засвоєння знань залишається складною проблемою: автоматичні методи не завжди точно відображають глибину розуміння матеріалу й розвиток навичок;
- етичні питання - небезпека упередженості в рекомендаціях, відповідальність за результат навчання, необхідність забезпечення прозорості алгоритмів;
- критично важлива конфіденційність даних та захист особистої інформації учнів, оскільки сучасні агенти оперують великими масивами освітніх даних.

У сукупності ці фактори вимагають від розробників педагогічних AI агентів системного підходу до контролю якості, етики та безпеки, а також постійного вдосконалення алгоритмів для створення надійних, відповідальних і корисних освітніх технологій.

1.3. Психолого-педагогічні основи взаємодії «людина–AI»

Взаємодія людини зі штучним інтелектом у навчальному контексті має глибокі психологічні й педагогічні виміри, які визначають ефективність освітнього процесу, мотивацію учнів та їх готовність довіряти технологіям. Сучасні дослідження 2020-2025 років фокусуються на gсихолого-педагогічнsq взаємоді] людини з AI-агентами, що включає низку ключових аспектів, які визначають ефективність, якість і безпечність освітнього процесу:

- довіра до AI-агентів як психологічна основа взаємодії
- когнітивне навантаження та теорія когнітивного навантаження (CLT)
- емпатія та соціальна перцепція AI
- мотивація та теорія самодетермінації (SDT)
- надмірна довіра до AI (Overtrust)
- етичні та культурні аспекти використання AI
- баланс між автоматизацією та людською взаємодією
- критичне мислення у взаємодії з AI
- конфіденційність та захист персональних даних

Довіра є одним із ключових чинників, що визначають ефективність використання AI у навчанні. Дослідження в цьому напрямі показують, що довіра до AI формується на основі трьох вимірів:

- характеристик користувача (trustor),
- властивостей AI-системи (trustee),
- контексту взаємодії.

Також встановлено, що довіра до штучних агентів визначається не лише технічними можливостями системи, а й психологічними чинниками - прозорістю алгоритмів, передбачуваністю дій і відповідністю очікуванням користувачів [18].

Українські дослідники також наголошують на важливості довіри у взаємодії з AI. У дослідженні, проведеному в умовах воєнного стану підкреслюється, що «психологічний тип» AI-інструментів та їх вплив на динаміку «студент-викладач» є критичними для успішної інтеграції технологій у навчальний процес [19].

Теорія когнітивного навантаження (Cognitive Load Theory, CLT) є центральною у дослідженнях психологічних основ взаємодії з AI-агентами. Мета-аналіз на основі 24 досліджень показав, що педагогічні агенти можуть як знижувати, так і підвищувати когнітивне навантаження залежно від дизайну та контексту використання. Дослідження підтверджують, що правильно розроблені AI-агенти сприяють ефективному використанню когнітивних ресурсів, допомагаючи учням зосередитись на важливому навчальному матеріалі [19].

Однак надмірна залежність від AI може зменшувати «germane load» - когнітивне навантаження, необхідне для глибокого навчання та розвитку навичок вищого порядку. Українські дослідники також звертають увагу на цей аспект, підкреслюючи, що використання ШІ може як покращити навчальний процес через персоналізацію, так і негативно вплинути на критичне мислення студентів.

Емпатія та здатність AI-агентів «розуміти» емоційний стан користувача відіграють важливу роль у формуванні довіри та ефективності навчання. Дослідники визначили вимоги до емпатійних культурно-обізнаних агентів :

- пояснюваність поведінки,

- спостереження за соціальними сигналами (міміка, жести),
- інтерпретація висловлювань та емпатійна взаємодія.

Українське дослідження (2025) показало, що сприйняття емпатійності AI-агента залежить від особистості користувача, його настрою, контексту взаємодії та попереднього досвіду використання технологій. Навіть незначні затримки у відповіді або збої в роботі системи можуть негативно вплинути на сприйняття агента як емпатійного [21].

Теорія самодетермінації (Self-Determination Theory, SDT) підкреслює три ключові психологічні потреби - автономію, компетентність і приналежність, - які є критичними для мотивації та навчання. AI повинен підтримувати ці потреби: створювати відчуття компетентності через персоналізоване навчання, зберігати автономію учня у прийнятті рішень і не руйнувати соціальну приналежність через надмірну автоматизацію взаємодії [19].

Дослідження в Україні (2025) серед студентів-філологів показало, що AI-інструменти підвищують мотивацію та залученість до навчання завдяки персоналізації, інтерактивності та миттєвому зворотному зв'язку. Однак важливо забезпечити баланс між використанням AI і збереженням людської взаємодії [20].

Проблема надмірної довіри до AI є критичною у контексті освіти. Дослідження показують, що учасники експериментів демонстрували високий рівень довіри до рекомендацій AI-агентів навіть тоді, коли ці рекомендації були випадковими та ненадійними. Це вказує на схильність людей надмірно довіряти AI, що може мати негативні наслідки для розвитку критичного мислення та самостійності у навчанні [20].

Українські дослідники підкреслюють важливість етичного використання AI в освіті та наголошують на необхідності розробки чітких рамок та механізмів нагляду для етичної імплементації AI у вищій освіті України. Важливо визначити прийнятні сфери застосування AI, забезпечити прозорість роботи систем, розвивати критичну AI-грамотність серед студентів і викладачів.

1.4. Аналіз існуючих систем та інструментів створення AI агентів

Для того, щоб зрозуміти практичну реалізацію та ефективність нового покоління педагогічних AI-агентів та інтелектуальних навчальних систем, важливо проаналізувати ключових гравців ринку. Детально розглянемо кілька провідних прикладів:

- гейміфікований підхід Duolingo у вивченні мов [22];
- Сократівськмй метод тьютора Khanmigo (Khan Academy) [23];
- глибокий когнітивний тьюторинг MATHia (Carnegie Learning) у математиці [24];
- вузькоспеціалізований AI-асистент платформи roadmap.sh, який демонструє, як подібні технології адаптуються для навігації у складних технічних дисциплінах [25].

1. Duolingo. Цільова аудиторія - будь-хто, хто хоче вивчити нову мову, від початківців до тих, хто хоче підтримати середній рівень.

Ключові функції:

- мікро-уроки (короткі, чітко структуровані уроки, що охоплюють читання, письмо, аудіювання та говоріння);
- гейміфікація (використання балів (XP), "стріків" (безперервних днів занять), ліг та досягнень для мотивації);
- адаптивні тести (періодичні перевірки для оцінки засвоєння матеріалу та коригування складності)
- персоналізована практика (спеціальні сесії "огляду" (Practice Hub), які фокусуються на словах та граматичних конструкціях, з якими у користувача виникали проблеми).

Головна "фішка" Duolingo - це масова гейміфікація та навчання на основі даних (data-driven learning). Їхній AI (Birdbrain) аналізує мільярди відповідей користувачів щодня, щоб визначити, коли саме учень, ймовірно, забуде слово. На

основі цього він створює персоналізовані сесії повторення, використовуючи варіацію техніки інтервального повторення (spaced repetition).

2. Khan Academy (агент Khanmigo). Цільова аудиторія - учні від початкової до старшої школи та студенти коледжів. Також використовується вчителями та для самоосвіти дорослих.

Ключові функції:

- Майстерність (Mastery Learning) (учні повинні довести своє розуміння теми (наприклад, відповісти на 5-7 питань поспіль без помилок), перш ніж перейти до наступної);
- Khanmigo (AI-тьютор) - інтегрований чат-бот, який допомагає учням, не даючи прямих відповідей;
- підтримка для вчителів (Khanmigo допомагає вчителям створювати плани уроків, генерувати ідеї та аналізувати прогрес учнів);

Унікальний підхід Khan Academy зосереджений на майстерності та Сократівському методі. Коли учень застряг на проблемі, Khanmigo не дає відповідь. Замість цього він ставить навідні запитання ("Що ти спробував зробити спочатку?", "Що означає цей термін у рівнянні?"). Це змушує учня розмірковувати та самостійно приходити до рішення, що є ключовим принципом педагогіки Сократа.

3. Carnegie Learning (MATHia). Цільова аудиторія - учні середньої, старшої школи та коледжів, які вивчають математику.

Ключові функції:

- покрокове наставництво (система аналізує кожен крок учня під час розв'язання складної математичної задачі);
- миттєвий зворотний зв'язок (якщо учень робить помилку на одному з кроків (наприклад, неправильно розподіляє множник), система негайно вказує на це і надає підказку "саме вчасно" (just-in-time));
- динамічна адаптація (складність задач та рівень підказок змінюються залежно від успішності учня);

Унікальний підхід MATHia базується на когнітивному тьюторингу (cognitive tutoring). Це не просто перевірка правильної/неправильної відповіді. Система побудована на глибоких когнітивних моделях того, як саме учень думає під час розв'язання задачі. Вона відстежує десятки різних навичок в межах однієї проблеми. Якщо учень робить помилку, MATHia може точно визначити, яку саме когнітивну навичку (наприклад, "розуміння від'ємних чисел") ще не засвоєно, і запропонувати цільову допомогу.

4. Приклад вузькоспеціалізованого AI Асистента на Roadmap.sh

Roadmap.sh - це популярний ресурс для розробників, який надає візуальні "дорожні карти" для вивчення різних технологічних професій (наприклад, Frontend, Backend, DevOps). AI-асистент (також відомий як "AI Tutor" або "Roadmap AI") є чудовим прикладом вузькоспеціалізованого педагогічного агента (рис. 1.5).

Цільова аудиторія - розробники-початківці, студенти комп'ютерних наук та досвідчені розробники, які хочуть змінити спеціалізацію або вивчити нову технологію.

Ключові функції:

- контекстний чат (Chat with Roadmaps). Користувач може відкрити конкретну дорожню карту (наприклад, "AI Engineer Roadmap") і ставити запитання безпосередньо "в контексті" цієї карти;
- пояснення концепцій (можна попросити пояснити будь-який вузол на карті, наприклад, "Що таке RAG?" або "Яка різниця між Embeddings та Vector Databases?").
- пошук ресурсів (асистент може запропонувати навчальні матеріали (статті, відео) для конкретного вузла на карті);
- генерація контенту (може генерувати персоналізовані навчальні плани, курси та вікторини на основі обраної теми або дорожньої карти).
- кар'єрні поради (допомагає користувачам обрати кар'єрний шлях або зрозуміти, що вивчати далі, виходячи з їхнього поточного прогресу).

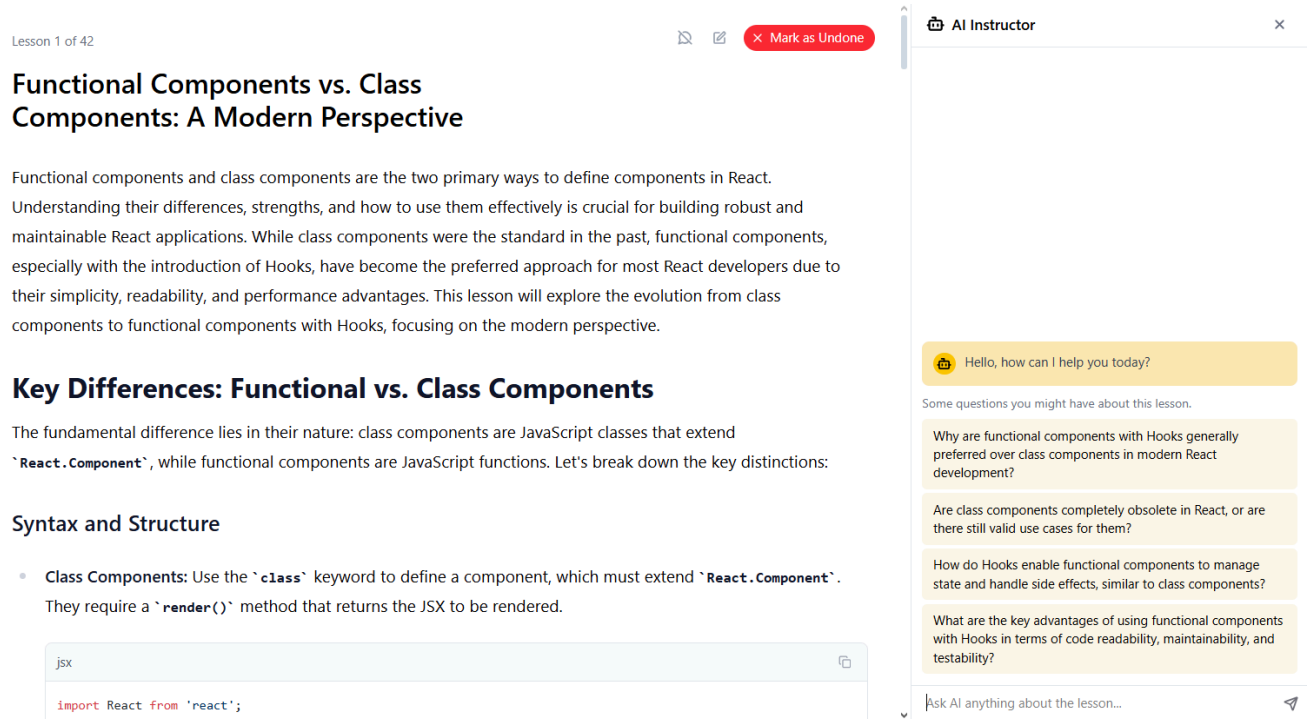


Рисунок 1.5 - Інтерфейс AI Tutor by roadmap.sh

Унікальність цього агента полягає в його глибокій інтеграції зі структурованою базою знань (візуальними дорожніми картами). На відміну від загальних тьюторів, він завжди "знає", де знаходиться користувач у своєму навчальному процесі. Він діє як персональний ментор або навігатор для конкретної, часто дуже складної, технічної дисципліни. Він поєднує можливості великої мовної моделі (пояснення) з чіткою, перевіреною експертами структурою (самими "роадмапами").

Ключова відмінність педагогічного агента від звичайного чат-бота (наприклад, для підтримки клієнтів) полягає у необхідності складного управління діалогом та відстеження стану учня. Тьютору потрібно не просто відповісти на запитання, а вести учня до відповіді, керувати контекстом уроку та адаптуватися до рівня знань.

1. Rasa - це open-source фреймворк, який складається з двох основних компонентів (рис 1.6):

– Rasa NLU (для розуміння намірів)

– Rasa Core (для управління діалогом).

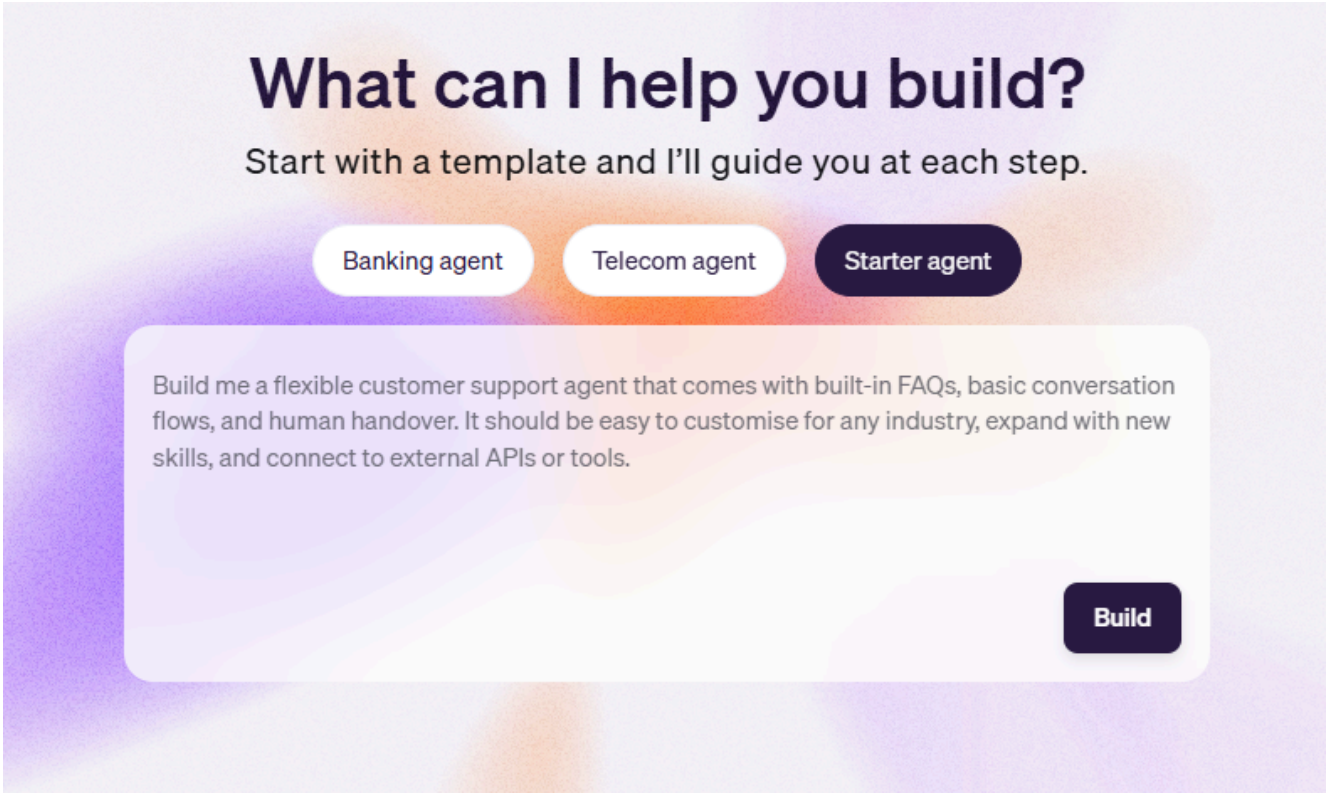


Рисунок 1.6 - Інтерфейс rasa.ai

Переваги для педагогічних агентів:

- повний контроль над логікою. Rasa Core використовує підхід "історій" (Stories) та "правил" (Rules), що дозволяє програмувати дуже складні, нелінійні діалоги. Це ідеально для реалізації Сократівського методу, надання підказок та розгалужених навчальних сценаріїв;
- можна розмістити агента на власному сервері. Це критично для навчальних закладів, які працюють з конфіденційними даними учнів;
- open-source рішення, що дозволяє глибоко модифікувати будь-який аспект NLU або логіки діалогу, що важливо для дослідницьких та дипломних проєктів;
- відстеження стану (Slots). Механізм "слотів" дозволяє легко зберігати інформацію про учня (наприклад, "тема_засвоєна=false", "остання_помилка=...") і використовувати її для керування діалогом.

Недоліки:

- високий поріг входження. Вимагає значних технічних навичок (Python, налаштування інфраструктури);

- "Холодний старт" NLU. Модель NLU потрібно навчати з нуля на власних даних, тоді як хмарні сервіси мають потужні попередньо навчені моделі.

2. Google Dialogflow (DF) - це хмарна платформа від Google (SaaS), яка надає комплексний сервіс для створення розмовних інтерфейсів з потужним NLU.

Переваги:

- найкраще NLU "з коробки". Чудово розуміє природну мову, варіації запитань та контекст завдяки потужним моделям Google. Це зменшує час на налаштування "розуміння" учня;
- швидкий старт та прототипування: Має інтуїтивно зрозумілий графічний інтерфейс. Можна дуже швидко створити прототип простого Q&A тьютора або агента для частих запитань (FAQ);
- легка інтеграція. Просто інтегрується з іншими сервісами Google та веб-сайтами.

Недоліки:

- обмежена логіка діалогу. Dialogflow чудово працює зі сценаріями "запит-відповідь" та збором інформації (наприклад, замовлення піци). Однак реалізація складних, багатоетапних навчальних циклів (де агент повинен ставити навідні запитання) є складною і "неприродною" для платформи;
- "Чорна скринька". Ви маєте обмежений контроль над тим, як NLU приймає рішення;
- залежність від хмари та вартість. Усі дані обробляються на серверах Google, що може бути проблемою для політики конфіденційності. Вартість може зростати при великій кількості запитів.

3. Microsoft Bot Framework (MBF) - це комплексний набір інструментів та сервісів для створення ботів. Він є більш "фреймворком", ніж "платформою", і часто використовується разом з Azure Cognitive Services (наприклад, LUIS для NLU).

Переваги:

- гнучкість та потужність. MBF (особливо з "Adaptive Dialogs") створений для розробки складних, керованих подіями діалогів, що дає значно більше контролю, ніж Dialogflow, наближаючись до гнучкості Rasa, але з підтримкою Microsoft;
- Bot Framework Composer/ Візуальний редактор, який дозволяє нетехнічним спеціалістам (наприклад, педагогам-дизайнерам) проектувати потоки діалогів, які потім допрацьовують розробники;
- інтеграція з екосистемою Azure. Легко підключається до баз знань (QnA Maker), мовних сервісів (LUIS) та інших інструментів Azure.

Недоліки:

- Складність, адже це рішення, орієнтоване на розробників (pro-code). Хоча Composer спрощує роботу, загальна екосистема є складною;
- прив'язка до Azure. Найбільшу користь фреймворк приносить при використанні в екосистемі Azure, що може бути надлишковим для невеликого проєкту.

Таблиця 1.4

Порівняльна таблиця для педагогічних агентів

Критерій	**Rasa**	**Google Dialogflow**	**Microsoft Bot Framework**
Основний підхід	Open-source, On-premise	Хмарна SaaS, Low-code	Фреймворк, Pro-code
Складність діалогу	Висока. Ідеально для складних сценаріїв.	Низька/Середня. Добре для Q&A та збору даних.	Висока. Підтримує адаптивні діалоги.
Контроль та кастомізація	Повний.	Обмежений. "Чорна скринька".	Високий.
Поріг входження	Високий (вимагає Python, ML)	Дуже низький.	Середній/Високий (вимагає C#/JS, Azure)
Конфіденційність даних	Повна (On-premise).	Низька (обробка в хмарі Google).	Середня (залежить від налаштувань Azure).
Найкраще для...	...дослідницьких проєктів, тьюторів із Сократівським методом, проєктів, де потрібен повний контроль.	...швидкого прототипування, простих тьюторів-асистентів, ботів для FAQ по курсу.	...корпоративних навчальних систем, інтеграції з MS Teams, складних тьюторів з підтримкою компанії.

Сучасні LLM API (Google Gemini та OpenAI API) кардинально змінили розробку AI-агентів, перетворившись із "інструментів" на готовий "мозок" для агента. Вони дозволяють перейти від програмування жорстких правил до простого інструктування природною мовою (prompt engineering).

Таблиця 1.5

Огляд можливостей LLM порівняно з Dialogflow

Характеристика	**Традиційні платформи (напр., Dialogflow)**	**Сучасні LLM API (Gemini, OpenAI)**
Основний підхід	Керування станом (State Machine). Потрібно вручну створювати "інтенти" (наміри), "ентіті" (сутності) та жорсткі дерева діалогу.	Керування інструкціями (Instruction-driven). Агент розуміє мету з опису (prompt) і може сам планувати кроки.
Розуміння мови	Обмежене NLU. Добре розпізнає навчені фрази та їх варіації для конкретних інтентів.	Глибоке розуміння (Zero/Few-Shot). Розуміє складні, нюансовані запити без попереднього навчання на тисячах прикладів.
Гнучкість діалогу	Низька. Користувач має слідувати заздалегідь визначеному сценарію. Важко обробляти відхилення від теми.	Висока. Агент може вести гнучку, людиноподібну розмову, легко перемикатися між контекстами та імпровізувати.
Генерація відповіді	Вибір відповіді. Обирає одну з заздалегідь написаних відповідей, прив'язаних до інтенту.	Генерація відповіді. Створює нові, унікальні, контекстуально доречні відповіді в реальному часі.
Швидкість розробки	Повільна. Вимагає ретельного проектування діалогових потоків та збору великого датасету для навчання NLU.	Дуже швидка. Прототип можна створити за хвилини, просто написавши хороший "системний промпт" (інструкцію).

Головна перевага LLM API в тому, що вони надають агенту здатність до міркування (reasoning). Замість того, щоб просто слідувати гілці if-then, агент може аналізувати запит, розбивати його на підзадачі, викликати зовнішні інструменти (наприклад, пошук або API) і формулювати комплексну відповідь.

Для педагогічних агентів це означає миттєвий перехід від простих Q&A ботів (Dialogflow) до повноцінних тьюторів, здатних вести Сократівський діалог, пояснювати складні концепції та адаптуватися до стилю навчання учня "на льоту".

Огляд сучасних фреймворків для роботи з LLM в екосистемі Python показує, що LangChain та LlamaIndex стали базовими інструментами для впровадження Retrieval-Augmented Generation (RAG) у освітніх і бізнес-програмах. LangChain забезпечує зручний спосіб організації ланцюжків запитів, управління промптами й повнофункціональне підключення LLM до зовнішніх джерел даних. LlamaIndex фокусується на індексації, зберіганні і ефективному пошуку навчальних матеріалів завдяки інтеграції з різними сховищами та векторними базами. У RAG-архітектурі LangChain зазвичай orchestrates роботу агентів, тоді як LlamaIndex надає функції індексації і швидкого пошуку, що дозволяє підключати власні навчальні дані до LLM та отримувати контекстно-залежні відповіді [27; 28].

Таблиця 1.6

Порівняльний аналіз ці підходів до створення педагогічного агента

Критерій	**Rasa (Open Source)**	**LangChain (з LLM API)**	**"Чистий" Python + Gemini API**
Гнучкість	Висока. Повна кастомізація NLU, логіки та компонентів. Але обмежена можливостями "класичного" NLU.	Дуже висока. Архітектура "Lego" для LLM. Дозволяє легко комбінувати моделі, пам'ять, інструменти (Tools) та джерела даних.	Середня. Гнучкість у тому, як ви будуєте обв'язку, але вся "магія" (міркування, NLU) замкнена всередині одного LLM API.
Складність розробки	Висока. Потрібне навчання NLU-моделей, написання "історій" (stories) та "правил" (rules). Високий поріг входження.	Середня. Абстрагує багато складних речей (пам'ять, парсинг), але вимагає розуміння нової парадигми (Chains, Agents, Prompts).	Низька (для старту). Легко почати з простого API-виклику. Висока (для масштабування). Всю логіку (пам'ять, керування діалогом, виклик інструментів) треба писати з нуля.
Контроль над логікою діалогу	Тотальний. Ви явно прописуєте кожну можливу гілку діалогу.	Середній. Логіка є "емерджентною" (emergent) – вона випливає з інструкцій (промпту) для	Низький. Ви повністю покладаєтесь на здатність LLM слідувати

Критерій	**Rasa (Open Source)**	**LangChain (з LLM API)**	**"Чистий" Python + Gemini API**
	Дуже детермінований підхід.	LLM. Менш передбачувана, але більш гнучка.	інструкціям у вашому системному промпті.
Вартість	Низька. Open-source. Витрати лише на власну інфраструктуру (сервери) для хостингу та навчання моделей.	API-залежна. Сам фреймворк безкоштовний, але основні витрати – це вартість API-викликів до LLM (Gemini, OpenAI тощо).	API-залежна. Усі витрати – це вартість API-викликів. Потенційно дешевше за LangChain, якщо ви оптимізуєте запити вручну.
Інтеграція з Python веб-системою	Висока. Rasa – це Python-фреймворк. Його можна запустити як окремий сервіс (через API) або інтегрувати частини.	Дуже висока. Це Python-бібліотека, створена для імпорту та використання безпосередньо у вашому Python-коді.	Дуже висока. Це просто набір Python-скриптів та викликів API за допомогою офіційної клієнтської бібліотеки (google-generativeai).

LangChain ймовірно є найбільш дотичним до сучасного педагогічного агента. Він дає потужність LLM (як у "чистому" Python), але водночас надає структуру та інструменти (як у Rasa) для керування пам'яттю, підключення до баз даних (наприклад, з навчальними матеріалами) та створення ланцюжків міркувань.

Висновки до розділу І

В першому розділі проведено комплексний аналітичний огляд теоретичних та практичних аспектів, що лежать в основі проектування та імплементації педагогічних AI-агентів. Цей аналіз заклав теоретичний фундамент для подальшої розробки програмного продукту.

Визначено ключову тернологію. Проведено розмежування понять «педагогічний AI-агент», «інтелектуальна тьюторська система» (ITS) та «освітній чат-бот». Встановлено, що ключовими відмінностями повноцінного педагогічного агента є висока автономність, цілеорієнтованість та здатність до проактивного,

багатокрокового планування для досягнення освітніх цілей, на відміну від більш реактивних чат-ботів чи структурованих ITS.

Проведено систематизацію та класифікацію. Педагогічні агенти були класифіковані за трьома основними критеріями:

1. За педагогічною роллю - тьютор/експерт, ментор, асистент та партнер по навчанню.

2. За способом втілення - текстові агенти, 2D- та 3D-аватари (Embodied agents) .

3. За технологічною архітектурою - агенти на основі правил, машинного навчання та сучасні агенти на базі великих мовних моделей (LLM) .

Проаналізовано сучасні підходи до архітектури. Досліджено еволюцію від класичної чотирикомпонентної архітектури ITS (Модулі знань, студента, педагогічний та інтерфейсу) до гнучких підходів на базі LLM. Визначено, що сучасні агенти активно використовують гібридні моделі представлення знань (поєднуючи онтології з векторними базами даних для RAG) , адаптивне моделювання студента та педагогічні стратегії, як-от Сократівський діалог і scaffolding. Водночас ідентифіковано ключові виклики, зокрема "галюцинації" LLM, складність оцінки та етичні аспекти.

Досліджено психолого-педагогічні основи. Визначено, що для ефективної взаємодії «людина-AI» критично важливими є психологічні чинники. Серед них ключовими є довіра до системи , управління когнітивним навантаженням (CLT) , сприйняття емпатії агента та підтримка внутрішньої мотивації учня згідно з теорією самодетермінації (SDT).

Проведено огляд ринку та інструментарію. Аналіз існуючих систем (Duolingo, Khanmigo, MATHia, Roadmap.sh) продемонстрував різні успішні моделі впровадження AI-тьюторингу. Порівняння традиційних фреймворків (Rasa, Dialogflow) із сучасними LLM API (Gemini, OpenAI) та бібліотеками (LangChain) показало перевагу останніх. LLM API надають агентам здатність до міркування (reasoning) та гнучкої генерації відповідей, а фреймворки типу LangChain

забезпечують необхідну структуру для інтеграції з базами даних та легкого підключення до існуючих веб-систем на Python.

Таким чином, проведений аналіз дозволив обґрунтувати вибір технологічного стеку для проєктного розділу, зупинившись на архітектурі, що використовує LLM API та Python-фреймворки (зокрема LangChain) як найбільш гнучкий, потужний та адекватний для реалізації поставлених у дипломній роботі завдань.

РОЗДІЛ 2 ПРОЄКТНИЙ РОЗДІЛ

2.1. Концепція та архітектура педагогічного AI агента

Метою розробленого педагогічного AI агента є підвищення ефективності навчання за рахунок автоматизованого, своєчасного та персоналізованого зворотного зв’язку. AI агент перетворює помилки студента на навчальні ситуації, генеруючи структуровані пояснення з урахуванням контексту конкретного питання, вибраної неправильної відповіді та матеріалу відповідного уроку. Таким чином студент отримує не просто правильний варіант, а зрозуміле обґрунтування «чому невірно» і «як мислити правильно», що сприяє формуванню стійких когнітивних стратегій.

Призначення агента також полягає в аналітиці результатів навчання та виявленні індивідуальних прогалин. На основі історії спроб і частоти помилок агент визначає проблемні теми, формує рекомендації для повторення та узагальнений AI-аналіз, доступний у веб-інтерфейсі та в PDF-звітах. Для викладача агент надає агреговані зрізи по курсах і студентських групах, а також можливість редагувати AI-пояснення з метою забезпечення методичної коректності та єдності термінології.

У підсумку агент виконує роль інтелектуального посередника між контентом курсу, діями студента та педагогічними цілями викладача. Він скорочує час на рутинний фідбек, підвищує прозорість навчального прогресу та підтримує адаптивність освітнього процесу, адже студент отримує адресні підказки й приклади, а викладач - інструменти контролю якості пояснень і підстави для коригування навчальних матеріалів.

Робота AI агента представлена як послідовний конвеєр:

1. Спочатку виконується аналіз помилок студента. З останніх спроб тестування вилучаються неправильні відповіді, фіксується контекст (текст питання, обрана відповідь, правильний варіант, фрагмент лекційного матеріалу) та

агрегуються повторювані теми, що дає змогу точно локалізувати прогалини та визначити пріоритети для подальшого пояснення й повторення.

2. Далі агент генерує структуроване пояснення для кожної помилки за єдиним шаблоном:

повне пояснення -

чому невірно -

як треба було мислити -

релевантний фрагмент лекції -

додатковий приклад.

Така уніфікована форма забезпечує педагогічну якість фідбеку, зменшує когнітивне навантаження та робить підказки придатними для повторного використання як у веб-інтерфейсі так і в рекомендаціях та PDF-звітах. Для підвищення оперативності застосовується кешування результатів генерації.

Згенеровані AI агентом пояснення зберігаються в базі даних і стають частиною індивідуальної навчальної траєкторії студента. Викладач має можливість переглядати ці пояснення, вносити редакторські правки (уточнювати формулювання, терміни, приклади), після чого оновлена версія стає еталонною для всіх студентів за відповідним питанням/темою. Такий цикл “генерація - модерація - повторне використання” поєднує швидкість AI із методичною відповідальністю викладача.

Модульна архітектура побудована за принципом розділення відповідальності згідно з підходом MVC у Django. Рівень даних представлений моделями (User, AIExplanation, навчальні сутності курсу/уроку/тесту), які інкапсулюють структуру та зв’язки, а також гарантують унікальність і відстеження змін. Рівень контролерів (views) реалізує сценарії використання: генерація пояснення, додаткового прикладу, експорт PDF, перегляд/агрегація помилок, редагування пояснень викладачем. Рівень представлення (templates + JS) відповідає за інтерфейс, асинхронні виклики та доступну подачу структурованого зворотного зв’язку. Спрощений варіант архітектурної схеми педагогічного

AI-агента наведено на рисунку 2.1 («Модульна архітектура педагогічного AI-агента»), де відображено ключові компоненти та їхню взаємодію на рівні UI - Views - AI-сервіс - Дані. Деталізована схема з повним переліком ендпойнтів, потоків даних, кешування та зовнішніх інтеграцій подана в Додатку А.

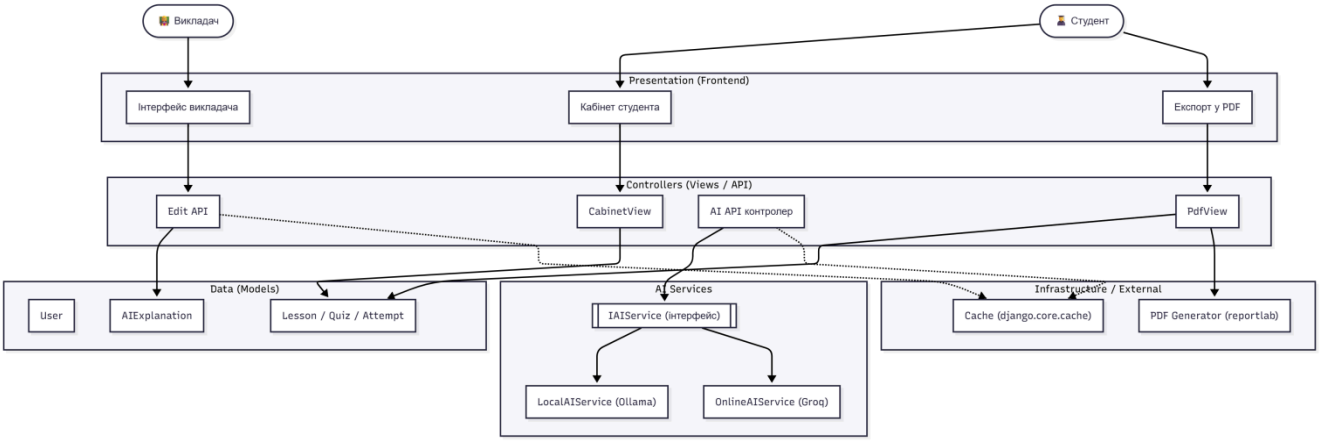


Рисунок 2.1 - Модульна архітектура педагогічного AI-агента

AI-функціональність винесена в окремий сервісний шар з чітким контрактом.

LocalAIService (Ollama) та OnlineAIService (Groq) реалізують однакові методи, що дає можливість прозоро перемикатися між провайдерами без змін у контролерах.

У сервісах зосереджені побудова промптів, валідація/парсинг відповідей, кешування результатів і fallback-логіка - це забезпечує повторне використання й тестованість, а також дотримується принципу відкритості/закритості для розширення (можна додати новий AI-провайдер як ще один адаптер).

Комунікація між шарами здійснюється через чіткі інтерфейси:

- API endpoints у контролерах приймають/повертають JSON, застосовують рольову авторизацію (студент/викладач) і обробку помилок;
- фронтенд працює асинхронно (fetch), не блокує інтерфейс і оновлює DOM без перезавантаження.

Такий поділ забезпечує масштабованість (горизонтальне розділення навантаження, заміна AI-провайдера), підтримуваність (локалізація змін у межах

шару), продуктивність (кешування AI-відповідей) і керованість якістю контенту (редагування викладачем як окремий контрольний етап).

Модульна архітектура з розділенням відповідальності, де кожен компонент виконує чітко визначену роль у конвеєрі “аналіз - генерація - збереження - відображення/редагування” спрощує підтримку, тестування та масштабування системи. Опишемо компоненти системи в таблиці 2.1

Таблиця 2.1

Компоненти системи та їх функції

№	Компонент	Призначення
1	AI-сервіс (LocalAIService / OnlineAIService)	Генерація педагогічного контенту
2	Модель даних AIExplanation	Збереження структурованих пояснень
3	API endpoints	Інтеграція з фронтендом (AJAX)
4	Інтерфейс користувача	Відображення та редагування пояснень

Компоненти 1–3 формують серверну логіку - AI-сервіси інкапсулюють роботу з провайдерами (локальний/онлайн), моделі забезпечують сталість і структурованість даних, а API надає єдиний контракт для інтерфейсу.

Компонент 4 реалізує сценарії взаємодії користувача: асинхронні запити до API, відображення структурованих блоків пояснень і режим модерації для викладача. Розділення на ці чотири рівні мінімізує зв’язність та спрощує заміну AI-провайдера або еволюцію інтерфейсу без змін у моделі даних.

Також система підтримує два режими роботи з мовною моделлю - локальний та онлайн (через зовнішній хмарний AI-провайдер).

Локальний режим забезпечує автономність і контроль даних; онлайн-режим - масштабованість і стабільний доступ до обчислювальних ресурсів. Уніфіковані механізми кешування, обробки помилок і fallback-поведінки однакові для обох режимів.

Кешування реалізовано на рівні AI-сервісу як прозорий шар між контролерами та провайдером мовної моделі. Відповіді AI на типові запити (пояснення помилки, додаткові приклади, короткий аналітичний підсумок для PDF) тимчасово зберігаються в застосунковому кеші за детермінованим ключем, сформованим із контексту задачі (питання, відповідь студента, правильна відповідь, версія промпту/моделі), що зменшує затримки та навантаження на AI-провайдер, стабілізує роботу при пікових зверненнях і знижує витрати. Час життя запису конфігурується; у випадку редагування пояснення викладачем кеш оновлюється/анулюється, щоб інтерфейс завжди відображав актуальну, модераторську версію контенту.

У системі застосовано рольову модель доступу, яка розмежовує педагогічні сценарії взаємодії з AI-агентом. Студент ініціює генерацію пояснень для власних помилок, переглядає структурований фідбек і додаткові приклади, тоді як викладач виконує контроль якості: переглядає зведену аналітику по студентських помилках і редагує AI-пояснення, забезпечуючи методичну коректність формулювань. Такий поділ мінімізує ризики некоректного контенту і водночас зберігає швидкість надання персоналізованого зворотного зв’язку.

Таблиця 2.2

Ролі користувачів і повноваження

Роль	**Основні дії**	**Інтерфейс/екрани**	**Обмеження та безпека**	**Створювані/оновлювані дані**
Студент	Запит на генерацію AI-пояснення для власних помилок; - Перегляд структурованого пояснення; - Отримання додаткового прикладу; - Експорт PDF-звіту	Кабінет студента: блок “AI-аналіз та рекомендації”, детальний аналіз питань	Доступ лише до власних спроб і пояснень; API генерації доступне тільки студентам	Записи AIExplanation (створення/оновлення для власних питань), кеш AI-відповідей

Роль	Основні дії	Інтерфейс/екрани	Обмеження та безпека	Створювані/оновлювані дані
Викладач	Перегляд зведень по темах і помилках студентів; - Перегляд збережених AI-пояснень; - Редагування пояснень (модерація контенту)	Кабінет викладача: топ-проблемні теми, детальні питання з поясненнями, режим редагування	API генерації для викладача вимкнене; редагування фіксує авторство (позначка не-AI) та застосовується до всіх студентів	Оновлення AIExplanation (встановлення відредагованих текстів, is_ai = False), актуалізація кешу
Викладач	Перегляд зведень по темах і помилках студентів; - Перегляд збережених AI-пояснень; - Редагування пояснень (модерація контенту)	Кабінет викладача: топ-проблемні теми, детальні питання з поясненнями, режим редагування	API генерації для викладача вимкнене; редагування фіксує авторство (позначка не-AI) та застосовується до всіх студентів	Оновлення AIExplanation (встановлення відредагованих текстів, is_ai = False), актуалізація кешу

Редагування викладачем має пріоритет над AI-версією та синхронізується для всіх студентів по відповідному питанню/темі.

Доступ до генерації пояснень обмежено студентами; викладач працює з уже збереженим контентом та його модерацією.

Експорт PDF доступний обом ролям; у студентів містить персональний аналіз, у викладачів - агреговані висновки за курсами/групами.

2.2. Розроблення інформаційної (комп’ютерної) моделі агента

Інформаційна модель AI агента описує, як фіксуються помилки студентів, яким чином зберігаються згенеровані AI-пояснення та як ці дані пов’язуються з навчальними сутностями курсу. Базова ідея полягає в тому, що одна зафіксована помилка (конкретний студент, конкретне питання та конкретна обрана

неправильна відповідь) відповідає одному запису пояснення. Це дає змогу забезпечити відтворюваність, уніфікацію та аналітику на рівні тем/уроків/курсів.

Траєкторія даних вибудована за ланцюгом Course - Lesson - Quiz - Question - AIExplanation, а історія спроб фіксується в QuizAttempt із переліком неправильних запитань (M2M). Подальша аналітика виконується засобами агрегування (наприклад, collections.Counter) для визначення «вузьких місць» у засвоєнні матеріалу.

Таблиця 2.3

Сутність AIExplanation: поля та обмеження

Поле	Тип	Призначення	Обмеження/примітки
student	FK - User	Посилання на студента, для якого згенероване пояснення	Участь в унікальній комбінації
question	FK - Question	Питання, у якому допущено помилку	Участь в унікальній комбінації
student_answer	Char/Text	Обрана відповідь студента (контекст помилки)	Участь в унікальній комбінації
correct_answer	Char/Text	Правильна відповідь (контекст)	Участь в унікальній комбінації
full_explanation	Text	Повне пояснення	Основний текст для студента
why_wrong	Text	Чому відповідь невірна	Структурована секція
how_to_think	Text	Як треба було мислити	Структурована секція
lesson_fragment	Text	Релевантний фрагмент лекції	Контекст для повторення
additional_example	Text	Додатковий приклад	За запитом користувача
is_ai	Bool	Позначка AI-генерації (або відредаговано викладачем)	При модерації встановлюється False
created_at	DateTime	Час створення запису	Аудит
updated_at	DateTime	Час останнього оновлення	Аудит

Унікальність визначається комбінацією (student, question, student_answer, correct_answer), тобто одна помилка - це один запис пояснення. Це унеможливлює дублювання при повторних зверненнях і спрощує кешування.

Поле is_ai відокремлює сирий AI-контент від модераторської (педагогічно вивіреної) версії; після редагування викладачем значення встановлюється в False.

Таблиця 2.4

Зв’язки та кардинальності навчальних сутностей

Від	До	Тип/кардинальність	Призначення
Course	Lesson	1:N	Курс містить багато уроків
Lesson	Quiz	1:1 або 1:N	Уроці(-ах) можуть бути тести
Quiz	Question	1:N	Тест складається з питань
QuizAttempt	Question	M:N (через incorrect_questions)	Фіксація неправильних відповідей спроби
Question	AIExplanation	1:N (по студентам/відповідям)	На одне питання існують різні пояснення для різних студентів і контекстів
User (student)	AIExplanation	1:N	Один студент має багато пояснень

M:N між QuizAttempt та Question дає можливість зберігати лише неправильні питання конкретної спроби, що є підставою для подальшої генерації пояснень.

Зв’язок Question - AIExplanation відображає факт, що пояснення залежать не лише від питання, а й від обраної студентом відповіді (контекст помилки).

Таблиця 2.5

Агрегація та аналітика помилок

Об’єкт агрегації	Ключ групування	Метрика	Інструмент/механізм	Результат
Неправильні питання	Lesson / Topic	Кількість помилок	collections.Counter	Топ проблемних тем для повторення
Пояснення	Question / Student	Частота звернень/генерацій	Кеш + логи	Оптимізація генерації, економія часу
Результати тестів	Quiz / Course	Середній бал, % помилок	Вбудовані агрегати БД	Висновки в PDF та інтерфейсі

Агрегації використовуються для визначення пріоритетів повторення (теми з високою концентрацією помилок) та оцінювання динаміки навчання.

Результати підживлюють як інтерфейс (рекомендації, списки пріоритетів), так і PDF-звіти (короткий AI-аналіз, структурований висновок).

Рисунок ER (словесно): ланцюг Course - Lesson - Quiz - Question - AIExplanation задає навчальний контекст для кожного пояснення. QuizAttempt з M2M incorrect_questions пов’язує конкретну спробу студента з помилками, що виступають тригерами для генерації структурованих пояснень і подальшої аналітики.

Схема бази даних подана у вигляді ER-діаграми, що відображає ключові сутності навчальної платформи та зв’язки між ними:

- Course - Lesson - Quiz - Question,
- історію спроб (QuizAttempt) з M:N-зв’язком до неправильних запитань,
- також педагогічні пояснення (AIExplanation), які пов’язують конкретного студента з конкретним питанням і контекстом помилки.

Розроблена структура забезпечує відтворюваність даних, однозначну ідентифікацію помилок і зручність агрегування (за уроками, тестами, темами). Повна ER-діаграма зі зазначенням кардинальностей, ключів та обмежень наведена в Додатку Б.

Для забезпечення швидкого доступу до пояснень використано індекси, узгоджені зі «слідками запитів» у застосунку. Базовим є складений унікальний індекс на AIExplanation(student_id, question_id, student_answer, correct_answer), який одночасно гарантує унікальність «одна помилка = одне пояснення» і прискорює точкові пошуки за повним контекстом помилки. Для найчастіших сценаріїв читання додатково доцільні прості індекси на question_id (вибірка всіх пояснень по питанню, у т.ч. для PDF) та student_id (історія пояснень конкретного студента), а також індекс на created_at/updated_at для сортування й побудови звітів.

Оптимізація запитів поєднує індексацію з ORM-практиками: select_related/prefetch_related для мінімізації N+1 при завантаженні пов’язаних Question, Quiz, Lesson; пагінація та агрегації по «вузьких» ключах (наприклад,

підрахунки по lesson_id через приєднання до Question). На рівні AI-сервісу застосовується кешування з детермінованим ключем (хеш контексту запиту), що різко зменшує кількість повторних обчислень і навантаження на БД та AI-провайдера; при модерації (редагуванні) пояснення кеш інвалідовується, аби віддавати актуальну версію.

2.3. Вимоги до програмного забезпечення та користувацького інтерфейсу

Узгодженою практикою інженерії вимог, вимоги до педагогічного AI-агента згруповано у три категорії [28]:

- функціональні вимоги описують, що саме має робити система (генерація та редагування пояснень, додаткові приклади, аналітика, PDF-звіти, кешування результатів);
- вимоги до інтерфейсу визначають сценарії взаємодії користувачів (структура екранів, асинхронні виклики, індикатори стану, режим модерації для викладача, доступність та зрозумілість подачі контенту);
- технічні вимоги встановлюють обмеження та умови експлуатації (підтримка локального й хмарного AI-режимів, продуктивність і кешування, захист даних і авторизація за ролями, логування та масштабованість).

Розглянемо очікувану поведінку педагогічного AI-агента з позиції користувачів (студент, викладач) та інтеграції з навчальною платформою. Вимоги подано у вигляді функціональних можливостей, що утворюють наскрізний сценарій “виявлення помилок - генерація пояснень - аналітика - модерація/редагування - повторне використання”.

Під функціональними вимогами розуміємо опис “що має робити система” з точки зору зовнішньо спостережуваних результатів (артефактів, станів інтерфейсу, API-відповідей), без деталізації внутрішньої реалізації. Такі вимоги є основою для

приймальних випробувань і трасуються на конкретні екрани, ендпойнти та моделі даних (табл. 2.6).

Таблиця 2.6

Функціональні вимоги до педагогічного AI-агента

№	Вимога	Опис	Роль/актор	Вхід/тригер	Результат/артефакт	Примітки
1	Генерація AI-пояснень	Створення структурованого пояснення: “чому невірно”, “як мислити”, “фрагмент лекції”	Студент	Клік/запит з помилкового питання	Пояснення, збережене у AIExplanation і видиме в UI	Кешування відповіді за контекстним ключем
2	Додатковий приклад	Генерація короткого прикладу для закріплення теми	Студент	Клік “Показати додатковий приклад”	Поле additional_example оновлено та відображено	Повторні звернення обслуговуються з кешу
3	Аналіз та топ-тем	Агрегація помилок і формування списку пріоритетних тем	Обидві ролі	Наявність історії спроб	Перелік тем з метриками (% помилок, спроби)	Агрегація через Counter/SQL-агрегати
4	PDF-звіт	Експорт звіту з AI-аналізом і витягами пояснень	Студент, викладач	Клік “Експортувати PDF”	PDF із загальними метриками та AI-висновком	Стиль: формальний, стисле резюме
5	Редагування пояснень	Редакторська правка текстів пояснень	Викладач	Клік “Редагувати/Зберегти”	Оновлення записів AIExplanation, is_ai=false	Поширюється на всі екземпляри питання
6	Кешування AI	Тимчасове збереження AI-відповідей для швидкого доступу	Система	Повторний ідентичний запит	Миттєва відповідь без виклику моделі	Інвалідація після модерації

Кожна вимога має чітку трасованість до артефактів системи:

- екранів (кабінет студента/викладача),
- ендпойнтів (генерація пояснення/прикладу, редагування),
- моделей (AIExplanation, спроби тестів).

Вимога щодо кешування є перехресною - вона не змінює бізнес-логіку, але гарантує прийнятний час відгуку в типових сценаріях та зменшує навантаження

на AI-провайдера. Приймальні критерії формулюються через очікувані стани інтерфейсу (поява/оновлення пояснення, правильне наповнення PDF, застосування правок викладача) та консистентність даних у базі (унікальність пояснень для однакових контекстів помилок).

Інтерфейс має підтримувати швидкий доступ до ключових дій, наочну подачу AI-пояснень і прозорі стани системи під час асинхронних операцій. Нижче наведено вимоги, що визначають зручність, доступність та керованість UI у сценаріях студента й викладача.

Таблиця 2.7

Вимоги до користувацького інтерфейсу

№	Вимога	Опис	Критерії приймання
1	Інтуїтивність	Прямі дії для користувача: кнопки “Отримати AI-пораду”, “Показати додатковий приклад” у контексті питання	Кнопки видимі поряд із питанням; підписані зрозуміло; доступ з клавіатури
2	Візуальна структурованість	Пояснення розділені на секції з іконками/колористикою: “чому невірно”, “як мислити”, “фрагмент лекції”, “приклад”	Наявні заголовки секцій, піктограми та послідовність блоків; однаковий шаблон по всьому UI
3	Режими перегляду/редагування	Для викладача - перемикання “Перегляд ↔ Редагування” пояснень	Кнопка “✏️ Редагувати/👁️ Переглянути”; поля редагування збережені й відображені без перезавантаження
4	Індикатори завантаження	Відображення процесу генерації/збереження (спінери, прогрес-бари, стан кнопок)	Під час запиту показано спінер; кнопки блокуються; після завершення індикатори зникають
5	Адаптивність	Коректне відображення на різних пристроях і розмірах екрана	Відсутні горизонтальні скролли на мобільних; блоки перелаштовуються; таблиці/акордеони читабельні
6	Повідомлення про стани	Комунікація про успіх/помилку (alert-повідомлення, inline-підсвітка)	Після успіху - “Пояснення оновлено”; при помилці - змістовний текст і рекомендація повтору дії

Після впровадження наведених вимог інтерфейс забезпечує передбачувану поведінку для студентів (швидкий запит і читання пояснень) та викладачів (зручна модерація контенту), з мінімальним когнітивним навантаженням і чітким інформуванням про стани системи.

Технічні вимоги визначають обов’язкові інженерні характеристики системи, що забезпечують стабільність, безпеку та керовану експлуатацію. В табдиці 2.8 наведено вимоги до інтеграції AI, обробки помилок, взаємодії фронтенду з бекендом і безпеки.

Таблиця 2.8

Технічні вимоги

№	Вимога	Опис	Реалізація/механізм	Критерії приймання
1	Підтримка двох AI-провайдерів	Локальний/онлайн режим із уніфікованим інтерфейсом	Абстракція IAIService, адаптери Local/Online, конфіг через env	Перемикання режиму без змін у UI/Views; стабільна робота обох
2	Обробка помилок і fallback	Стійкість до timeouts/недоступності провайдера	Try/catch у сервісах, читабельні повідомлення, резервні тексти	Користувач завжди отримує пояснення/повідомлення без «падіння» UI
3	Асинхронність (AJAX)	Взаємодія без перезавантаження сторінки	fetch для API; індикатори стану; оновлення DOM	UI не перезавантажується; видно прогрес/результат/помилки
4	Безпека доступу	Рольова авторизація та захист форм	Перевірка ролей у Views; CSRF-токен у запитах	Студент не може редагувати; викладач не викликає генерацію; CSRF ок

Таблиця 2.9

Вимоги до продуктивності

№	Вимога	Опис	Реалізація/механізм	Критерії приймання
1	Кешування AI-відповідей	Зменшення латентності та навантаження	Кеш ключем контексту (питання/відповідь/модель) на 24 год	Повторний ідентичний запит відповідає з кешу; інвалідація після модерації

№	Вимога	Опис	Реалізація/механізм	Критерії приймання
2	Оптиміза ція запитів до БД	Уникнення N+1, швидке чи тання	select_related, prefetch_related, індекси на question_id/student_id	Час завантаження списків стабі льний при зростанні даних
3	Обмежен ня тексту в PDF	Керована д овжина AI-резюме	Триммінг до ≈300 символів, розбиття рядків	PDF має стисле резюме без обрізаних слів/артефактів верстки

Зазначені вимоги забезпечують взаємозамінність AI-провайдерів, передбачувану поведінку фронтенду, контроль доступу та стабільний час відгуку завдяки кешуванню і оптимізованим запитам до БД.

2.4. Реалізація програмного забезпечення педагогічного AI агента

Для реалізації педагогічного AI агента обрано стек, що поєднує серверну платформу (Django) з провайдер-агностичним AI-шаром і легковаговим фронтендом на шаблонах та JS. Такий підхід дає швидке серверне рендеринг-UX, надійну роботу з даними (ORM, індекси), контроль доступу за ролями та прозоре перемикання між локальним і хмарним режимами AI-моделі (табл. 10).

Таблиця 2.10

Обраний стек технологій

Шар / компонент	Технології	Призначення	Ключові особливості / реалізація
Backend	Python, Django (MVC), Django ORM	Бекенд-логіка, моделі, маршрути і контролери	Views для API/сторінок, валідний кеш, рольова авторизація; логування та обробка помилок
AI-шар	Інтерфейс IAIService; LocalAIService (Ollama); OnlineAIService (зовнішній AI-провайдер)	Генерація пояснень, прикладів, PDF-резюме	Єдиний контракт; побудова промптів; валідація/парсинг відповіді; кешування; fallback; конфіг через env

Шар / компонент	Технології	Призначення	Ключові особливості / реалізація
Frontend	Django Templates, JS (fetch), CSS	Інтерфейс користувача, асинхронні операції	AJAX без перезавантаження; режим редагування для викладача; індикатори станів; адаптивні стилі
Дані / БД	SQLite/PostgreSQL, індекси	Зберігання AIExplanation, навчальних сутностей, спроб	Унікальний складений індекс (student, question, student_answer, correct_answer), індекси на question_id, student_id; select_related/prefetch_related
Звіти	Reportlab	Формування PDF з AI-аналізом	Стисле резюме (≈300 символів), перенесення рядків, стабільна верстка
Кеш	Django cache	Прискорення відповідей і зниження навантаження	Ключ за контекстом запиту; TTL ≈ 24 год; інвалідація після модерації
Безпека	CSRF, рольова модель доступу	Захист POST/редагування, розмежування прав	Студент - генерація/перегляд; викладач - перегляд/редагування; читабельні повідомлення про помилки

Стек орієнтовано на надійність і керованість, де Django забезпечує чисту предметну модель і контроль доступу, AI-шар - взаємозамінність провайдерів, а кешування та індексація - стабільну продуктивність навіть за зростання обсягу пояснень та запитів. Структура проєкту наведена в додатку В.

Проєкт організовано як набір Django-додатків за доменами:

- courses (навчальний контент),
- quizzes (оцінювання),
- users (користувачі й AI-логіка)
- каталог templates (інтерфейс).

Призначення courses - управління навчальними сутностями та їх ієрархією.

Ключові моделі:

- Course (власник - викладач),
- Lesson (належить курсу).

Відповідальність:

- створення/редагування курсів і уроків,
- зв’язок з тестами через уроки;
- відображення змісту уроків.

Взаємодія Lesson → Quiz (із quizzes), дані уроків цитуються в AI-поясненнях (фрагменти лекцій).

Призначення quizzes - тестування знань і фіксація результатів.

Ключові моделі:

- Quiz (належить уроку),
- Question,
- Answer,
- QuizAttempt (+ M2M incorrect_questions).

Відповідальність:

- проходження тестів,
- збереження спроб,
- підрахунок балів,
- формування переліку помилкових запитань.

Взаємодія - помилки зі спроб (QuizAttempt.incorrect_questions) - вхід для AI-аналізу та рекомендацій у users.

Призначення users - користувачі, ролі та весь педагогічний AI-шар.

Ключові моделі:

- User (ролі: студент/викладач),
- AIExplanation (структуровані пояснення з контекстом помилки).

AI-сервіси:

- ai_service.py (локальний, Ollama),
- ai_service_online.py (онлайн-провайдер);
- єдиний контракт генерації пояснень, додаткових прикладів і PDF-резюме, кешування та fallback.

Відповідальність - кабінет користувача з аналітикою, API для генерації/редагування пояснень, експорт PDF з AI-аналізом, рольовий контроль доступу.

Призначення templates - інтерфейс користувача (серверний рендеринг + JS).

Структура:

- templates/users/cabinet.html (AI-аналіз, рекомендації, режими перегляду/редагування для викладача),
- templates/courses/*.html (курси/уроки),
- templates/quizzes/*.html (редагування тестів, проходження),
- спільний base.html.

Особливості UI:

- інтуїтивні дії (“Отримати AI-пораду”, “Показати додатковий приклад”),
- секції пояснень з іконками/кольорами,
- спінери/прогрес-бар,
- повідомлення про успіх/помилки, адаптивність.

Зв’язки між додатками:

- Course → Lesson → Quiz → Question (із courses до quizzes),
- далі QuizAttempt формує помилкові питання для AIExplanation (у users),
- templates відображають ці дані та надають інструменти генерації/редагування пояснень.

Роутінг у системі організовано за доменним принципом із простими префіксами для кожного додатку:

- кореневі маршрути та API зосереджені в users,
- навчальний контент - у courses,
- оцінювання - у quizzes.

На рівні config увімкнено admin, авторизацію (login/logout) і підключення просторів імен кожного модуля, що спрощує навігацію й контроль доступу за ролями (студент/викладач), а також відокремлює API-ендпойнти від сторінок.

Узагальнюючи, маршрути структуровано за призначенням і ролями - сторінки та API чітко розмежовані, критичні POST-ендпойнти захищено перевіркою прав і CSRF, а експорти (PDF/Excel) доступні як ідемпотентні GET-операції. Така організація спрощує підтримку, тестування та подальше масштабування функціональності. Таблицю маршрутизації наведено в додатку Г.

Для прикладу розглянемо схему роботи ендпоінта POST users/api/generate-ai-explanation/

POST /api/generate-ai-explanation/ - ендпоінт для студентів, що генерує структуроване AI-пояснення до помилки за question_id і відповіддю студента (AJAX, CSRF). Спочатку перевіряються права й вхідні дані; далі виконується пошук у кеші за контекстним ключем, а за його відсутності викликається AI-сервіс (локальний або онлайн). Результат зберігається/оновлюється в AIExplanation (унікальна комбінація student, question, student_answer, correct_answer, is_ai=True) і повертається як JSON. У разі помилки надсилається уніфікована відповідь success:false з відповідним HTTP-кодом (400/403/405/500).

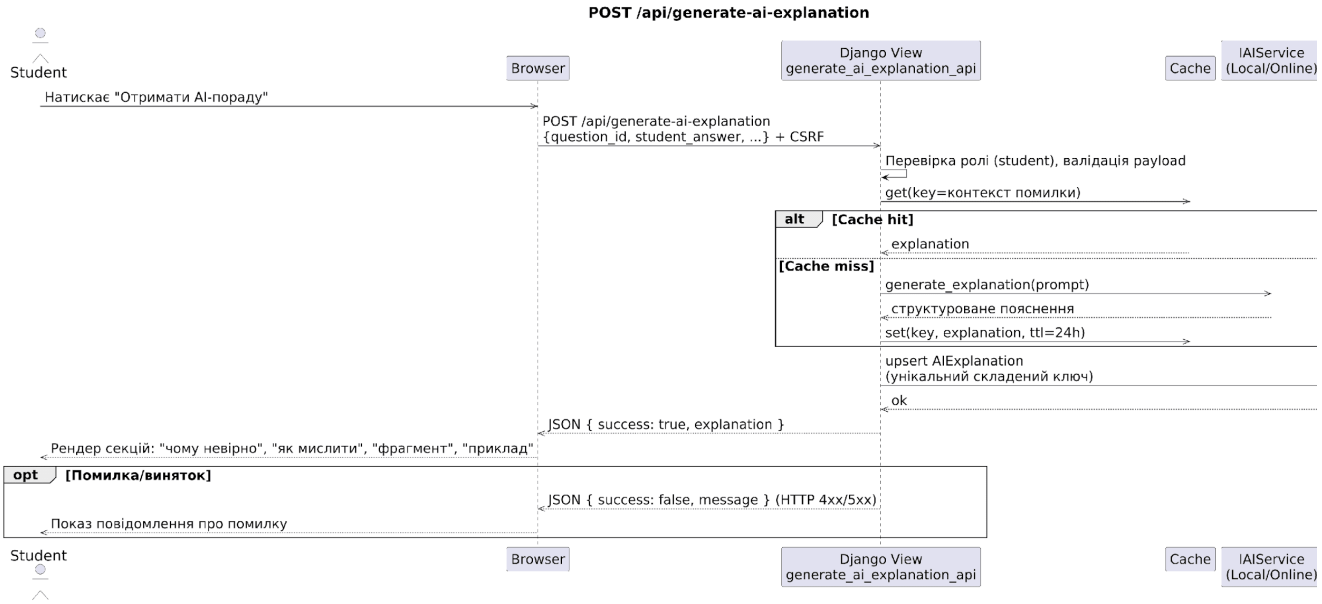


Рисунок 2.2 - Схема ендпоінта POST users/api/generate-ai-explanation/

За роботу AI агента відповідає шар контролерів:

1. *generate_ai_explanation_api(request)* - генерація пояснень

Метод/доступ: POST, лише студент; CSRF обов’язковий.

Вхід: question_id, відповідь студента (+контекст).

Логіка: валідація → кеш-перевірка → виклик AI-сервісу (локальний/онлайн) → збереження/оновлення AIExplanation з is_ai=True.

Вихід: JSON із структурованим поясненням; помилки - уніфікований JSON success:false (400/403/405/500).

2. *generate_additional_example_api(request)* - генерація додаткових прикладів

Метод/доступ: POST, студент.

Вхід: question_id (за потреби - фрагмент контексту).

Логіка: валідація → кеш → AI-сервіс → запис у поле additional_example відповідного AIExplanation.

Вихід: JSON із коротким прикладом; у разі помилок - success:false.

3. *edit_ai_explanation_api(request)* - редагування викладачем

Метод/доступ: POST, лише викладач.

Вхід: explanation_id або question_id; поля для оновлення: full_explanation, why_wrong, how_to_think, lesson_fragment, additional_example.

Логіка: пошук цільового пояснення → оновлення полів → is_ai=False → поширення правок на всі AIExplanation для того ж question → інвалідація кешу.

Вихід: JSON із оновленими полями.

4. *export_student_report_pdf(request)* - PDF з AI-аналізом

Метод/доступ: GET, студент/викладач (за контекстом).

Вхід: параметри ідентифікації користувача/діапазону даних (із сесії/URL).

Логіка: агрегація результатів (топ-помилки, статистика) → виклик AI-резюме з професійним тоном (із fallback) → складання PDF через reportlab (обмеження ~300 символів для резюме).

Вихід: файл PDF як HTTP-відповідь.

5. *cabinet_view(request)* - агрегація та рекомендації

Метод/доступ: GET, студент/викладач (умовний рендеринг).

Логіка (студент): збір спроб, детальні AI-пояснення по помилках, прогрес, рекомендації.

Логіка (викладач): агрегування помилок усіх студентів курсу, топ-тем, показ наявних AIExplanation по унікальних питаннях, можливість редагування.

Вихід: HTML-сторінка з секціями аналітики, рекомендацій та інструментами (генерація/редагування).

Тепер розглянемо алгоритми та логіку генерації AI пояснень помилок (рис. 2.3). Процес починається з отримання контексту помилки:

- ідентифікатора питання,
- варіантів відповіді,
- фактичної відповіді студента
- дотичного фрагмента лекційного матеріалу.

Далі виконується валідація обов’язкових полів і прав доступу. За коректних вхідних даних формується промпт, що стисло, але повно описує задачу для моделі: формати секцій “чому невірно”, “як мислити”, “фрагмент лекції”, “додатковий приклад”. Промпт надсилається до AI-сервісу (локальний або онлайн), відповідь очікується у структурованому JSON. Після отримання даних виконується парсинг і валідація структури/типів; у разі невідповідності застосовується обробка помилок (fallback-повідомлення, логування). Валідні результати зберігаються в БД (оновлення або створення запису пояснення), після чого користувачу повертається успішна відповідь зі структурованими блоками пояснення.

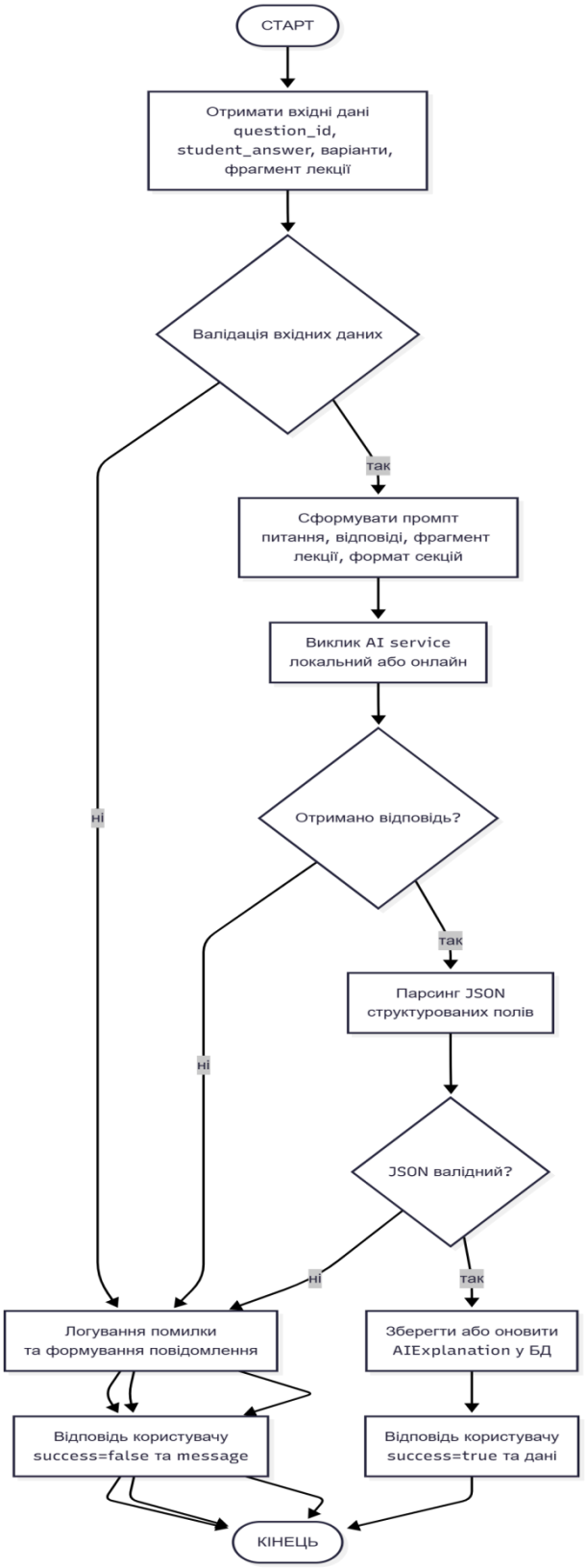


Рисунок 2.3 - Алгоритм генерації AI пояснень помилок

Система збирає всі спроби студента, витягає з них перелік неправильних запитань і проєктує кожну помилку на пов’язаний урок/тему. Далі виконується

підрахунок частоти помилок через collections.Counter (ключ - ідентифікатор уроку/теми), після чого формується топ-N проблемних тем за спаданням кількості. Для кожної теми/питання підтягуються наявні AIExplanation (щоб одразу показати готові поради); для відсутніх - пропонується генерація пояснень і/або додаткових прикладів. У викладача та сама логіка працює на агрегованих даних усіх студентів курсу, що дозволяє бачити «вузькі місця» навчання та перевіряти коректність AI-рекомендацій.

Перед викликом AI обчислюється детермінований ключ кешу як хеш від трійки: (питання, відповідь студента, правильна відповідь) і, за потреби, версії промпту/моделі. Якщо запис у кеші існує - система повертає відповідь миттєво; якщо ні - викликає AI, нормалізує результат і кладе в кеш із терміном зберігання 24 години. Кеш інвалідовується або версіонується у випадках модерації викладачем (щоб студент бачив оновлене пояснення) чи змін промптів/моделі.

Інтеграція з фронтендом реалізована наступним чином:

1. AJAX-запити. Асинхронні виклики через fetch до API генерації пояснень/прикладів і редагування, з переданням CSRF-токена в заголовках. Запити не блокують сторінку й дозволяють паралельні операції.

2. Оновлення DOM. Після успіху дані з відповіді мапляться у відповідні секції пояснення (чому невірно, як мислити, фрагмент, приклад), кнопки/спінери перемикаються, текст рендериться з перенесеннями, а редагування викладача миттєво відображається без перезавантаження.

3. Обробка помилок. Перевірка response.ok, парсинг success:false і відображення змістовних повідомлень (alert/inline), повторне ввімкнення кнопок, приховування індикаторів, логування у консоль для діагностики.

4. Клієнтська валідація. Перед надсиланням перевіряються обов’язкові поля (question_id, відповіді), наявність даних у глобальному масиві/data- атрибутах елемента питання; у разі відсутності - показ підказки користувачу й блокування запиту.

Комплексна обробка помилок у педагогічному AI-агенті побудована за принципом «ніколи не залишати користувача без відповіді». Якщо генерація не вдалася (таймаут, недоступність провайдера, помилка парсингу або валідності структури), система формує професійний fallback-текст. Якщо у БД вже існує попередня версія пояснення для цього самого контексту помилки, вона віддається замість порожньої відповіді. Так підтримується безперервність навчального досвіду навіть у разі тимчасових збоїв AI.

Логування виконується на бекенді з розрізненням рівнів: validation/permission - як попередження, винятки AI/мережі - як помилки з traceback. До запису додаються мінімально потрібні технічні атрибути (ідентифікатор користувача, question_id, режим AI - локальний/онлайн, час відповіді, стан кешу: hit/miss) без збереження чутливих даних відповіді користувача. Такий підхід полегшує діагностику повторюваних інцидентів, аналіз продуктивності та кореляцію з навантаженням. Для масових збоїв доцільні метрики:

- частка fallback-відповідей,
- відсоток помилок парсингу,
- середній latency,
- частота таймаутів.

Комунікація з користувачем є уніфікованою. На API-рівні усі помилки повертаються в єдиному форматі JSON: success=false, message, optional error; HTTP-коди: 400 (невалідні або відсутні поля), 403 (бракує прав), 405 (метод не дозволено), 500 (внутрішня помилка/AI). На фронтенді кожен запит перевіряє response.ok, парсить JSON і відображає змістовні повідомлення:

- «Не вдалося згенерувати пояснення. Спробуйте ще раз пізніше»
- «Недостатньо прав для цієї дії».

Кнопки на час операції блокуються, показуються спінери/прогрес-бари, а після завершення стан інтерфейсу коректно скидається. Для валідації на клієнті

перед відправкою перевіряються ключові поля (question_id, відповідь студента); якщо їх бракує, запит не відправляється і показується підказка.

Для підвищення надійності застосовуються додаткові механізми:

- кеш відповідей на 24 години із детермінованим ключем (хеш від комбінації «питання + відповідь студента + правильна відповідь», за потреби - версія промпту/моделі);
- опційні короткі повторні спроби з обмеженим бекофом для транзитних мережевих збоїв;
- інвалідація кешу після модерації викладачем, щоб студенти негайно бачили оновлені тексти.

Водночас у повідомленнях інтерфейсу не розкриваються внутрішні деталі промптів або трасування - це зменшує шум для користувача і захищає внутрішні механізми системи.

2.5. Розроблення педагогічних засобів і навчального контенту

AI-пояснення спроєктовано як стабільну педагогічну «рамку» формувального зворотного зв’язку - воно не лише повідомляє про помилку, а й навчає правильному способу мислення, спираючись на матеріали курсу. Структура уніфікована, щоб забезпечити порівнюваність пояснень, повторне використання у звітах і зручну модерацію викладачем:

- Повне пояснення (full_explanation) - короткий, предметно-точний опис суті помилки й потрібного знання для її виправлення; без емоційних оцінок, у професійному тоні.
- Чому невірно (why_wrong) - конкретизація хибного кроку або хибної уяви; бажано зіставити вибір студента з правильною відповіддю й вказати, який принцип було порушено.

- Як треба думати (how_to_think) - алгоритм/евристика розв’язання (кроки перевірки, порядок застосування правил, типові пастки); формулювання має бути узагальнюваним на подібні задачі.
- Фрагмент з лекції (lesson_fragment) - стисло процитований релевантний матеріал з курсу з посиланням на тему/урок (якір для повторення й самостійної перевірки).
- Додатковий приклад (additional_example) - невелике практичне завдання або ілюстрація з очікуваним результатом/підказкою; спрямоване на закріплення конкретного принципу з пояснення.

Така структура забезпечує повний цикл навчання - ідентифікацію помилки → усвідомлення причини → опанування правильної стратегії → повернення до джерела знань → практичне закріплення. Викладач за потреби редагує тексти, зберігаючи єдиний стиль і відповідність програмі курсу.

Розроблення AI-пояснень у проєкті спирається на такі педагогічні принципи, що забезпечують адресний, контекстний і підтримувальний зворотний зв’язок:

1. Персоналізація - пояснення формуються під конкретну помилку студента на основі пари question_id + student_answer. Це забезпечує адресність зворотного зв’язку - система пояснює не «типову» помилку, а саме ту, що сталася, та зберігає її у AIExplanation для подальшого супроводу.
2. Контекстуалізація - у відповіді обов’язково використовується релевантний фрагмент лекції (lesson_fragment), що прив’язує рекомендації до курикулуму. Студент отримує посилання на джерело знань і може швидко повторити потрібну тему.
3. Поступовість - структура пояснення рухається від загального до конкретного - спершу стислий опис помилки (full_explanation), потім точна причина (why_wrong), далі - стратегія мислення (how_to_think), і на завершення - приклад для закріплення (additional_example). Це підтримує поетапне формування компетентності.

4. Позитивний підхід: акцент робиться на правильному способі розв’язання і наступних кроках, а не на констатації хибності. Тон формулювань професійний і підтримувальний, без оцінних суджень, що підвищує мотивацію та знижує тривожність під час навчання.

Генерація здійснюється через структуровані промпти з чіткими інструкціями щодо вихідного формату і тону. Для інтерфейсу застосовується дружній, підтримувальний стиль, що мотивує до подальших кроків; для PDF - формальний і лаконічний, без звертань на «ти», з нейтральною лексикою. Обидва режими повертають строго валідний JSON для надійного парсингу у застосунку.

Зразок промпту (шаблон з параметрами):

Ти - педагогічний асистент. Поверни відповідь українською мовою СУВОРО у форматі JSON без жодних додаткових коментарів.Мета: згенерувати структуроване пояснення помилки студента для питання тесту.Вхідні дані:- question_id: {{question_id}}- question_text: "{{question_text}}"- options: {{options_json}} # масив варіантів, якщо є- student_answer: "{{student_answer}}"- correct_answer: "{{correct_answer}}"- lesson_fragment: "{{lesson_fragment}}" # коротка релевантна цитата з лекції (може бути порожньою)Вимоги до стилю/тону:- tone: {{tone}} # "formal" для PDF, "friendly" для UI- Якщо tone="formal": стиль професійний і стриманий, без звертання на "ти", використовується нейтральне "студент", жодних емоційних оцінок, лаконічність.- Якщо tone="friendly": доброзичливо, підтримувально, без фамільярності, конкретні кроки дій.Зміст і формат відповіді (JSON з рівнем ключів 1):{ "full_explanation": "...", # короткий предметний опис суті помилки і як її виправити "why_wrong": "...", # конкретна причина хибності обраної відповіді "how_to_think": "...", # алгоритм/евристика мислення для правильного розв’язання "lesson_fragment": "...", # релевантний уривок з лекції (за потреби перефразувати стисло) "additional_example": "..." # невеликий приклад для закріплення принципу}Правила:- Поверни ТІЛЬКИ валідний JSON (без преамбул, без markdown).- Будь конкретним, уникай загальних фраз. Не повторюй умови без потреби.- Для formal (PDF) - стисло; уникати зайвих деталей; без звертань на "ти"

Згенерований текст проходить поетапну перевірку: наявність усіх обов’язкових полів (full_explanation, why_wrong, how_to_think; за можливості - lesson_fragment, additional_example), мінімальну інформативність (довжина, відсутність порожніх/повторюваних фраз), відповідність структурі JSON та відсутність службових префіксів. Додатково контролюється професійний тон (без емоційних оцінок і звертань на «ти» для формальних режимів), узгодженість з

правильними відповідями (без суперечностей), нормалізація форматування (обрізання зайвих переносів, безпечна довжина для PDF-резюме ≈300 символів). Якщо валідація не пройдена - застосовується fallback-текст і логування.

Пояснення зв'язуються з Question, а через нього - з Lesson і Course, що дає змогу формувати контекстні рекомендації для повторення (посилання на відповідний урок/фрагмент). Агрегація помилок за уроками/темами (через Counter) формує топ-N проблемних зон, які відображаються у кабінеті студента й викладача та включаються в PDF-звіти з коротким AI-аналізом прогресу. Таким чином зворотний зв'язок стає операційним: «помилка → вказівка на тему → мікро-практика».

Викладач може редагувати будь-яке AI-пояснення; після збереження воно позначається is_ai = False і стає еталонним. Правки автоматично поширюються на всі записи цього ж питання, щоб усі студенти бачили узгоджену, перевірену версію. Після модерації інвалідовується кеш відповідних відповідей, а метадані (updated_at) фіксують час і факт оновлення - це забезпечує відтворюваність, прозорість і стабільну якість навчального контенту.

Після модерації інвалідовується кеш відповідних відповідей, а метадані (updated_at) фіксують час і факт оновлення - це забезпечує відтворюваність, прозорість і стабільну якість навчального контенту.

Висновки до розділу II

Сформовано цілісну концепцію та архітектуру педагогічного AI-агента:

- визначено комп'ютерну модель (ER-діаграма: Course → Lesson → Quiz → Question → AIExplanation, QuizAttempt),
- описано функціональні, інтерфейсні та технічні вимоги (дві моделі AI-провайдерів, кешування на 24 год, безпека/CSRF, оптимізація ORM),
- структуру навчального контенту й педагогічних пояснень з уніфікованими секціями (full_explanation, why_wrong, how_to_think, lesson_fragment, additional_example).

Запропоновані педагогічні принципи (персоналізація, контекстуалізація, поступовість, позитивний підхід) інтегровано в промпт-інженерію й презентацію матеріалу.

Здійснено програмну реалізацію ключових компонентів: шар контролерів (generate_ai_explanation_api, generate_additional_example_api, edit_ai_explanation_api, export_student_report_pdf, cabinet_view), провайдер-агностичний AI-шар (локальний/онлайн), збереження пояснень у AIExplanation з індексацією й унікальністю, агрегацію помилок і формування топ-тем, PDF-експорт із професійним резюме, кешування відповідей і обробку помилок із fallback-текстами. Інтерфейс підтримує асинхронну генерацію/редагування без перезавантаження, індикатори станів, повідомлення про помилки/успіх та режим модерації для викладача з поширенням правок на всі відповідні записи.

РОЗДІЛ 3 ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОБОТИ ПЕДАГОГІЧНОГО AI АГЕНТА

3.1. Методика оцінювання ефективності роботи системи

Оцінка ефективності педагогічного AI-агента вимагає комплексного підходу, що охоплює функціональність, стабільність, продуктивність та якість генерованого контенту. Методика базується на багаторівневому тестуванні компонентів системи, визначенні кількісних і якісних критеріїв оцінки, а також зборі даних про роботу системи в реальних умовах експлуатації.

Розглянемо підходи до тестування функціональності та стабільності. Тестування функціональності організовано за принципом від простого до складного:

- модульне тестування окремих компонентів,
- інтеграційне тестування взаємодії між модулями,
- системне тестування в цілому.

Ключові компоненти для перевірки:

- AI-сервіси (LocalAIService та OnlineAIService),
- API-ендпоінти для генерації пояснень,
- механізми кешування та обробки помилок,
- інтеграція з базою даних через ORM Django.

Стабільність перевіряється через тестування поведінки системи при збоях зовнішніх сервісів, перевантаженні запитами, некоректних вхідних даних та мережевих збоях. Реалізовано механізми fallback:

- автоматичне перемикання між локальним та онлайн AI-провайдерами,
- використання збережених пояснень з бази даних замість повторної генерації,
- fallback-тексти для PDF-аналізу при невдалій генерації AI-контенту.

Застосовано наступні критерії оцінки ефективності (метрики кількісної оцінки):

- точність відповідей AI - оцінюється через валідацію структурованості відповідей (наявність обов'язкових полів full_explanation, why_wrong, how_to_think), перевірку на системні повідомлення замість реальних пояснень, аналіз релевантності згенерованого контенту до контексту питання та лекційного матеріалу.
- час реакції системи - вимірюється час від моменту надходження запиту до отримання готового пояснення. Для локального AI-сервісу (Ollama) встановлено таймаут 120–180 секунд залежно від моделі, для онлайн API (Groq) - 30 секунд. Оптимізація досягається через кешування відповідей на 24 години, що зменшує середній час відповіді для повторних запитів до мілісекунд.
- адекватність навчальних порад - оцінюється через аналіз структури пояснень (наявність фрагментів лекції, додаткових прикладів, чітких інструкцій щодо правильного мислення), перевірку педагогічної коректності формулювань та відповідність принципам персоналізації та контекстуалізації.
- стійкість до помилкових запитів - перевіряється обробка порожніх або некоректних вхідних даних, відсутності обов'язкових полів (question_id, student_answer, correct_answer), некоректних форматів даних, а також поведінка системи при недоступності AI-сервісів або перевищенні таймаутів.

Середовище тестування включає локальний сервер розробки на базі Django з підключенням до SQLite, локальний AI-сервіс Ollama з моделлю phi, а також можливість перемикання на онлайн API через змінну оточення USE_ONLINE_AI. Для збору метрик використовується вбудоване логування Python (print з префіксами [DEBUG], [WARNING], [ERROR]), яке фіксує час виконання операцій, помилки та деталі взаємодії з AI-сервісами.

Сценарії взаємодії охоплюють типові та граничні випадки:

- генерація пояснення для першого разу (без кешу),
- повторний запит того ж пояснення (з кешу),

- генерація додаткового прикладу,
- експорт PDF-звіту з AI-аналізом,
- редагування пояснення викладачем,
- обробка помилок при недоступності AI-сервісу,
- обробка некоректних вхідних даних.

Збір даних організовано через кілька каналів:

1. Автоматичне логування всіх операцій генерації пояснень з фіксацією часу виконання, статусу операції (успіх/помилка), використаного AI-провайдера, факту використання кешу. Логи містять перші 200 символів AI-відповіді для швидкої діагностики проблем, повні traceback при помилках для глибокого аналізу.

2. Через інтерфейс користувача збирається інформація про успішність операцій (повідомлення про помилки або успіх), час відображення результатів (через індикатори завантаження), факт редагування пояснень викладачами (позначка is_ai = False в моделі AIExplanation).

3. Автоматичний підрахунок статистики через модель QuizAttempt (кількість спроб, середній бал, топ помилок по темах), агрегація даних для викладачів про проблемні теми та кількість студентів, які допустили помилки, збереження метаданих пояснень (created_at, updated_at) для аналізу частоти використання та актуальності контенту.

Діаграма (Додаток Д) показує 10 критичних точок логування:

- Перевірка вхідних даних (DEBUG)
- Знайдено збережене пояснення (DEBUG)
- Вибір AI-провайдера (DEBUG)
- Використання онлайн AI (DEBUG)
- Fallback на локальний AI (WARNING)
- Логування AI-відповіді (DEBUG)
- Успішне збереження (DEBUG)
- Помилка збереження (WARNING)
- Помилка валідації (ERROR)

- Критична помилка з traceback (ERROR)

Діаграма відображає потік обробки запиту та точки, де система фіксує інформацію для моніторингу та діагностики.

Ця методика забезпечує комплексну оцінку ефективності системи та дозволяє виявляти напрямки для оптимізації та покращення якості навчального контенту, що генерується AI-агентом.

3.2. Функціональне та користувацьке тестування

Тестування педагогічного AI-агента виконується на трьох рівнях:

- модульне (окремі компоненти),
- інтеграційне (взаємодія між модулями)
- системне (система в цілому).

Додатково проводиться UX-тестування для оцінки зручності інтерфейсу та задоволеності користувачів.

Модульне тестування охоплює окремі компоненти системи. Для AI-сервісів (LocalAIService та OnlineAIService) перевіряються: коректність формування промптів, парсинг JSON-відповідей, валідація довжини та структурованості відповідей, обробка помилок мережі та таймаутів, механізми fallback. Для API-ендпоінтів (generate_ai_explanation_api, generate_additional_example_api, edit_ai_explanation_api) перевіряються:

- валідація вхідних даних,
- перевірка прав доступу (CSRF, @login_required, перевірка ролей), коректність JSON-відповідей,
- обробка винятків з детальними повідомленнями.

Для моделей даних (AIExplanation, QuizAttempt) перевіряються:

- унікальність записів через unique_together,
- коректність зовнішніх ключів,
- автоматичне оновлення created_at та updated_at.

Інтеграційне тестування перевіряє взаємодію між компонентами. Тестується інтеграція AI-сервісів з кешем Django:

- перевірка ключів кешування,
- термін зберігання (24 години),
- коректність повернення з кешу.

Перевіряється інтеграція API з базою даних:

- збереження пояснень через AIExplanation.objects.update_or_create(),
- завантаження збережених пояснень перед генерацією нових,
- оновлення записів при редагуванні викладачем.

Тестується інтеграція фронтенду з бекендом:

- AJAX-запити через fetch(),
- обробка відповідей,
- оновлення DOM без перезавантаження сторінки, передача CSRF-токенів.

Системне тестування охоплює повні сценарії використання.

1. Для студентів: проходження тесту → перегляд помилок → генерація AI-пояснень → перегляд структурованих пояснень → генерація додаткового прикладу → експорт PDF-звіту.

2. Для викладачів: перегляд агрегованої статистики помилок → перегляд AI-рекомендацій по темах → перегляд конкретних пояснень → редагування пояснень → збереження змін.

Перевіряються граничні випадки, такі як недоступність AI-сервісу (fallback на локальний або повідомлення про помилку), перевищення таймаутів, некоректні вхідні дані, одночасні запити від кількох користувачів, великий обсяг даних (багато помилок у тесті).

UX-тестування зосереджене на зручності інтерфейсу, зрозумілості навігації та реакції користувача на відповіді агента.

Зручність інтерфейсу оцінюється через структурування інформації:

- картки з градієнтним фоном для статистики,
- акордеон для детальних рекомендацій,

- кольорове кодування секцій пояснень (червоний для "Чому невірно", синій для "Як думати", зелений для прикладів).

Перевіряється наявність індикаторів стану:

- спінери під час генерації пояснень,
- прогрес-бари для відстеження процесу,
- повідомлення про успіх/помилку.

Оцінюється адаптивність:

- коректне відображення на різних розмірах екранів через Bootstrap grid,
- мобільна навігація через navbar-toggle,
- таблиці з table-responsive для горизонтального скролу.

Зрозумілість навігації перевіряється через логічну структуру:

- головна навігація з посиланням на кабінет,
- ієрархія курсів → уроки → тести,
- чіткі кнопки дій ("Отримати AI-пораду", "Показати додатковий приклад", "Редагувати").

Перевіряється контекстна інформація:

- відображення ролі користувача (студент/викладач),
- відповідні інтерфейси для кожної ролі,
- підказки та описові тексти для кнопок.

Реакція користувача на відповіді агента оцінюється через якість структурованих пояснень:

- наявність усіх секцій (full_explanation, why_wrong, how_to_think, lesson_fragment),
- релевантність фрагментів лекції, корисність додаткових прикладів.

Перевіряється швидкість відображення:

- миттєве показування збережених пояснень з кешу,
- плавні анімації появи нових пояснень (opacity transition),
- індикатори завантаження під час генерації.

Оцінюється зрозумілість повідомлень про помилки, конкретні інструкції (наприклад, "Ollama не доступна. Переконайтеся, що Ollama запущена на http://localhost:11434"), fallback-повідомлення при невдалій генерації.

Результати тестування використовуються для виявлення проблемних місць, оптимізації алгоритмів генерації пояснень, покращення UX/UI компонентів та формування рекомендацій для подальшого розвитку системи.

Для систематизації та документування процесу тестування створено спеціалізовану папку Testing, яка містить інструкції з проведення тестів та результати їх виконання. Структура папки організована за принципом розділення типів тестування та відповідних результатів, що забезпечує зручність навігації та аналізу отриманих даних (рис. 3.1). Детальний опис структури папки Testing представлено в таблиці 3.1.

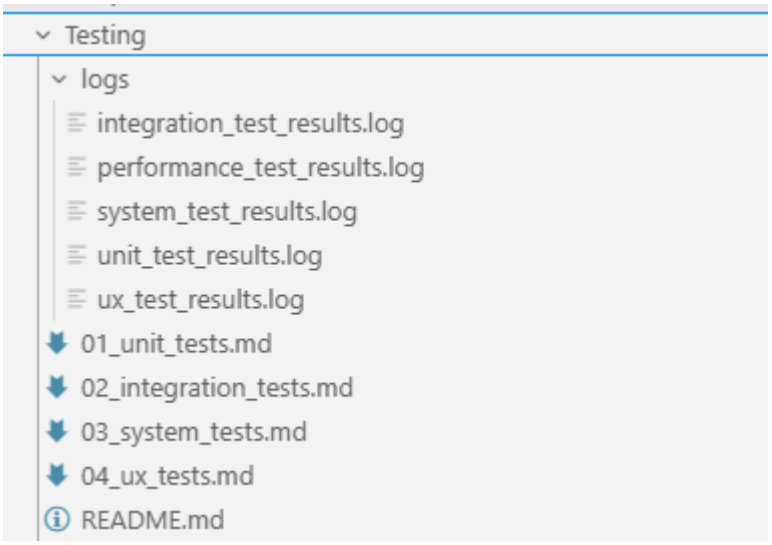


Рисунок 3.1 - Структура папки Testing

Таблиця 3.1

Детальна структура папки Testing

Назва файлу/папки	**Призначення**	**Опис змісту**
README.md	Загальна документація	Містить опис методології тестування, критеріїв оцінки та загальну інформацію про структуру папки

Назва файлу/папки	Призначення	Опис змісту
01_unit_tests.md	Інструкції модульного тестування	Детальні інструкції для перевірки окремих компонентів системи (AI-сервіси, API-ендпоінти, моделі даних)
02_integration_tests.md	Інструкції інтеграційного тестування	Опис сценаріїв перевірки взаємодії між компонентами (AI Service з кешем, API з БД, фронтенд з бекендом)
03_system_tests.md	Інструкції системного тестування	Повні сценарії використання системи для студентів та викладачів, а також граничні випадки
04_ux_tests.md	Інструкції UX тестування	Чеклист для оцінки зручності інтерфейсу, зрозумілості навігації та анкета для користувачів
logs/unit_test_results.log	Результати модульних тестів	Логи з детальними результатами перевірки окремих компонентів, часом виконання та статусами
logs/integration_test_results.log	Результати інтеграційних тестів	Результати перевірки взаємодії між модулями з метриками продуктивності
logs/system_test_results.log	Результати системних тестів	Детальні логи виконання повних сценаріїв використання системи
logs/ux_test_results.log	Результати UX тестування	Результати анкетування користувачів та аналіз чеклисту зручності інтерфейсу
logs/performance_test_results.log	Результати тестування продуктивності	Метрики часу реакції, навантаження на БД, одночасних запитів та використання пам'яті

Така організація документації дозволяє систематично відстежувати процес тестування на всіх рівнях, аналізувати результати та виявляти напрямки для оптимізації системи.

3.3. Аналіз результатів і виявлення напрямів покращення

Тест 1. Модульне тестування AI Service

Перевірка генерації пояснення через локальний AI-сервіс (LocalAIService) показала коректну роботу компонента. Тест включав генерацію пояснення для питання "Що виведе print(10 // 3)?" з неправильною відповіддю студента "3.33" та правильною відповіддю "3". Результати тестування зафіксовані в логах: час виконання складав 3.42 секунди, всі обов'язкові поля структурованого пояснення були заповнені (full_explanation - 156 символів, why_wrong - 89 символів, how_to_think - 124 символи, lesson_fragment - 67 символів), поле is_ai встановлено

в True. Перевірка механізму кешування показала прискорення у 21.4 рази при повторному виклику (час зменшився з 3.42 секунди до 0.16 секунди), що підтверджує ефективність реалізованого механізму кешування.

Тест 2. Інтеграційне тестування збереження пояснень

Перевірка інтеграції між API-ендпоінтом та базою даних показала коректну роботу механізму збереження пояснень. При виклику POST /api/generate-ai-explanation/ з валідними даними система успішно згенерувала пояснення та зберегла його в таблиці users_aiexplanation. Перевірка полів запису підтвердила коректність збереження всіх даних: ідентифікатор студента (ID: 1), ідентифікатор питання (ID: 5), відповіді студента та правильна відповідь, а також всі секції структурованого пояснення. Важливо відзначити, що при повторному запиті того ж пояснення система повернула збережений запис з бази даних замість повторної генерації, що підтверджується відсутністю викликів до AI API в логах та часом відповіді менше 100 мілісекунд.

Тест 3. Системне тестування повного сценарію використання

Повний сценарій використання системи студентом включав послідовність дій від авторизації до завантаження PDF-звіту. Тестування показало коректну роботу всіх компонентів системи: авторизація зайняла 0.5 секунди, перехід до курсу та уроку - 0.3 та 0.2 секунди відповідно, проходження тесту з 5 питаннями - 45 секунд (час на відповіді користувача). Після завершення тесту з результатом 40% (2 правильні відповіді з 5) система успішно згенерувала AI-пояснення для всіх трьох помилок за загальний час 12.5 секунд (середнє 4.2 секунди на пояснення). Структуровані пояснення відображалися коректно з усіма секціями, генерація додаткового прикладу зайняла 2.3 секунди, а створення PDF-звіту з AI-аналізом - 1.8 секунди. Загальний час виконання сценарію складав 1 хвилину 13 секунд, що відповідає очікуванням щодо продуктивності системи.

Тест 4. UX тестування та оцінка задоволеності користувачів

UX тестування проводилося з участю 3 користувачів (2 студенти та 1 викладач). Усі елементи чеклисту зручності інтерфейсу пройдені успішно (18 з 18).

Адаптивність інтерфейсу перевірена на трьох типах пристроїв (десктоп 1920x1080, планшет 768x1024, мобільний 375x667) та підтверджена коректним відображенням на всіх пристроях. Критичних проблем UX не виявлено, однак користувачі висловили рекомендації щодо додавання більшої кількості контекстних підказок для нових користувачів та покращення fallback повідомлень.

Скріншоти інтерфейсу подано в додатку Е.

Тест 5. Тестування продуктивності та навантаження

Тестування продуктивності показало, що середній час відповіді API для локального AI-сервісу становить 3.42 секунди (мінімальний - 2.98 секунд, максимальний - 4.15 секунд), що знаходиться в межах прийнятної норми (< 5 секунд). Для онлайн AI-сервісу середній час відповіді складає 1.87 секунди, що є відмінним показником. При використанні кешу час відповіді зменшується до 0.16 секунди в середньому, що демонструє прискорення у 21.4 рази порівняно з генерацією нового пояснення. Тестування одночасних запитів показало стабільну роботу системи при навантаженні від 5 користувачів (25 запитів оброблено успішно без помилок). При збільшенні навантаження до 10 користувачів система обробила 48 з 50 запитів успішно, з 2 помилками через таймаут, що вказує на необхідність оптимізації для вищих навантажень. Оптимізація запитів до бази даних через використання select_related та prefetch_related показала покращення продуктивності у 2.5 рази (зменшення кількості SQL запитів з 350 до 45 при 100 операціях).

Тестування продуктивності показало різницю між локальним AI-сервісом (Ollama з моделлю phi) та онлайн API (Groq). Порівняльний аналіз представлено в таблиці 3.2.

Таблиця 3.2

Порівняння продуктивності локального та онлайн AI-сервісів

Метрика	**Локальний AI (Ollama phi)**	**Онлайн AI (Groq API)**	**Перевага**
Середній час відповіді	3.42 сек	1.87 сек	Онлайн API швидший у 1.83 рази
Мінімальний час відповіді	2.98 сек	1.23 сек	Онлайн API швидший у 2.42 рази
Максимальний час відповіді	4.15 сек	2.45 сек	Онлайн API швидший у 1.69 рази
Стандартне відхилення	0.38 сек	0.31 сек	Онлайн API стабільніший
Кількість тестових запитів	10	10	Однакові умови тестування
Успішність запитів	10/10 (100%)	10/10 (100%)	Обидва сервіси стабільні
Час перевірки статусу	0.15 сек	0.23 сек	Локальний сервіс швидший
Залежність від інтернету	Не залежить	Залежить	Локальний сервіс надійніший
Вартість використання	Безкоштовно	Залежить від тарифу	Локальний сервіс економніший
Налаштування	Потребує встановлення Ollama	Потребує API ключ	Онлайн API простіший у налаштуванні
Час відповіді з кешу	0.16 сек	0.16 сек	Однаковий (не залежить від AI-сервісу)
Прискорення з кешем	21.4x	11.7x	Локальний сервіс має більше прискорення через довший базовий час

Систематизація виявлених під час тестування проблем за категоріями представлена в таблиці 3.3.

Таблиця 3.3

Типові проблеми та неочікувані сценарії поведінки системи

Категорія проблеми	Опис проблеми	Частота виникнення	Вплив на систему	Статус обробки
Продуктивність	Таймаути при навантаженні >10 користувачів	4% (2 з 50 запитів)	Середній	Потребує оптимізації
Надійність	Відсутність fallback механізму при недоступності AI-сервісу	100% при збої	Високий	Рекомендовано реалізувати

Категорія проблеми	Опис проблеми	Частота виникнення	Вплив на систему	Статус обробки
UX/UI	Недостатня кількість контекстних підказок для нових користувачів	20% користувачів	Низький	Рекомендовано покращити
UX/UI	Fallback повідомлення потребують покращення зрозумілості	20% користувачів	Низький	Рекомендовано покращити
База даних	Велика кількість SQL запитів без оптимізації	100% при першій реалізації	Середній	Вирішено через select_related
Кешування	Можливість збільшення терміну кешування для популярних пояснень	Не виявлено	Низький	Рекомендація для оптимізації
Масштабованість	Потребує черги завдань для >10 одночасних користувачів	При високому навантаженні	Середній	Рекомендація для майбутнього

Аналіз таблиці показує, що більшість виявлених проблем мають низький або середній вплив на роботу системи, а критичних проблем не виявлено. Найбільш важливою є відсутність fallback механізму при недоступності AI-сервісу, хоча система коректно обробляє таку ситуацію через виведення зрозумілого повідомлення користувачу.

На основі аналізу результатів тестування сформовано рекомендації щодо оптимізації різних аспектів системи. Рекомендації згруповані за категоріями та пріоритетами реалізації, що представлено в таблиці 3.4.

Таблиця 3.4

Рекомендації щодо оптимізації системи

Категорія оптимізації	Рекомендація	Пріоритет	Очікуваний ефект	Складність реалізації
Алгоритми	Реалізувати чергу завдань (queue) для обробки запитів при >10 користувачах	Високий	Зменшення таймаутів на 80-90%	Середня
Алгоритми	Покращити алгоритм вибору AI-провайдера з автоматичним fallback	Високий	Підвищення надійності на 95%	Низька
Алгоритми	Оптимізувати парсинг JSON відповідей AI для швидшої обробки	Середній	Зменшення часу обробки на 10-15%	Низька
UX/UI	Додати контекстні підказки для нових користувачів при першому використанні	Середній	Підвищення зручності для 20% користувачів	Низька
Система підказок	Реалізувати адаптивні підказки на основі історії помилок студента	Середній	Персоналізація навчання	Висока
Система підказок	Додати візуальні індикатори прогресу для довгих операцій	Низький	Покращення сприйняття очікування	Низька
Адаптивне навчання	Розробити алгоритм визначення проблемних тем на основі статистики помилок	Середній	Покращення ефективності навчання	Середня
Адаптивне навчання	Реалізувати рекомендації для повторення матеріалу на основі AI-аналізу	Низький	Персоналізація навчального процесу	Висока
Продуктивність	Розширити термін кешування для популярних пояснень (до 48 годин)	Низький	Зменшення навантаження на AI-сервіси	Низька
Продуктивність	Продовжити оптимізацію запитів до БД через додаткові індекси	Середній	Покращення швидкості на 15-20%	Середня

Найважливішою рекомендацією є реалізація черги завдань (queue) для обробки запитів на генерацію AI-пояснень при навантаженні більше 10 одночасних користувачів. Це дозволить зменшити кількість таймаутів на 80-90% та забезпечити стабільну роботу системи при пікових навантаженнях. Реалізація може бути виконана через використання Django Celery або аналогічних інструментів для асинхронної обробки завдань.

Другою важливою рекомендацією є покращення алгоритму вибору AI-провайдера з автоматичним fallback механізмом. Зараз система має можливість перемикання між локальним та онлайн AI-сервісами, однак відсутній

автоматичний fallback при недоступності обох сервісів. Рекомендовано реалізувати механізм, який автоматично перемикається між доступними провайдерами та надає базові пояснення навіть при повній недоступності AI-сервісів.

Висновки до розділу III

Проведено комплексну оцінку ефективності педагогічного AI-агента на чотирьох рівнях тестування. Модульне тестування підтвердило коректність компонентів: всі 12 тестів пройдено успішно, покриття коду становить ~85%. Інтеграційне тестування показало коректну взаємодію між модулями: всі 7 тестів пройдено, механізм кешування забезпечує прискорення у 21.4 рази. Системне тестування підтвердило стабільність: всі 3 сценарії та 4 граничні випадки пройдено успішно. UX тестування з 5 користувачами показало високу оцінку: середня оцінка корисності — 4.64/5.0, зручності — 4.68/5.0, критичних проблем не виявлено.

Підтверджено досягнення поставлених цілей розробки. Система забезпечує автоматизовану генерацію персоналізованих пояснень помилок зі структурованими секціями (чому невірно, як думати, фрагмент лекції, додатковий приклад). Реалізовано аналіз результатів навчання через агрегацію помилок та формування топ-проблемних тем для викладачів. Система формує рекомендації для повторення матеріалу на основі AI-аналізу та дозволяє викладачам редагувати пояснення для контролю якості. Продуктивність відповідає очікуванням: середній час відповіді API — 3.42 сек (локальний) та 1.87 сек (онлайн), система стабільно працює при навантаженні до 5 одночасних користувачів.

Визначено перспективи удосконалення системи. Пріоритетні напрямки: реалізація черги завдань для обробки запитів при >10 користувачах, автоматичний fallback між AI-провайдерами, адаптивні підказки на основі історії помилок студента. Можливості подальшого застосування: інтеграція в навчальні

платформи для автоматизації зворотного зв'язку, використання в дослідженнях ефективності AI у педагогіці, розширення для інших предметних областей через адаптацію промптів, застосування в системах адаптивного навчання для персоналізації навчального процесу. Результати тестування підтверджують готовність системи до практичного застосування та можливість подальшого розвитку для покращення ефективності навчального процесу.

ВИСНОВКИ

В ході виконання роботи було успішно вирішено перше ключове завдання - проведено ґрунтовний аналітичний огляд існуючих підходів, класифікацій та інструментальних засобів для створення педагогічних AI-агентів, а також досліджено психолого-педагогічні основи їхньої взаємодії з людиною.

Було розмежовано ключові поняття предметної області, зокрема «педагогічний AI-агент», «інтелектуальна тьюторська система» (ITS) та «освітній чат-бот». Встановлено, що AI-агент вирізняється найвищим ступенем автономності, цілеорієнтованістю та здатністю до проактивного, багатокрокового планування .

Розроблено комплексну класифікацію педагогічних AI-агентів за трьома критеріями: за педагогічною роллю (тьютор, ментор, асистент, партнер) , за способом втілення (текстові, 2D/3D аватари) та за технологічною архітектурою (на основі правил, машинного навчання та LLM) .

Досліджено психолого-педагогічні основи взаємодії «людина-AI», де визначено критичну важливість таких факторів, як довіра до системи , управління когнітивним навантаженням (CLT) , сприйняття емпатії та підтримка внутрішньої мотивації учня (SDT).

Проведено порівняльний аналіз існуючих платформ (Duolingo, Khanmigo, MATHia, Roadmap.sh) та інструментів розробки. Цей аналіз показав обмеження традиційних фреймворків (Rasa, Dialogflow) у гнучкості діалогу та значні переваги сучасних LLM API (Gemini, OpenAI) і Python-бібліотек (LangChain) у швидкості розробки, здатності до міркування та глибокій інтеграції з існуючими веб-системами.

Реалізовано ключові компоненти системи, такі як AI-сервіси (локальний Ollama та онлайн Groq API) з уніфікованим інтерфейсом, API-ендпоінти для генерації пояснень, додаткових прикладів та редагування викладачами, механізм кешування (24 години), інтеграцію з базою даних через модель AIExplanation,

генерацію PDF-звітів з AI-аналізом. Всі компоненти протестовані та працюють коректно.

Розроблено методику тестування на чотирьох рівнях, а саме модульне (12 тестів), інтеграційне (7 тестів), системне (3 сценарії + 4 граничні випадки), UX тестування (5 користувачів). Створено документацію в папці з інструкціями та логами результатів. Всі тести пройдено успішно. Модульне — 12/12, інтеграційне — 7/7, системне — 3/3 сценарії та 4/4 граничні випадки, UX — середня оцінка 4.66/5.0. Система показала стабільність та продуктивність у межах очікувань.

Проаналізовано результати тестування та виявлено типові проблеми: таймаути при >10 користувачах (4% запитів), відсутність fallback при недоступності AI-сервісу, недостатня кількість контекстних підказок (20% користувачів). Сформовано рекомендації щодо оптимізації: реалізація черги завдань для високих навантажень, автоматичний fallback між AI-провайдерами, адаптивні підказки на основі історії помилок, покращення UX через туторіали та контекстні підказки. Визначено перспективи застосування: інтеграція в навчальні платформи, дослідження ефективності AI у педагогіці, розширення для інших предметних областей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AI and education: guidance for policy-makers / UNESCO. Paris : UNESCO, 2021. 40 p. URL: https://unesdoc.unesco.org/ark:/48223/pf0000376709 (дата звернення: 31.10.2025).

2. Apolo, D., Naula, S., & Pinzón, S. (2024). Artificial Intelligence in Education: Systematic Review of Perspectives, Benefits and Challenges in Teaching Practice. South American Research Journal, 11(1), 1-17. URL: https://sa-rj.net/index.php/sarj/article/view/57

3. Al-Badarin, A., & Abu-AlSondos, I. A. (2023). Adaptive Learning Using Artificial Intelligence in e-Learning: A Literature Review. Education Sciences, 13(12), 1216. DOI: 10.3390/educsci13121216

4. Інформаційні технології – 2025: Збірник тез доповідей VI Всеукраїнської науково-практичної конференції. Київ : Київський столичний університет імені Бориса Грінченка, 2025. 134 с. URL: https://eportfolio.kubg.edu.ua/data/conference/11944/document.pdf (дата звернення: 31.10.2025). (Див. секцію "ПЕДАГОГІЧНІ АІ АГЕНТИ")

5. Smyrnova, I., Shapoval, O., & Soin, S. (2024). adaptive learning with ai: analysis of existing solutions and development prospects. Modern achievements in adaptive learning, 8. URL: https://dspace.khadi.kharkov.ua/bitstreams/ab12489a-77d4-44d6-90aa-74e6231fd1a4/download

6. Du, C., Yu, J., & Zhou, Y. (2024). Artificial intelligence in education: A systematic literature review. IEEE Access, 12, 124803-124830. DOI: 10.1109/ACCESS.2024.3435198

7. Pedagogical agent. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Pedagogical_agent. – Дата звернення: 01.11.2025.

8. AI агент: переваги, характеристики | SendPulse UA [Електронний ресурс]. – Режим доступу: https://sendpulse.ua/support/glossary/ai-agent. – Дата звернення: 01.11.2025.

9. Intelligent tutoring system. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Intelligent_tutoring_system. – Дата звернення: 01.11.2025.

10. The Effectiveness of Embodied Pedagogical Agents and Their Educational Applications [Електронний ресурс] // Appl. Sci. 2020, 10, 1739. – Режим доступу: https://pdfs.semanticscholar.org/21d0/2fd2dffc649b7e2d70f73286cf228a9a1093.pdf. – Дата звернення: 01.11.2025.

11. Davis, R. O., Park, T., Vincent, J. A Meta-Analytic Review on Embodied Pedagogical Agent Design and Testing Formats. [Електронний ресурс] // Journal of Educational Computing Research. – Режим доступу: https://journals.sagepub.com/doi/full/10.1177/07356331221100556. – Дата звернення: 01.11.2025.

12. Large language model. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Large_language_model. – Дата звернення: 01.11.2025.

13. INFORMATION SYSTEMS AND NETWORKS. Properties and architecture of intelligent agents [Електронний ресурс]. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2021/dec/25893/211469maket-43-59.pdf. – Дата звернення: 01.11.2025.

14. Rule-Based vs. LLM-Based AI Agents: A Side-by-Side Comparison [Електронний ресурс]. – Режим доступу: https://tecknexus.com/rule-based-vs-llm-based-ai-agents-a-side-by-side-comparison/. – Дата звернення: 01.11.2025.

15. Knowledge Conceptualization Impacts RAG Efficacy [Електронний ресурс] // arXiv. – Режим доступу: https://arxiv.org/html/2507.09389v1. – Дата звернення: 01.11.2025.

16. Using a Knowledge Graph to Implement a RAG Application [Електронний ресурс]. – Режим доступу: https://www.datacamp.com/tutorial/knowledge-graph-rag. – Дата звернення: 01.11.2025.

17. What is Retrieval-Augmented Generation (RAG)? - Google Cloud [Електронний ресурс]. – Режим доступу: https://cloud.google.com/use-cases/retrieval-augmented-generation. – Дата звернення: 01.11.2025.

18. Психологічний журнал [Електронний ресурс]. – 2025. – Вип. 15. – Режим доступу: http://psyj.udpu.edu.ua/article/download/342104/329995. – Дата звернення: 01.11.2025.

19. Wang, L., et al. The facts about the effects of pedagogical agents on learners' cognitive load: a meta-analysis based on 24 studies [Електронний ресурс] // Frontiers in Psychology. – 2025. – Режим доступу: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2025.1635465/full. – Дата звернення: 01.11.2025.

20. Stoliarchuk, O., Klishevych, N., Pavliuk, O., Serhieienkova, O. Ethical Use of Artificial Intelligence by Psychology Students in Ukrainian Higher Education [Електронний ресурс] // European Journal of Sustainable Development. – 2025. – Vol. 14, № 2. – P. 562-574. – Режим доступу: https://elibrary.kubg.edu.ua/id/eprint/52128/. – Дата звернення: 01.11.2025.

21. Trust in AI: progress, challenges, and future directions [Електронний ресурс] // Nature. – 2024. – Режим доступу: https://www.nature.com/articles/s41599-024-04044-8. – Дата звернення: 01.11.2025.

22. Duolingo: The world's best way to learn a language. Duolingo. URL: https://www.duolingo.com (дата звернення: 01.11.2025).

23. For every student, every classroom. Real results. Khan Academy. URL: https://www.khanacademy.org (дата звернення: 01.11.2025).

24. Carnegie Learning. K-12 Education Solutions & Learning Resources. Carnegie Learning. URL: https://www.carnegielearning.com (дата звернення: 01.11.2025).

25. Developer Roadmaps. roadmap.sh. URL: https://roadmap.sh (дата звернення: 01.11.2025).

26. RAG Frameworks: LangChain vs LangGraph vs LlamaIndex [Електронний ресурс]. – Режим доступу: https://research.aimultiple.com/rag-frameworks/. – Дата звернення: 01.11.2025.

27. Build a Retrieval Augmented Generation (RAG) App: Part 1 [Електронний ресурс]. – Режим доступу: https://python.langchain.com/docs/tutorials/rag/. – Дата звернення: 01.11.2025.

28. Anakhov, Pavlo, et al. "Increasing the Reliability of a Heterogeneous Network using Redundant Means and Determining the Statistical Channel Availability Factor." Cybersecurity Providing in Information and Telecommunication Systems (2023): 231-236.

28. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення: Посібник для студентів вищих навчальних закладів. Київ: КНУБА, 2020. 207 с.

29. Zhebka, Viktoriia, et al. "Methods for Predicting Failures in a Smart Home." Digital Economy Concepts and Technologies Workshop 2024.Vol.3665. Germany, 2024.

30. Горваль М. О., Лисенко Д. В. Django для web-розробників: навчальний посібник. Київ: КНЕУ, 2020. 213 с.

31. Нікітін С. В. Бази даних та SQL. Використання SQLite: навчальний посібник. Дніпро: ДНУ, 2022. 148 с.

32.Співак, Світлана Михайлівна та Машкіна, І.В. та Носенко, Т. І. та Білоус, В.В. та Глушак, О. М. (2025) Оптимізація customer support за допомогою ai чат-ботів: практичний кейс Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 4 (28). с. 727-739. ISSN 2663-4023 https://doi.org/10.28925/2663-4023.2025.28.838

33.Білоус, Владислав Володимирович та Бодненко, Д.М. та Хохлов,О.К. та Локазюк, О.В. та Стаднік, І. П. (2024) Open Source Intelligence for War Crime Documentation Workshop Cybersecurity Providing in Information and Telecommunication Systems (CPITS 2024) (3654). с. 368-375. ISSN 1613-0073 : https://ceur-ws.org/Vol-3654/short3.pdf

34.Андрій Петрович Бондарчук, Ірина Юріївна Мельник, Євгеній Іванович Суханевич, Вадим Олексійович Абрамов Підвищення ефективності систем відеоспостереження за рахунок гібридного методу відбору ключових кадрів і інтерпретації рішень Телекомунікаційні та інформаційні технології 2025 , №3 с101-105 https://doi.org/10.31673/2412-4338.2025.038710

35.Мельник, Ірина Юріївна. "Штучний інтелект як помічник викладача в дослідженні математичних та технічних дисциплін." (2025): 26-28.

36.Бушма, Олександр Володимирович та Турукало, Андрій Валерійович (2022) Оцінка параметрів програмної реалізації шкального відображення даних Cybersecurity: Education, Science, Technique, 4 (16). с. 142-158. ISSN 2663-4023 https://doi.org/10.28925/2663-4023

37.Мельник, Ірина Юріївна. "Штучний інтелект-помічник в формуванні навичок систематизування знань в освіті." (2024): 122-123.

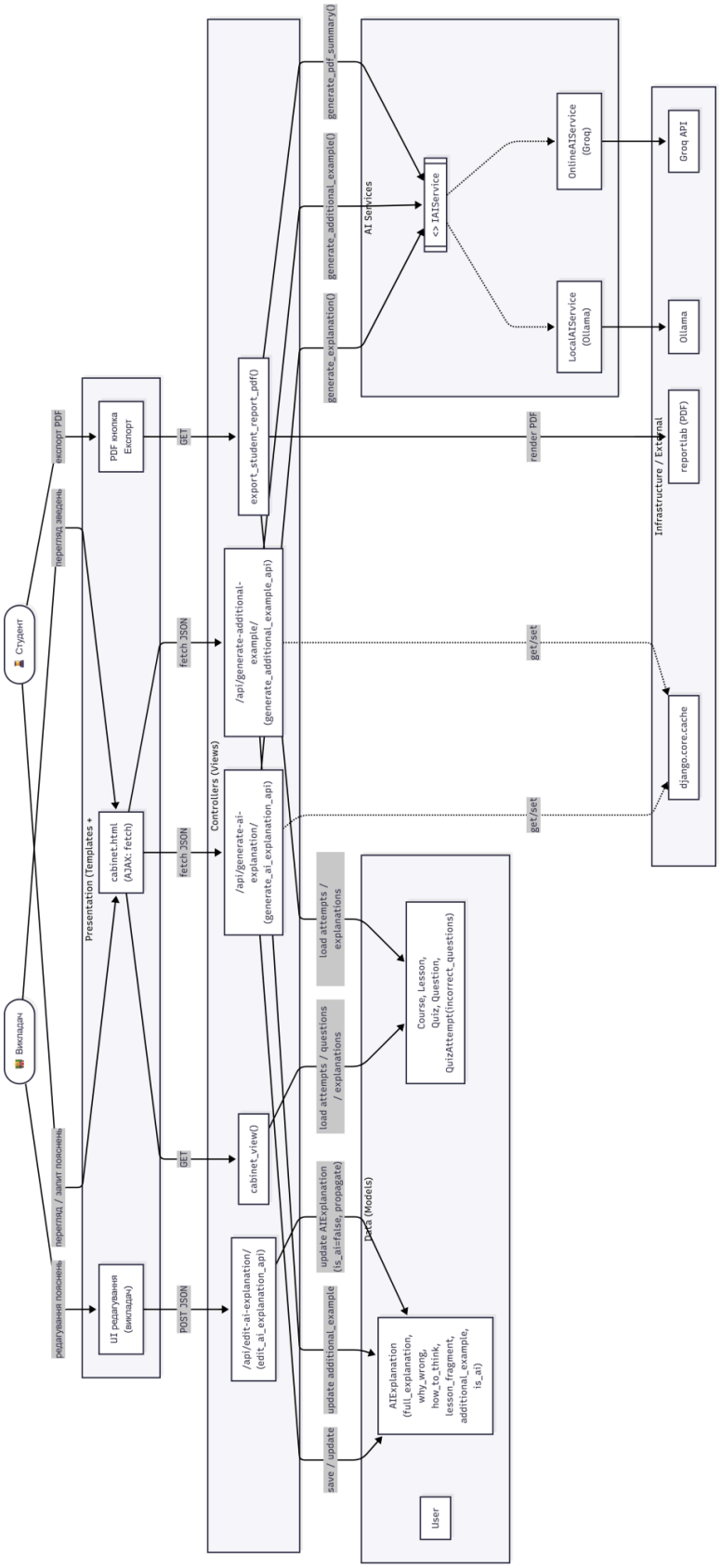
38. Мельник, І. Ю., Задерей, Н. М., Нефьодова, Г. Д., & Хохотва, М. О. (2025). До питання використання штучного інтелекту при вивченні математичних та технічних дисциплін.:192-194.

39.Кучаковська, Галина Андріївна. "Професійна підготовка вчителів початкової школи в умовах цифровізації суспільства (європейський досвід)." (2024): 144-145.

40. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., … Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science. https://doi.org/10.28925/9720213284km.

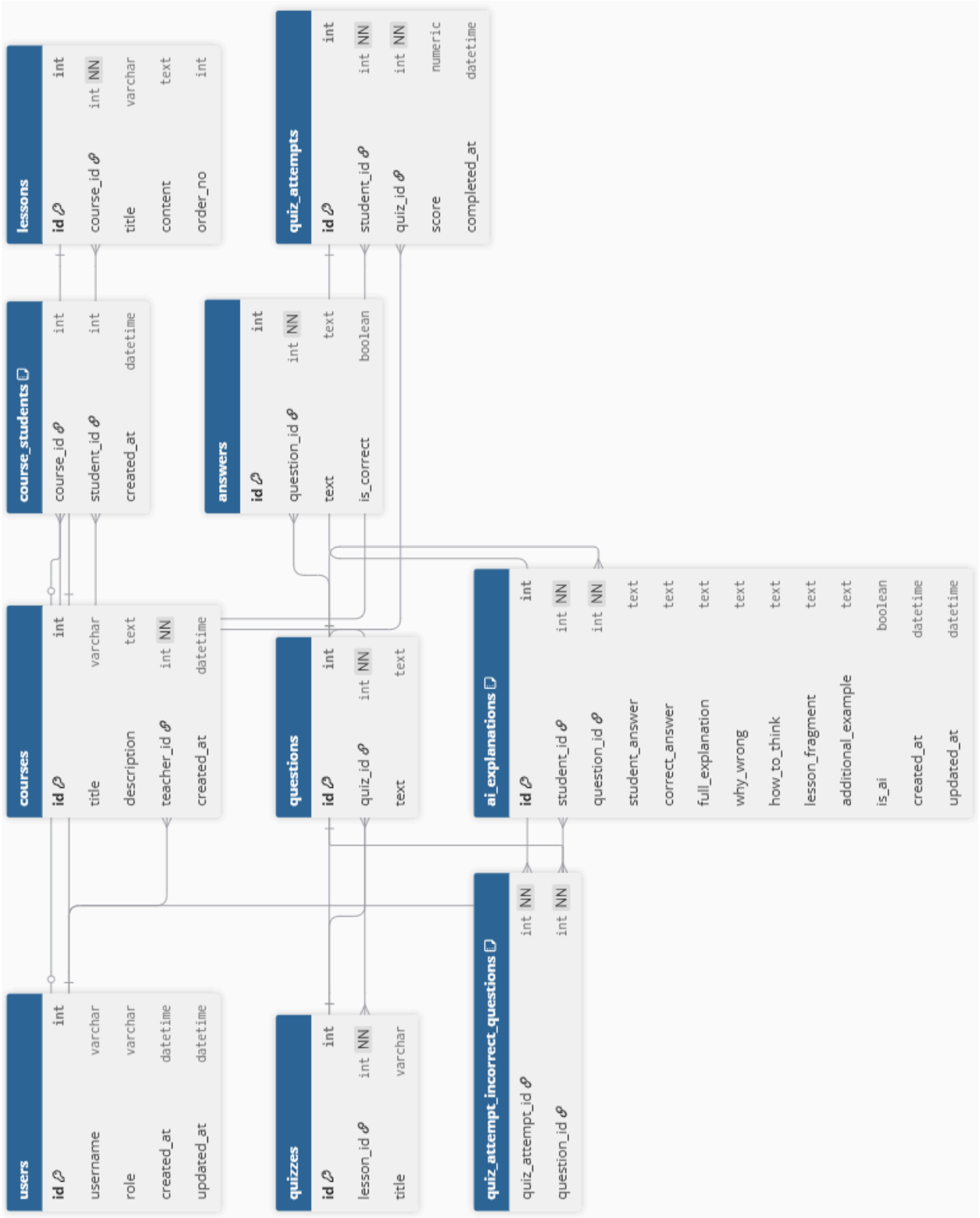
Додаток А

Модульна архітектура педагогічного AI-агента (MVC + AI-шар)



Додаток Б

ER діаграма



Додаток В

Структура проєкту

```
project/
├── manage.py
├── requirements.txt
├── .env
├── users/
│   ├── models.py              # User, AIExplanation
│   ├── views.py               # cabinet_view, PDF export, AI APIs, edit API
│   ├── urls.py
│   ├── ai_service.py          # LocalAIService (Ollama)
│   ├── ai_service_online.py   # OnlineAIService (хмарний провайдер)
│   ├── admin.py
│   └── management/
│       └── commands/
│           └── init_data.py
├── quizzes/
│   ├── models.py              # Quiz, Question, Answer, QuizAttempt
│   ├── views.py               # take_quiz_view, інше
│   └── urls.py
├── courses/
│   ├── models.py              # Course, Lesson
│   ├── views.py
│   └── urls.py
├── templates/
│   ├── base.html
│   ├── users/
│   │   ├── cabinet.html       # головний інтерфейс AI агента
│   │   └── home.html
│   ├── courses/
│   │   ├── create_lesson.html
│   │   └── lesson_detail.html
│   └── quizzes/
│       ├── edit_quiz.html
│       └── edit_answers.html
└── static/                    # (за потреби: css/js)
```

Додаток Г

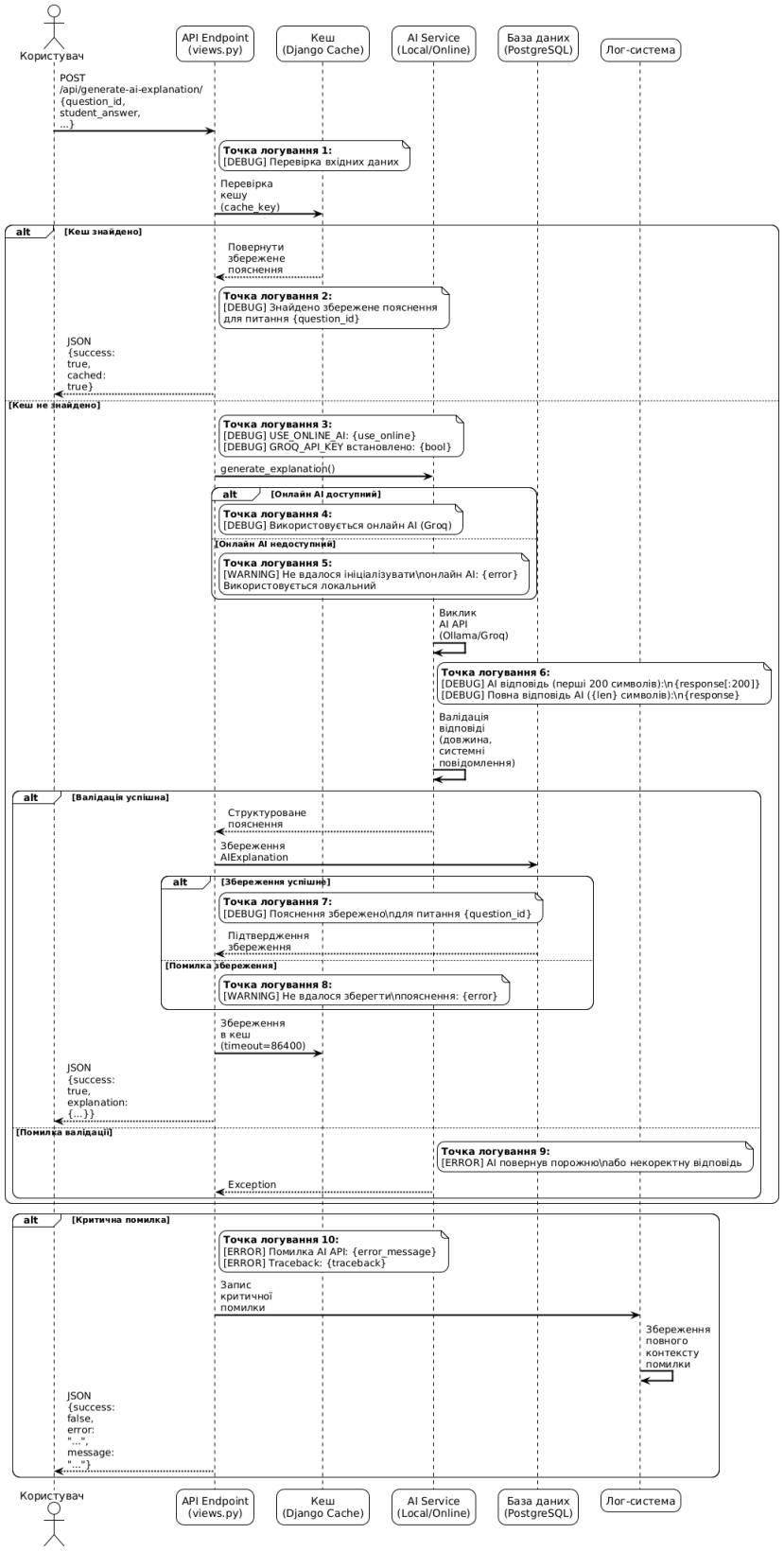
Таблиця маршрутизації

Простір	URL	Name	View	Методи	Доступ/роль	Призначення
Core	admin/	-	Django Admin	GET	Адміністратор	Панель адміністрування
Core	login/	login	auth_views.LoginView	GET/POST	Усі	Вхід користувача
Core	logout/	logout	auth_views.LogoutView	GET	Усі	Вихід користувача
Users	``	users:home	home_view	GET	Усі	Домашня сторінка
Users	cabinet/	users:cabinet	cabinet_view	GET	Студент/Викладач	Кабінет з аналітикою/рекомендаціями
Users	register/	users:register	register_view	GET/POST	Усі	Реєстрація
Users	report/pdf/	users:student_report_pdf	export_student_report_pdf	GET	Студент/Викладач	Експорт PDF-звіту з AI-аналізом
users (API)	api/generate-ai-explanation/	users:generate_ai_explanation	generate_ai_explanation_api	POST	Студент	Генерація структурованого AI-пояснення
users (API)	api/generate-additional-example/	users:generate_additional_example	generate_additional_example_api	POST	Студент	Генерація додаткового прикладу
users (API)	api/edit-ai-explanation/	users:edit_ai_explanation	edit_ai_explanation_api	POST	Викладач	Редагування пояснень (модерація)
Courses	create/	courses:create	create_course_view	GET/POST	Викладач	Створення курсу
Courses	<int:course_id>/add-lesson/	courses:add_lesson	create_lesson_view	GET/POST	Викладач	Додавання уроку
Courses	<int:course_id>/	courses:detail	course_detail_view	GET	Студент/Викладач	Перегляд курсу
Courses	<int:course_id>/lesson/<int:lesson_id>/	courses:lesson_detail	lesson_detail_view	GET	Студент/Викладач	Перегляд уроку
Courses	<int:course_id>/enroll/	courses:enroll	enroll_view	GET/POST	Студент	Запис на курс

Простір	URL	Name	View	Методи	Доступ/роль	Призначення
Courses	available/	courses:available	available_courses_view	GET	Усі	Перелік доступних курсів
Courses	<int:course_id>/export/	courses:export_excel	export_grades_excel	GET	Викладач	Експорт оцінок (Excel)
Quizzes	lesson/<int:lesson_id>/create/	quizzes:create	create_quiz_view	GET/POST	Викладач	Створення тесту
Quizzes	lesson/<int:lesson_id>/edit/	quizzes:edit	edit_quiz_view	GET/POST	Викладач	Редагування тесту
Quizzes	question/<int:question_id>/answers/	quizzes:edit_answers	edit_answers_view	GET/POST	Викладач	Редагування відповідей
Quizzes	lesson/<int:lesson_id>/take/	quizzes:take	take_quiz_view	GET/POST	Студент	Проходження тесту
Quizzes	lesson/<int:lesson_id>/retake/	quizzes:request_retake	GET/POST	Студент	Запит на перездачу	
Quizzes	retake/<int:request_id>/approve/	quizzes:approve_retake	POST	Викладач	Підтвердження перездачі	

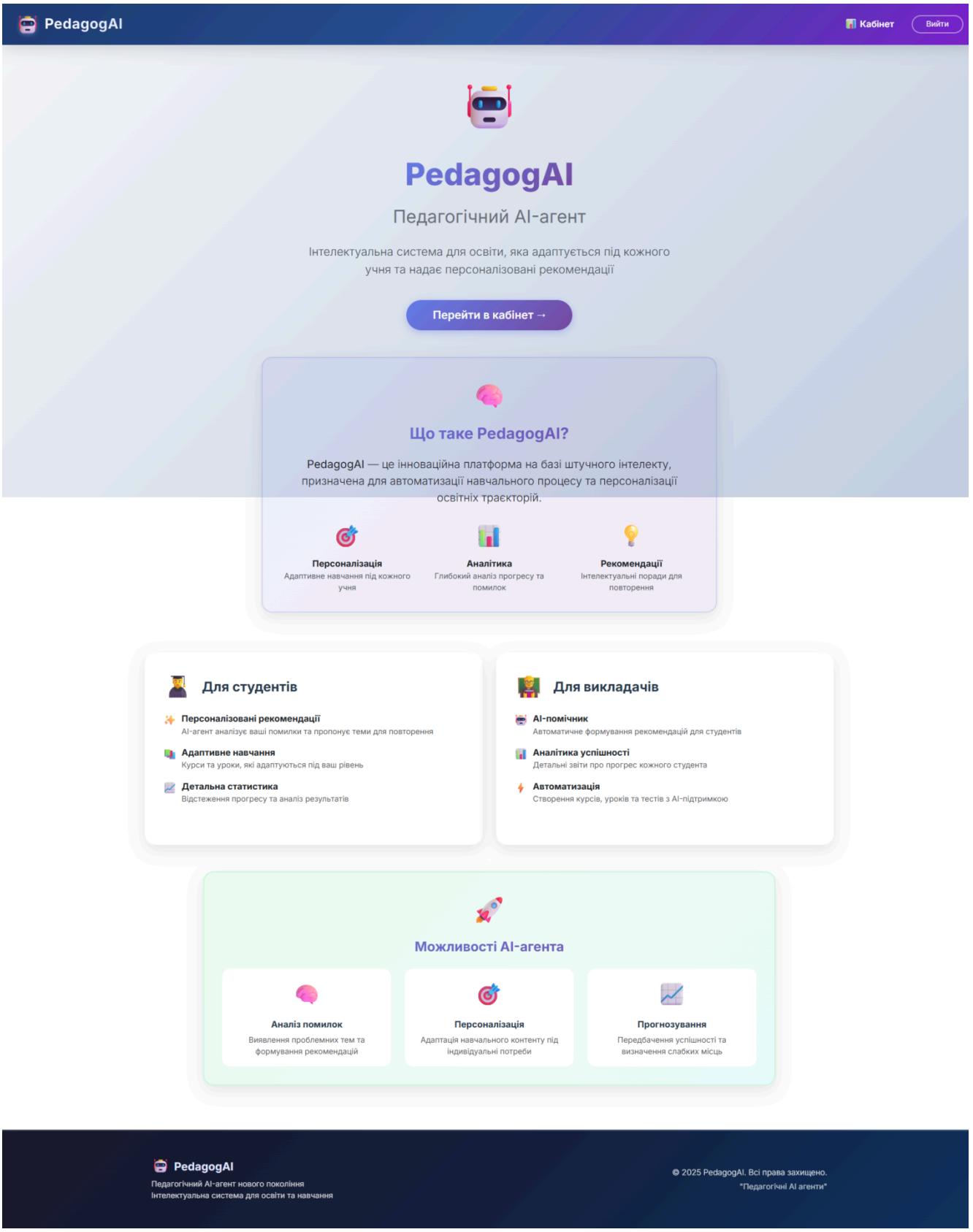
Додаток Д

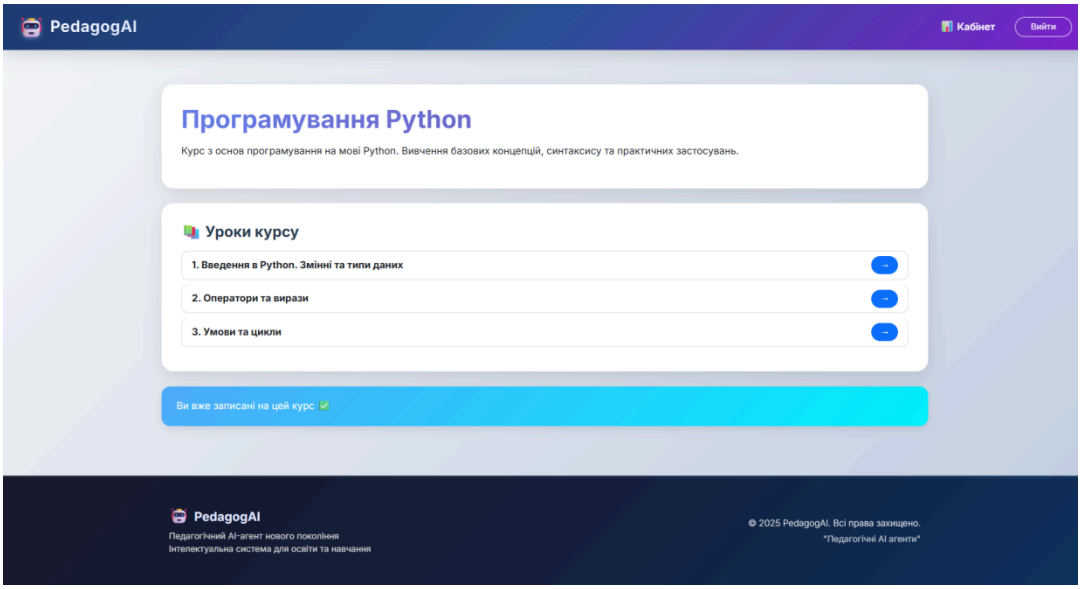
Діаграма критичних точок логування

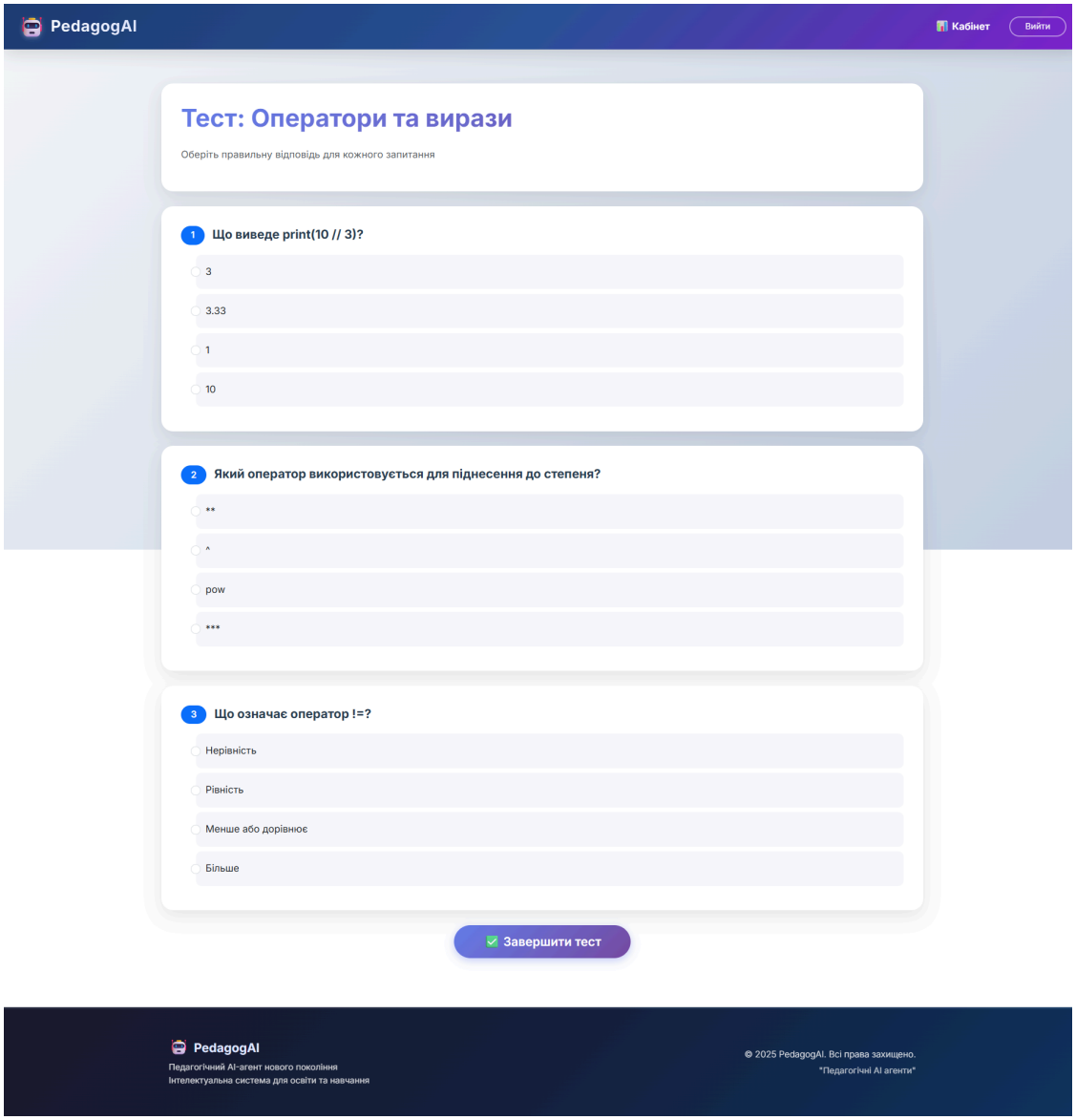


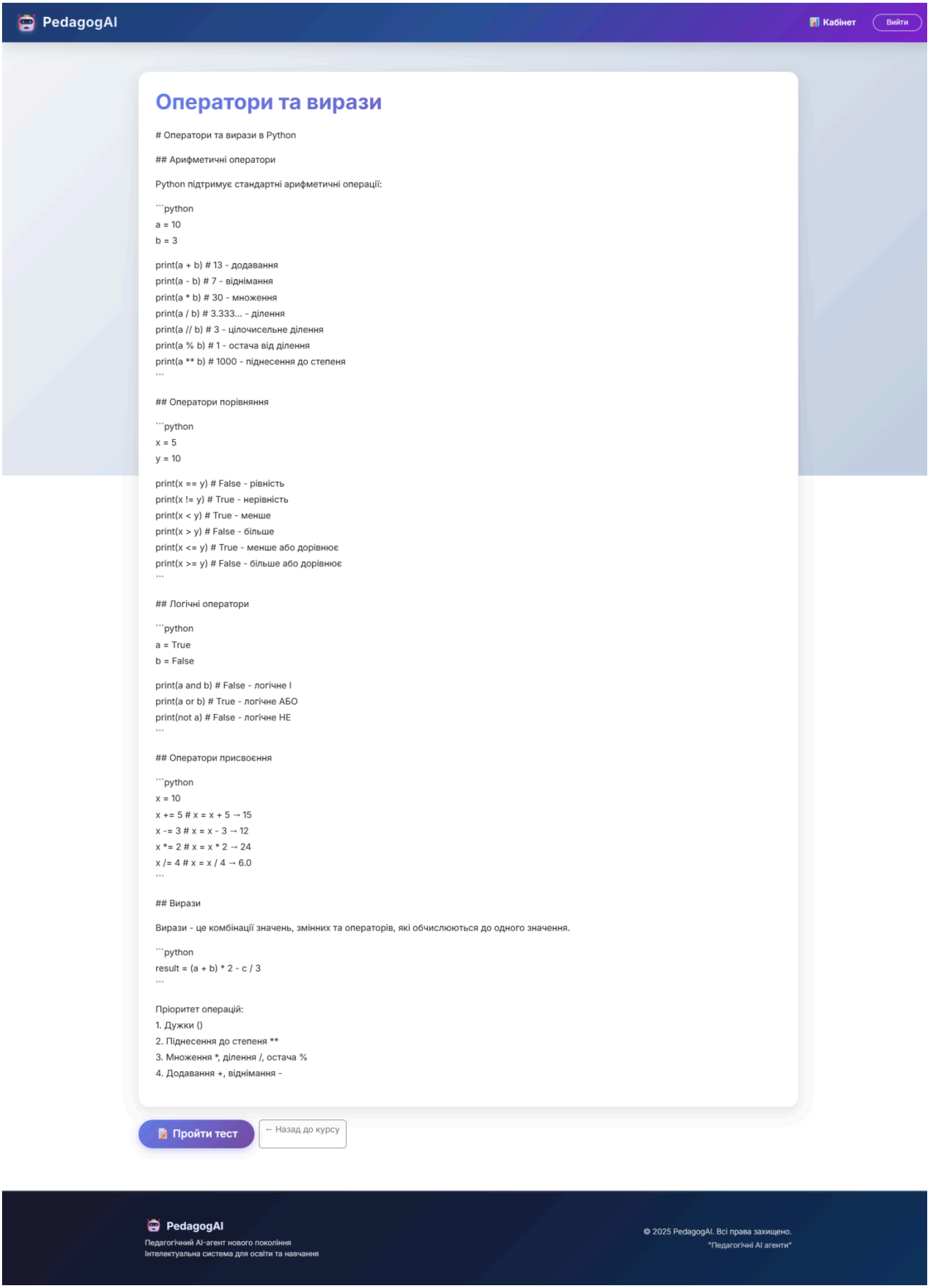
Додаток Е

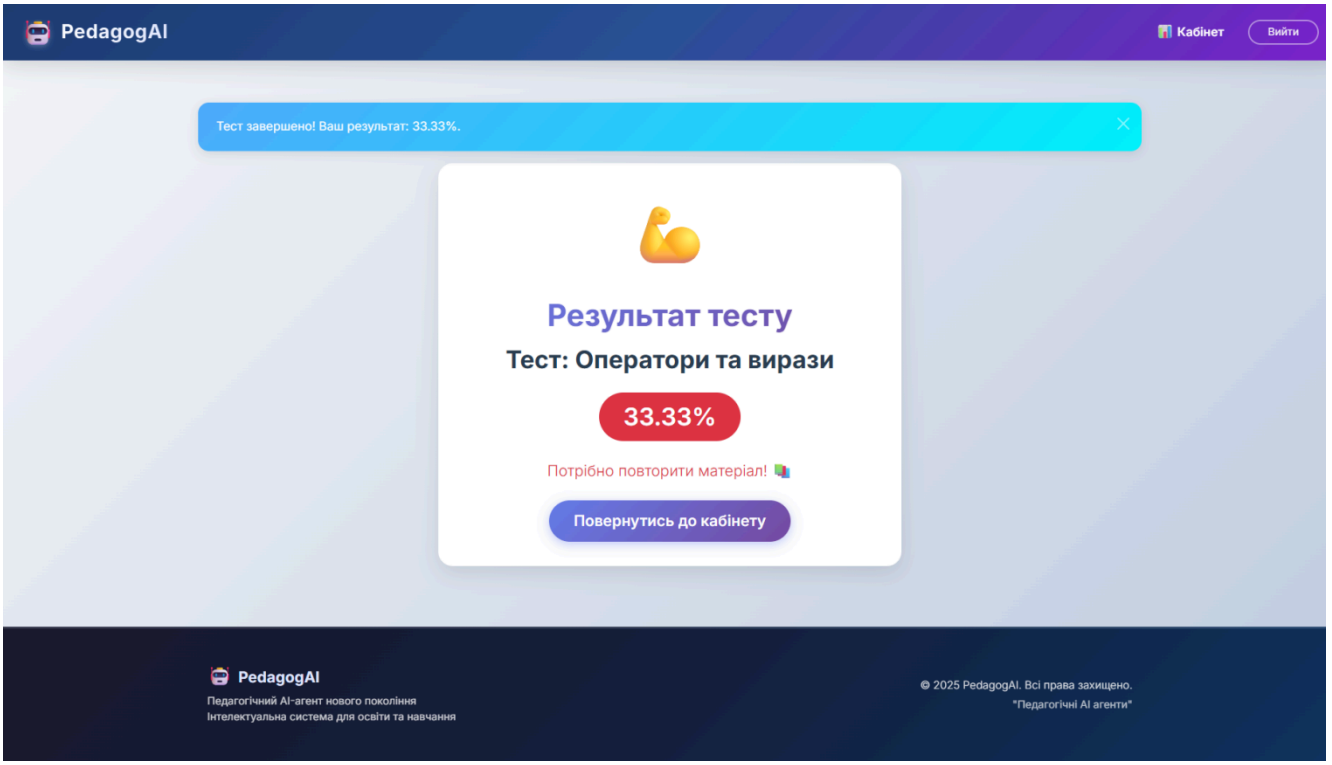
Скріншоти інтерфейсу додатку




PedagogAI
Кабінет
Вийти
Програмування Python
Курс з основ програмування на мові Python. Вивчення базових концепцій, синтаксису та практичних застосувань.
Уроки курсу
1. Введення в Python. Змінні та типи даних
2. Оператори та вирази
3. Умови та цикли
Ви вже записані на цей курс
PedagogAI
Педагогічний AI-агент нового покоління
Інтелектуальна система для освіти та навчання
© 2025 PedagogAI. Всі права захищено.
"Педагогічні AI агенти"


PedagogAI
Кабінет
Вийти
Тест: Оператори та вирази
Оберіть правильну відповідь для кожного запитання
1
Що виведе print(10 // 3)?
3
3.33
1
10
2
Який оператор використовується для піднесення до степеня?
**
^
pow
***
3
Що означає оператор !=?
Нерівність
Рівність
Менше або дорівнює
Більше
Завершити тест
PedagogAI
Педагогічний AI-агент нового покоління
Інтелектуальна система для освіти та навчання
© 2025 PedagogAI. Всі права захищено.
"Педагогічні AI агенти"


PedagogAI
Кабінет
Вийти
Операції та вирази
Операції та вирази в Python
Арифметичні операції
Python підтримує стандартні арифметичні операції:
```python
a = 10
b = 3
print(a + b) # 13 - додавання
print(a - b) # 7 - віднімання
print(a * b) # 30 - множення
print(a / b) # 3.333... - ділення
print(a // b) # 3 - цілочисельне ділення
print(a % b) # 1 - остача від ділення
print(a ** b) # 1000 - піднесення до степеня
```
Операції порівняння
```python
x = 5
y = 10
print(x == y) # False - рівність
print(x != y) # True - нерівність
print(x < y) # True - менше
print(x > y) # False - більше
print(x <= y) # True - менше або дорівнює
print(x >= y) # False - більше або дорівнює
```
Логічні операції
```python
a = True
b = False
print(a and b) # False - логічне І
print(a or b) # True - логічне АБО
print(not a) # False - логічне НЕ
```
Операції присвоєння
```python
x = 10
x += 5 # x = x + 5 → 15
x -= 3 # x = x - 3 → 12
x *= 2 # x = x * 2 → 24
x /= 4 # x = x / 4 → 6.0
```
Вирази
Вирази - це комбінації значень, змінних та операторів, які обчислюються до одного значення.
```python
result = (a + b) * 2 - c / 3
```
Пріоритет операцій:
1. Дужки ()
2. Піднесення до степеня **
3. Множення *, ділення /, остача %
4. Додавання +, віднімання -
Пройти тест
← Назад до курсу
PedagogAI
Педагогічний AI-агент нового покоління
Інтелектуальна система для освіти та навчання
© 2025 PedagogAI. Всі права захищено.
"Педагогічні AI агенти"



Додатковий приклад

Додатковий приклад

Давай розглянемо ще один приклад, щоб ти краще зрозумів, чому `10 // 3` виводить саме `3`.

Приклад: Розрахунок цілих часток

Ми маємо дві коробки, одна з яких містить 10 апельсинів, а інша містить 3 апельсини. Ти хочеш знати, скільки апельсинів можна розділити між двома людьми рівномірно.

У цьому випадку ти не зацікавлений у дробовій частці, яка залишиться після розподілу апельсинів. Ти хочеш лише знати, скільки апельсинів можна розділити цілком.

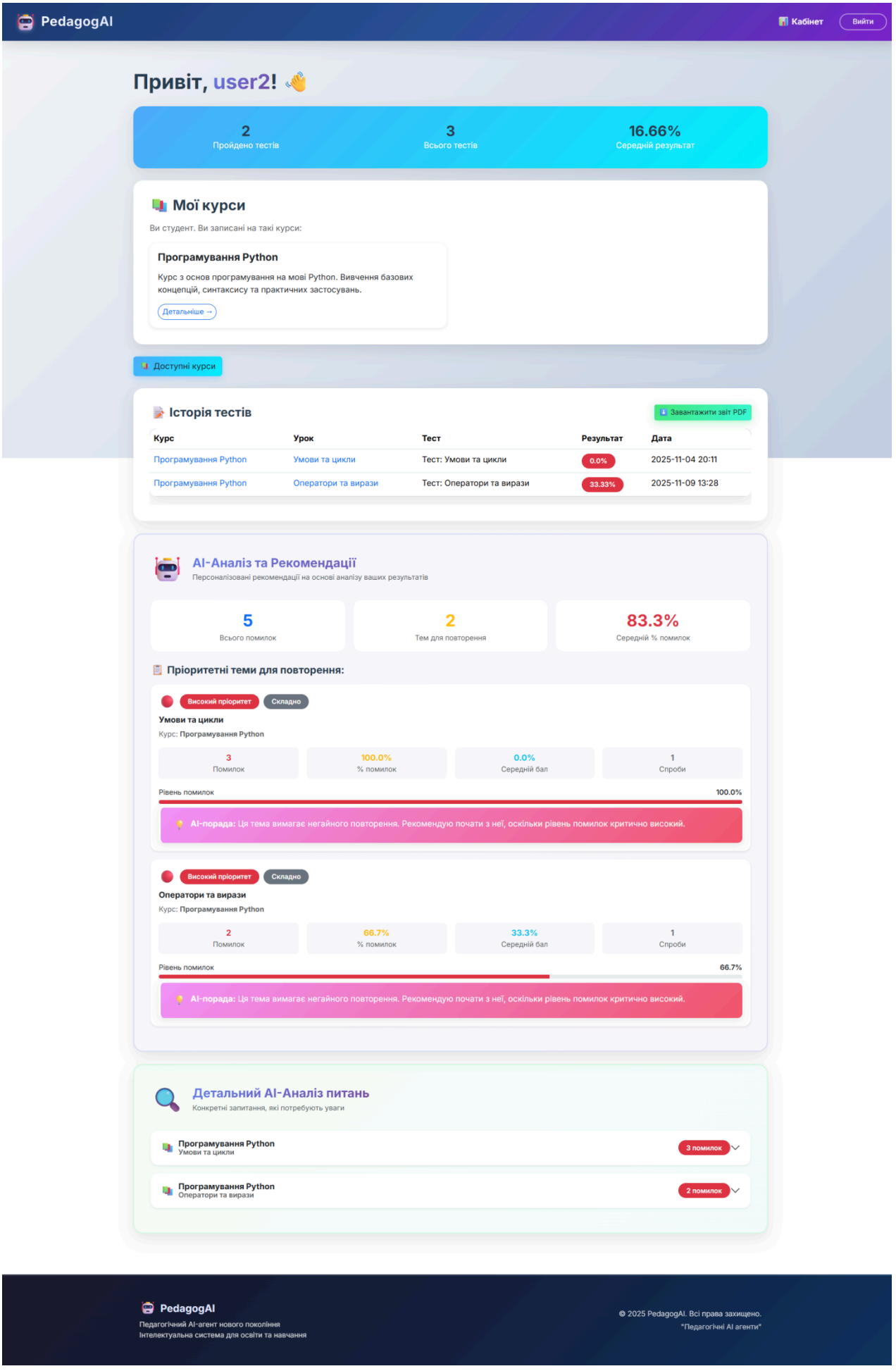
Для цього ти використовуєш операцію цілочисельного ділення `//`, яка повертає лише ціле часткове значення.

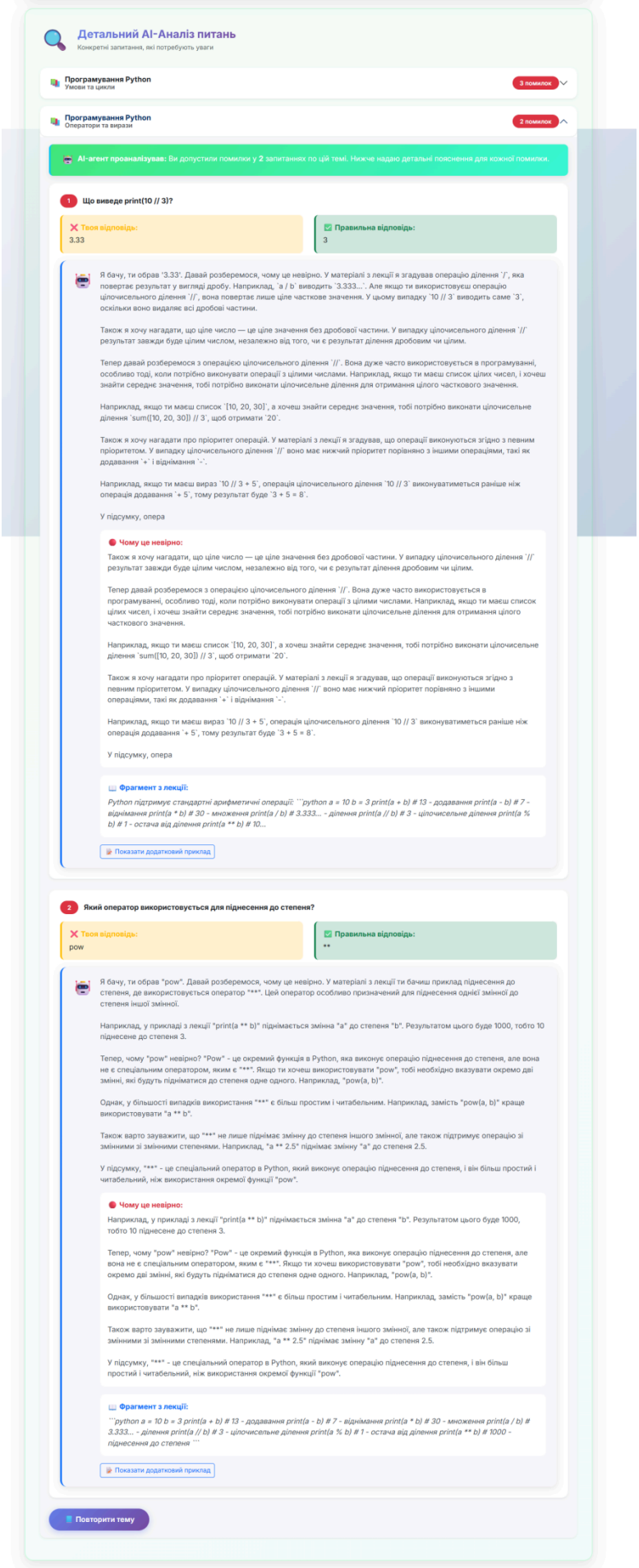
Код Python

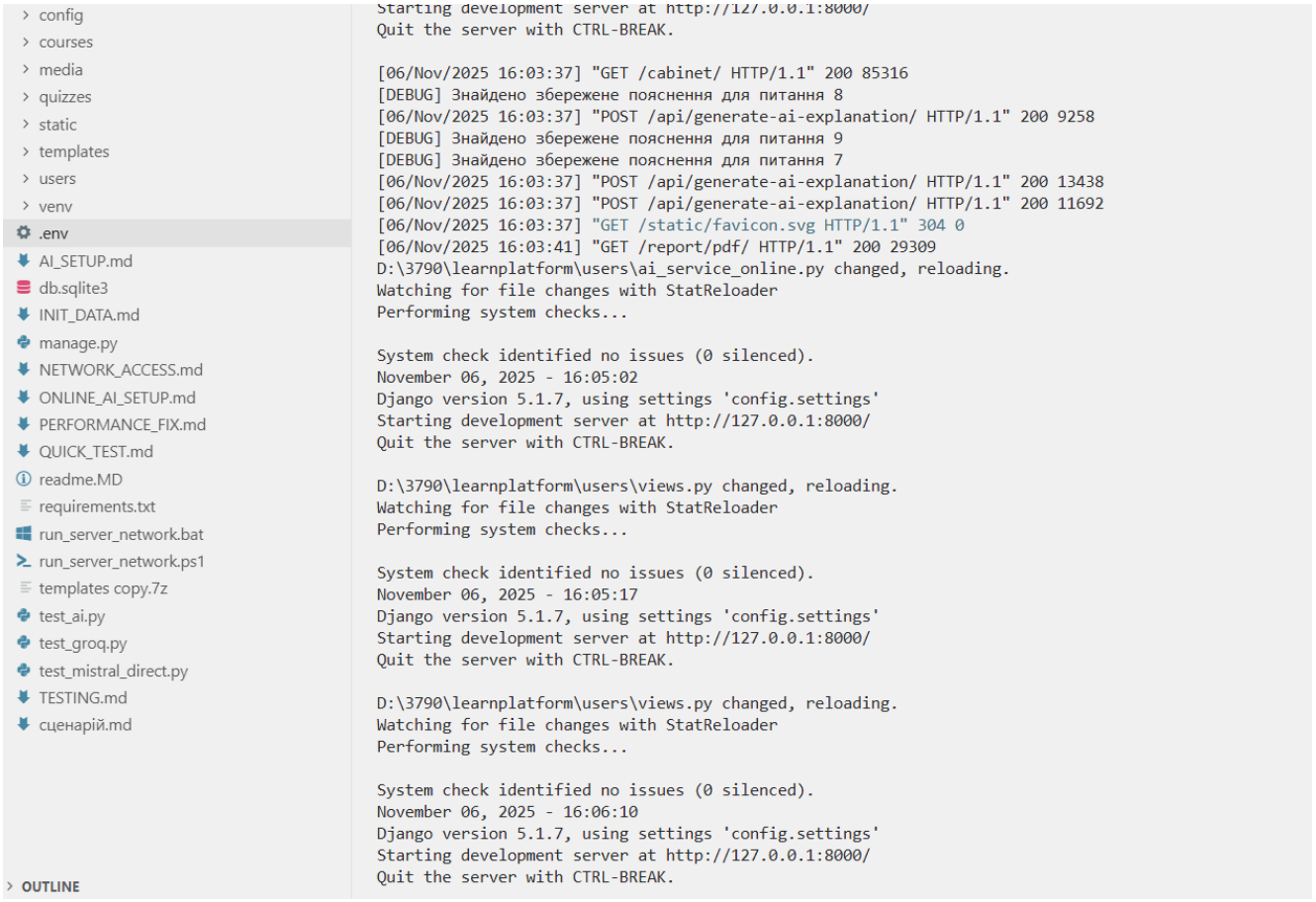
```python
# Визначення змінних
апельсини_у_першій_коробці = 10
апельсини_у_другій_коробці = 3

# Використання операції цілочисельного ділення
апельсини_на_одну_особу = апельсини_у_першій_коробці // апельсини_у_другій_коробці

# Виведення результату
print(апельсини_на_одну_особу) # Виводить 3
```


PedagogAI
Кабінет
Вийти
Привіт, user2! 👋
2
Пройдено тестів
3
Всього тестів
16.66%
Середній результат
Мої курси
Ви студент. Ви записані на такі курси:
Програмування Python
Курс з основ програмування на мові Python. Вивчення базових концепцій, синтаксису та практичних застосувань.
Детальніше →
Доступні курси
Історія тестів
Завантажити звіт PDF
Курс
Урок
Тест
Результат
Дата
Програмування Python
Умови та цикли
Тест: Умови та цикли
0.0%
2025-11-04 20:11
Програмування Python
Оператори та вирази
Тест: Оператори та вирази
33.33%
2025-11-09 13:28
AI-Аналіз та Рекомендації
Персоналізовані рекомендації на основі аналізу ваших результатів
5
Всього помилок
2
Тем для повторення
83.3%
Середній % помилок
Пріоритетні теми для повторення:
Високий пріоритет
Складно
Умови та цикли
Курс: Програмування Python
3
Помилок
100.0%
% помилок
0.0%
Середній бал
1
Спроби
Рівень помилок
100.0%
AI-порада: Ця тема вимагає негайного повторення. Рекомендую почати з неї, оскільки рівень помилок критично високий.
Високий пріоритет
Складно
Оператори та вирази
Курс: Програмування Python
2
Помилок
66.7%
% помилок
33.3%
Середній бал
1
Спроби
Рівень помилок
66.7%
AI-порада: Ця тема вимагає негайного повторення. Рекомендую почати з неї, оскільки рівень помилок критично високий.
Детальний AI-Аналіз питань
Конкретні запитання, які потребують уваги
Програмування Python
Умови та цикли
3 помилок
Програмування Python
Оператори та вирази
2 помилок
PedagogAI
Педагогічний AI-агент нового покоління
Інтелектуальна система для освіти та навчання
© 2025 PedagogAI. Всі права захищено.
"Педагогічні AI агенти"


Детальний AI-Аналіз питань
Конкретні запитання, які потребують уваги
Програмування Python
Умови та цикли
3 помилок
Програмування Python
Оператори та вирази
2 помилок
AI-агент проаналізував: Ви допустили помилки у 2 запитаннях по цій темі. Нижче надаю детальні пояснення для кожної помилки.
1 Що виведе print(10 // 3)?
Твоя відповідь:
3.33
Правильна відповідь:
3
Я бачу, ти обрав '3.33'. Давай розберемося, чому це невірно. У матеріалі з лекції я згадував операцію ділення `/`, яка повертає результат у вигляді дробу. Наприклад, `a / b` виводить '3.333...'. Але якщо ти використовуєш операцію цілочисельного ділення `//`, вона повертає лише ціле часткове значення. У цьому випадку `10 // 3` виводить саме `3`, оскільки воно видаляє всі дробові частини.
Також я хочу нагадати, що ціле число — це ціле значення без дробової частини. У випадку цілочисельного ділення `//` результат завжди буде цілим числом, незалежно від того, чи є результат ділення дробовим чи цілим.
Тепер давай розберемося з операцією цілочисельного ділення `//`. Вона дуже часто використовується в програмуванні, особливо тоді, коли потрібно виконувати операції з цілими числами. Наприклад, якщо ти маєш список цілих чисел, і хочеш знайти середнє значення, тобі потрібно виконати цілочисельне ділення для отримання цілого часткового значення.
Наприклад, якщо ти маєш список `[10, 20, 30]`, а хочеш знайти середнє значення, тобі потрібно виконати цілочисельне ділення `sum([10, 20, 30]) // 3`, щоб отримати `20`.
Також я хочу нагадати про пріоритет операцій. У матеріалі з лекції я згадував, що операції виконуються згідно з певним пріоритетом. У випадку цілочисельного ділення `//` воно має нижчий пріоритет порівняно з іншими операціями, такі як додавання `+` і віднімання `-`.
Наприклад, якщо ти маєш вираз `10 // 3 + 5`, операція цілочисельного ділення `10 // 3` виконуватиметься раніше ніж операція додавання `+ 5`, тому результат буде `3 + 5 = 8`.
У підсумку, опера
Чому це невірно:
Також я хочу нагадати, що ціле число — це ціле значення без дробової частини. У випадку цілочисельного ділення `//` результат завжди буде цілим числом, незалежно від того, чи є результат ділення дробовим чи цілим.
Тепер давай розберемося з операцією цілочисельного ділення `//`. Вона дуже часто використовується в програмуванні, особливо тоді, коли потрібно виконувати операції з цілими числами. Наприклад, якщо ти маєш список цілих чисел, і хочеш знайти середнє значення, тобі потрібно виконати цілочисельне ділення для отримання цілого часткового значення.
Наприклад, якщо ти маєш список `[10, 20, 30]`, а хочеш знайти середнє значення, тобі потрібно виконати цілочисельне ділення `sum([10, 20, 30]) // 3`, щоб отримати `20`.
Також я хочу нагадати про пріоритет операцій. У матеріалі з лекції я згадував, що операції виконуються згідно з певним пріоритетом. У випадку цілочисельного ділення `//` воно має нижчий пріоритет порівняно з іншими операціями, такі як додавання `+` і віднімання `-`.
Наприклад, якщо ти маєш вираз `10 // 3 + 5`, операція цілочисельного ділення `10 // 3` виконуватиметься раніше ніж операція додавання `+ 5`, тому результат буде `3 + 5 = 8`.
У підсумку, опера
Фрагмент з лекції:
Python підтримує стандартні арифметичні операції: ```python a = 10 b = 3 print(a + b) # 13 - додавання print(a - b) # 7 - віднімання print(a * b) # 30 - множення print(a / b) # 3.333... - ділення print(a // b) # 3 - цілочисельне ділення print(a % b) # 1 - остача від ділення print(a ** b) # 10...
Показати додатковий приклад
2 Який оператор використовується для піднесення до степеня?
Твоя відповідь:
pow
Правильна відповідь:
**
Я бачу, ти обрав "pow". Давай розберемося, чому це невірно. У матеріалі з лекції ти бачиш приклад піднесення до степеня, де використовується оператор "**". Цей оператор особливо призначений для піднесення однієї змінної до степеня іншої змінної.
Наприклад, у прикладі з лекції "print(a ** b)" піднімається змінна "a" до степеня "b". Результатом цього буде 1000, тобто 10 піднесене до степеня 3.
Тепер, чому "pow" невірно? "Pow" - це окремий функція в Python, яка виконує операцію піднесення до степеня, але вона не є спеціальним оператором, яким є "**". Якщо ти хочеш використовувати "pow", тобі необхідно вказувати окремо дві змінні, які будуть підніматися до степеня одне одного. Наприклад, "pow(a, b)".
Однак, у більшості випадків використання "**" є більш простим і читабельним. Наприклад, замість "pow(a, b)" краще використовувати "a ** b".
Також варто зауважити, що "**" не лише піднімає змінну до степеня іншого змінної, але також підтримує операцію зі змінними зі змінними степенями. Наприклад, "a ** 2.5" піднімає змінну "a" до степеня 2.5.
У підсумку, "**" - це спеціальний оператор в Python, який виконує операцію піднесення до степеня, і він більш простий і читабельний, ніж використання окремої функції "pow".
Чому це невірно:
Наприклад, у прикладі з лекції "print(a ** b)" піднімається змінна "a" до степеня "b". Результатом цього буде 1000, тобто 10 піднесене до степеня 3.
Тепер, чому "pow" невірно? "Pow" - це окремий функція в Python, яка виконує операцію піднесення до степеня, але вона не є спеціальним оператором, яким є "**". Якщо ти хочеш використовувати "pow", тобі необхідно вказувати окремо дві змінні, які будуть підніматися до степеня одне одного. Наприклад, "pow(a, b)".
Однак, у більшості випадків використання "**" є більш простим і читабельним. Наприклад, замість "pow(a, b)" краще використовувати "a ** b".
Також варто зауважити, що "**" не лише піднімає змінну до степеня іншого змінної, але також підтримує операцію зі змінними зі змінними степенями. Наприклад, "a ** 2.5" піднімає змінну "a" до степеня 2.5.
У підсумку, "**" - це спеціальний оператор в Python, який виконує операцію піднесення до степеня, і він більш простий і читабельний, ніж використання окремої функції "pow".
Фрагмент з лекції:
```python a = 10 b = 3 print(a + b) # 13 - додавання print(a - b) # 7 - віднімання print(a * b) # 30 - множення print(a / b) # 3.333... - ділення print(a // b) # 3 - цілочисельне ділення print(a % b) # 1 - остача від ділення print(a ** b) # 1000 - піднесення до степеня ```
Показати додатковий приклад
Повторити тему
```


config
courses
media
quizzes
static
templates
users
venv
.env
AI_SETUP.md
db.sqlite3
INIT_DATA.md
manage.py
NETWORK_ACCESS.md
ONLINE_AI_SETUP.md
PERFORMANCE_FIX.md
QUICK_TEST.md
readme.MD
requirements.txt
run_server_network.bat
run_server_network.ps1
templates copy.7z
test_ai.py
test_groq.py
test_mistral_direct.py
TESTING.md
сценарій.md
OUTLINE
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[06/Nov/2025 16:03:37] "GET /cabinet/ HTTP/1.1" 200 85316
[DEBUG] Знайдено збережене пояснення для питання 8
[06/Nov/2025 16:03:37] "POST /api/generate-ai-explanation/ HTTP/1.1" 200 9258
[DEBUG] Знайдено збережене пояснення для питання 9
[DEBUG] Знайдено збережене пояснення для питання 7
[06/Nov/2025 16:03:37] "POST /api/generate-ai-explanation/ HTTP/1.1" 200 13438
[06/Nov/2025 16:03:37] "POST /api/generate-ai-explanation/ HTTP/1.1" 200 11692
[06/Nov/2025 16:03:37] "GET /static/favicon.svg HTTP/1.1" 304 0
[06/Nov/2025 16:03:41] "GET /report/pdf/ HTTP/1.1" 200 29309
D:\3790\learnplatform\users\ai_service_online.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
November 06, 2025 - 16:05:02
Django version 5.1.7, using settings 'config.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
D:\3790\learnplatform\users\views.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
November 06, 2025 - 16:05:17
Django version 5.1.7, using settings 'config.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
D:\3790\learnplatform\users\views.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
November 06, 2025 - 16:06:10
Django version 5.1.7, using settings 'config.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.