Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Автоматизована генерація тестових завдань на основі текстових навчальних матеріалів засобами LLM
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-3
Тетяна Гуржи
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ст. викл. Олена Гриньова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гуржи Тетяні Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Автоматизована генерація тестових завдань на основі текстових навчальних матеріалів засобами LLM

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.

3. Вихідні дані до роботи Науково-технічні публікації, навчальні текстові матеріали, технічна документація OpenAI API, документація ChatGPT, офіційна документація .NET (ASP.NET Core), документація Entity Framework, документація з розробки на JavaScript та React, інтернет-ресурси OpenAI та Microsoft.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Постановка задачі
3) Проєктування та програмна реалізація інформаційної системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	19.05.2025 – 22.05.2025	виконано
3	Актуальність теми	23.5.2025	виконано
4	Мета роботи та вимоги до системи	26.05.2025	виконано
5	Огляд існуючих рішень	27.05.2025	виконано
6	Опис програмної реалізації	28.05.2025 – 29.05.2025	виконано
7	Оформлення пояснювальної записки	29.05.2025 – 31.05.2025	виконано
8	Підготовка до захисту	01.06.2025 – 16.06.2025	виконано
9	Захист перед ЕК	17.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач  _____
(підпис)

Керівник роботи _____ ст. викл. Олена Гриньова
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 63 с., 35 рис., 1 дод., 14 джерел.

ОСВІТА, ТЕСТИ, ШТУЧНИЙ ІНТЕЛЕКТ, ENTITY FRAMEWORK CORE, LLM, MICROSOFT SQL SERVER.

Об’єкт дослідження – використання великих мовних моделей для автоматизації створення тестових завдань з метою підвищення якості контролю знань та оптимізації навчального процесу.

Предмет дослідження – система автоматизованого створення тестових завдань на основі навчальних матеріалів із використанням великих мовних моделей, яка забезпечує адаптивне оцінювання знань студентів.

Мета роботи – розробка та дослідження програмної системи автоматизованого створення тестових завдань на основі навчальних матеріалів із використанням великих мовних моделей, що дозволить підвищити ефективність освітнього процесу та забезпечити більш об’єктивне оцінювання знань студентів.

Методи дослідження – аналіз наукових джерел з питань застосування штучного інтелекту в освіті, дослідження принципів роботи великих мовних моделей, проєктування архітектури інформаційної системи, моделювання структури бази даних, реалізація вебзастосунку із використанням сучасних фреймворків, інтеграція мовної моделі через API для генерації навчального контенту.

Розглянуто питання підвищення ефективності контролю знань студентів шляхом автоматизації процесу створення тестових завдань із використанням LLM. Запропоновано архітектуру системи, яка дозволяє автоматично аналізувати навчальні матеріали, генерувати різні типи тестових запитань із варіантами відповідей та глосарій основних термінів.

ABSTRACT

Bachelor’s thesis contains: 63 pp., 35 fig., 1 ann., 14 references.

EDUCATION, TESTING, ARTIFICIAL INTELLIGENCE, ENTITY FRAMEWORK CORE, LLM, MICROSOFT SQL SERVER.

Object of the Research – the application of large language models for automating the creation of test tasks with the goal of improving the quality of knowledge assessment and optimizing the educational process.

Subject of the Research – a system for the automated generation of test tasks based on educational materials using large language models, which provides adaptive assessment of students' knowledge.

Purpose of the Work – to develop and study a software system for the automated generation of test tasks based on educational materials using large language models, aimed at increasing the efficiency of the educational process and ensuring a more objective assessment of students' knowledge.

Research Methods – analysis of scientific literature on the application of artificial intelligence in education; study of the principles behind large language models; design of the information system architecture; modeling of the database structure; implementation of a web application using modern frameworks; and integration of a language model via API to generate educational content.

This work addresses the issue of enhancing the effectiveness of knowledge assessment by automating the process of creating test tasks using large language models (LLMs). It proposes a system architecture that enables the automatic analysis of educational materials, generation of various types of test questions with answer options, and compilation of a glossary of key terms.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Аналіз предметної галузі 10
1.1 Актуальність теми 10
1.2 Огляд існуючих рішень 12
2 Постановка задачі 15
2.1 Мета роботи 15
2.2 Вимоги до системи 15
2.3 Постановка задачі 17
2.4 Архітектура системи та обґрунтування вибору технологій 18
3 Проєктування та програмна реалізація інформаційної системи 21
3.1 Трирівнева клієнт-серверна архітектура системи 21
3.2 Схема бази даних 22
3.3 Використані технології та компоненти системи 24
3.3.1 Реалізація клієнтської частини 24
3.3.2 Реалізація серверної частини 29
3.4 Генерація дистракторів на основі мовної моделі 35
3.5 Реалізація генерації тестів і глосаріїв 37
3.6 Інтерфейс користувача та система ролей 42
3.7 Тестування системи 53
Висновки 59
Перелік джерел посилання 61
Додаток А Відомість кваліфікаційної роботи 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – інтерфейс програмування додатків;

ASP – Active Server Pages – активні серверні сторінки;

DTO – Data Transfer Object – об'єкт передавання даних;

EF – Entity Framework – фреймворк сутностей;

HTTP – HyperText Transfer Protocol – протокол передавання гіпертексту;

LLM – Large Language Model – велика мовна модель;

ORM – Object-Relational Mapping – об'єктно-реляційне відображення;

REST – Representational State Transfer – передавання стану представлення;

SQL – Structured Query Language – мова структурованих запитів;

XML – eXtensible Markup Language – розширювана мова розмітки.

ВСТУП

Сучасна освіта переживає глибоку трансформацію під впливом цифрових технологій, які змінюють не лише способи подачі навчального матеріалу, а й підходи до оцінювання знань. Тестування залишається одним із найефективніших інструментів для визначення рівня засвоєння інформації, виявлення прогалин у знаннях та забезпечення зворотного зв’язку для студентів і викладачів. Проте створення якісних тестових завдань у традиційний спосіб вимагає значних часових і інтелектуальних ресурсів. Особливо це відчутно в умовах масового чи дистанційного навчання, коли індивідуальна взаємодія з кожним студентом обмежена, а обсяги навчального контенту постійно зростають.

В останні роки значного розвитку зазнали технології штучного інтелекту, зокрема великі мовні моделі (Large Language Models, LLM), які здатні працювати з великими обсягами тексту, аналізувати контекст, виявляти ключові поняття та формулювати логічно структуровані запитання. Це відкриває нові можливості для автоматизації освітніх процесів, зокрема в галузі тестування. Використання LLM дозволяє автоматично генерувати тестові завдання на основі навчальних матеріалів, що суттєво зменшує навантаження на викладачів і підвищує об’єктивність оцінювання знань.

У сучасних умовах, коли інформаційні технології стрімко розвиваються, а обсяг навчальних матеріалів постійно зростає, забезпечення якісного та ефективного контролю знань стає особливо актуальним. Традиційні методи створення тестів не завжди встигають адаптуватися до змін у навчальних програмах та підвищених вимог до освітнього процесу. Автоматизація цього процесу за допомогою LLM дозволяє не лише скоротити час на підготовку тестів, а й враховувати індивідуальні особливості студентів, адаптуючи завдання до їхнього рівня знань і темпу

навчання. Це сприяє підвищенню справедливості оцінювання та мотивації до глибшого засвоєння матеріалу.

Важливою перевагою автоматизованої генерації тестів є можливість масштабування освітніх платформ, зменшення залежності від людського фактора та мінімізація суб’єктивного впливу на зміст і структуру тестових завдань. Це особливо актуально для масових відкритих онлайн-курсів, де кількість учасників може сягати десятків тисяч. Водночас впровадження подібних рішень потребує ретельного аналізу й контролю якості: автоматично згенеровані завдання мають відповідати педагогічним стандартам, бути точними, зрозумілими та не призводити до спрощення змісту чи надмірної стандартизації оцінювання.

Дослідження можливостей використання великих мовних моделей для автоматизованої генерації тестів є кроком до глибшої інтеграції штучного інтелекту в освіту. Такий підхід дозволяє не лише полегшити роботу викладача, а й підвищити якість освітнього процесу загалом. Запропоновані в межах цієї роботи рішення можуть стати основою для створення нових інструментів оцінювання та сприяти впровадженню інновацій у навчанні, що особливо важливо для підготовки конкурентоспроможних фахівців у цифрову епоху.

Таким чином, впровадження автоматизованих систем генерації тестових завдань на основі LLM відкриває нові перспективи для розвитку освітніх технологій. Такий підхід поєднує ефективність, масштабованість і гнучкість сучасних цифрових рішень із високими стандартами якості контролю знань. Реалізація подібних систем сприятиме модернізації освітнього процесу, підвищенню його об’єктивності, індивідуалізації навчання та розвитку самостійності студентів у процесі здобуття знань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Актуальність теми

Цифрові технології дедалі глибше інтегруються в усі сфери суспільного життя, і система освіти в цьому процесі посідає особливе місце. Сучасне навчальне середовище змінюється не тільки завдяки новим формам подачі інформації, а й завдяки трансформації підходів до контролю знань. Оцінювання перестає бути виключно формальністю – воно стає інструментом аналітики, корекції навчального процесу й підтримки індивідуальної траєкторії розвитку студента. В умовах швидкого зростання обсягів навчальної інформації, збільшення кількості студентів та переходу до змішаних і дистанційних форматів навчання актуальним стає пошук гнучких і масштабованих рішень для об’єктивного оцінювання. Одним із найуживаніших інструментів залишається тестування, яке дозволяє порівняно швидко, в уніфікованій формі визначити рівень засвоєння навчального матеріалу.

Попри це, створення ефективних тестових завдань є складним і трудомістким процесом. Для підготовки повноцінного тесту викладач має не лише проаналізувати зміст навчального матеріалу, а й сформулювати запитання так, щоб вони були зрозумілими, точними та відповідали цільовому рівню складності. Крім того, необхідно підібрати дистрактори – варіанти відповідей, які мають бути логічними, але водночас неправильними, щоб уникнути очевидності правильного вибору. Такий підхід вимагає часу, уваги до деталей і часто потребує кількох ітерацій перевірки та редагування. Якщо врахувати необхідність регулярного оновлення тестів у зв’язку зі змінами в навчальному матеріалі, стає зрозумілим, чому це завдання настільки обтяжливе. У масштабах великої кількості студентів навантаження на викладача зростає в рази, що нерідко

призводить до використання шаблонних завдань або застарілих варіантів тестів без глибокої адаптації.

Додатковий виклик постав перед системою освіти в період пандемії COVID-19, коли освітній процес масово перейшов у дистанційний формат. Це змусило навчальні заклади терміново впроваджувати засоби дистанційного контролю знань. Саме тоді автоматизовані системи тестування почали відігравати особливо важливу роль. Однак значна частина таких систем виявилася обмеженою у функціональності: вони здебільшого не мали можливості автоматично генерувати завдання на основі наданих текстів або навчальних матеріалів. Через це викладачам доводилося створювати тести вручну або адаптувати зовнішні платформи під конкретні потреби, що вимагало додаткових зусиль і ресурсів [1].

У цьому контексті цілком логічним стало зростання інтересу до застосування технологій штучного інтелекту в освіті, зокрема до великих мовних моделей (LLM). За останні роки такі моделі значно покращили здатність обробляти природну мову, аналізувати великі обсяги інформації, розуміти контекст і генерувати змістовно релевантні тексти. Моделі типу GPT4 [2], Perplexity Sonar [3] та інші демонструють здатність формулювати логічні запитання, пояснювати складні теми простими словами та знаходити зв’язки між різними фрагментами тексту. Це відкриває нові можливості для їх інтеграції в освітній процес – зокрема, для створення навчальних матеріалів, інтелектуальних помічників і, безумовно, генерації тестових завдань.

Застосування LLM для автоматизованого створення тестів має низку істотних переваг. По-перше, це дозволяє формувати унікальні тести, орієнтовані на конкретний навчальний контент, без потреби у ручній розробці кожного запитання. По-друге, такі моделі можуть класифікувати питання за тематикою та складністю, що дозволяє адаптувати тести до рівня підготовки конкретного студента чи групи. По-третє, штучний інтелект здатен створювати правдоподібні варіанти відповідей, що зменшує

ймовірність вгадування та підвищує об’єктивність оцінювання. До того ж кожному варіанту відповіді може бути присвоєно певну вагу залежно від його відповідності правильному рішенню, що дозволяє враховувати частково правильні відповіді і отримувати більш точні результати тестування.

Варто підкреслити, що автоматизація створення тестів не передбачає усунення ролі викладача. Навпаки, вона спрямована на зменшення рутинного навантаження, що дає можливість зосередитися на аналітичній, методичній роботі, інтерпретації результатів і наданні індивідуальної підтримки студентам. Зі свого боку, студенти отримують інструменти для самоперевірки, повторення матеріалу, а також доступ до автоматично сформованих глосаріїв із ключовими поняттями теми, що сприяє кращому засвоєнню інформації та підготовці до підсумкового контролю.

Таким чином, використання великих мовних моделей для автоматизованої генерації тестових завдань повністю відповідає сучасним потребам освіти [4]. Такий підхід поєднує ефективність, адаптивність і масштабованість цифрових рішень з високою якістю оцінювання знань. Актуальність теми цієї роботи зумовлена не лише технологічними можливостями ШІ, а й запитами освітньої практики на сучасні, гнучкі й зручні інструменти контролю знань.

1.2 Огляд існуючих рішень

Одним із найвідоміших і найчастіше використовуваних рішень для організації дистанційного навчання є система Moodle. Вона вже давно стала стандартом для багатьох освітніх закладів у світі, у тому числі й в Україні. Moodle дозволяє створювати курси, додавати тести з різними типами запитань, аналізувати результати, експортувати та імпортувати завдання, налаштовувати систему оцінювання тощо. Разом із тим, попри свою функціональність, створення тестів у Moodle відбувається виключно

вручну. Це потребує значного часу та зусиль з боку викладача. Крім того, система не пропонує інструментів автоматичної генерації запитань на основі навчального матеріалу, не має вбудованих засобів оцінювання складності завдань чи частково правильних відповідей.

Окрім Moodle, сьогодні існує чимало онлайн-сервісів, що намагаються автоматизувати процес створення тестів на основі текстів, документів або мультимедійного контенту. Наприклад, платформи Revisely і Quizgecko дозволяють генерувати тестові завдання з веб-сторінок, PDF файлів чи навіть відео. Вони підтримують різні формати запитань: з вибором варіанту, на встановлення відповідності, відкритого типу тощо. Ці сервіси можуть бути корисними для швидкої підготовки матеріалів. Проте, на жаль, більшість із них орієнтована виключно на англомовне середовище, а підтримка української мови відсутня або реалізована на базовому рівні. Це ускладнює їхнє практичне використання в українському освітньому середовищі.

Схожу функціональність пропонують платформи Fillout [5] та FlexiQuiz [6], які дозволяють створювати тести з інтерактивним інтерфейсом, налаштовувати структуру завдань, встановлювати таймери, кількість спроб і переглядати результати. У них зручний інтерфейс і достатньо можливостей для базового контролю знань. Втім, і тут проблема мови залишається актуальною: повної підтримки української, як правило, немає, або ж вона не адаптована до специфіки національної освітньої системи.

Інші інструменти, наприклад EasyTestMaker чи Canva Quiz Maker, орієнтовані на ручне створення тестів за допомогою готових шаблонів. Вони прості у використанні та підходять для одноразової підготовки завдань. Проте відсутність інтеграції з навчальними матеріалами та автоматизації обмежує їхню ефективність у випадках, коли потрібно регулярно оновлювати або формувати великі обсяги тестового контенту.

Окрім зазначених недоліків, більшість сучасних платформ не забезпечують гнучкого налаштування тестів відповідно до специфіки навчального курсу та рівня підготовки студентів. Часто відсутня можливість автоматичного аналізу навчального матеріалу для створення індивідуалізованих тестових завдань. Багато сервісів не інтегруються з електронними журналами чи іншими освітніми платформами, що ускладнює комплексний моніторинг успішності студентів. Деякі інструменти мають обмеження щодо обсягу завантаженого контенту або кількості тестів, які можна створити безкоштовно. Важливим аспектом є й захист персональних даних, що не завжди гарантовано на закордонних онлайн-платформах. Крім того, більшість сервісів не підтримують автоматичну генерацію дистракторів, які б ускладнювали процес вгадування відповідей. Відсутність глибокого аналізу складності питань та можливості формування тестів різних рівнів також знижує ефективність оцінювання. Викладачі змушені витрачати додатковий час на перевірку та редагування автоматично згенерованих тестів. Це ще раз підкреслює актуальність розробки національного інструменту, який би враховував потреби українських освітян та специфіку навчального процесу [7]. Таким чином, створення сучасної системи автоматизованої генерації тестових завдань із підтримкою української мови та адаптацією до різних дисциплін є важливим кроком для підвищення якості дистанційного навчання.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета роботи

У межах цієї кваліфікаційної роботи передбачається створення програмної системи, яка автоматизуватиме процес формування тестових завдань на основі навчальних матеріалів. Основна увага приділяється зниженню навантаження на викладача під час підготовки контрольних заходів, підвищенню якості та об’єктивності оцінювання знань студентів, а також забезпеченню адаптації складності завдань до різного рівня підготовки. Особливу увагу зосереджено на уникненні повторюваності запитань, уніфікації стилю та структури тестових матеріалів, а також підтримці викладача в умовах дистанційного навчання, коли необхідно швидко формувати значний обсяг матеріалів у різних форматах.

2.2 Вимоги до системи

Запропонована система має забезпечувати автоматичний аналіз навчального контенту з виокремленням ключових понять, генерацію тестових запитань на основі виявленої інформації, формування варіантів відповідей із можливістю визначення рівня складності кожного запитання. Одним із важливих функціональних елементів є присвоєння ваги кожному варіанту відповіді, що дозволяє коректно враховувати частково правильні відповіді під час підсумкового оцінювання. Крім того, система повинна автоматично формувати глосарій основних термінів теми, який може використовуватись як довідковий матеріал для студента під час навчання або повторення. Очікується, що вхідні дані система прийматиме у форматах .txt, .pdf, .docx і .pptx, а результати експортуватиме у структурованому вигляді, придатному для подальшого використання, зокрема в таких навчальних платформах, як Moodle. Також важливо передбачити

можливість зберігання згенерованих матеріалів безпосередньо на платформі для подальшого доступу, редагування або відкриття їх для інших користувачів у межах конкретного курсу.

До ключових функціональних можливостей системи слід віднести: імпорт навчальних матеріалів із поширених текстових і презентаційних джерел, автоматичне створення запитань різних рівнів складності, формування варіантів відповідей із розрахунком ваги кожного, створення термінологічного глосарію, а також збереження результатів у зручному форматі (наприклад, .txt або .xml) для подальшої роботи. Окрім цього, користувач повинен мати змогу попередньо переглянути й за потреби відредагувати згенерований тест перед його остаточним збереженням або експортом.

Щоб забезпечити індивідуалізовану роботу з платформою, необхідно реалізувати базову систему авторизації користувачів. Це дозволить створювати облікові записи, зберігати персональні результати, словники та списки згенерованих тестів для кожного окремого користувача. Під час реєстрації користувач вводить ім’я, прізвище та адресу електронної пошти, після чого отримує доступ до персонального кабінету. Система повинна підтримувати щонайменше дві ролі – викладача та студента. Такий підхід дозволить організувати освітній процес більш гнучко: викладач зможе створювати курси, додавати до них теми, завантажувати навчальні матеріали, генерувати тести та відкривати їх для учасників курсу. При цьому тести можна буде не тільки зберігати у вигляді файлів, а й залишати на платформі для подальшого проходження. Студенти, зі свого боку, матимуть можливість приєднуватися до курсів, проходити доступні тести, переглядати свої результати й формувати для себе тренувальні тести для самоперевірки на основі завантажених матеріалів. Це дозволяє поєднати як контрольовану взаємодію в межах курсу, так і індивідуальну підготовку в зручному для студента форматі.

Серед ключових вимог до системи є підтримка роботи з україномовними текстами, зрозумілий інтерфейс користувача, можливість налаштовувати параметри генерації та мінімальна залежність від сторонніх сервісів. Інтерфейс має бути інтуїтивним і доступним навіть для користувачів без технічної підготовки. У той самий час система повинна бути достатньо гнучкою, щоби дозволити вибір типу тесту чи рівня деталізації відповідно до потреб викладача. Окрему увагу слід приділити стабільності роботи програми при обробці матеріалів різного обсягу.

2.3 Постановка задачі

Для досягнення поставленної мети необхідно вирішити такі завдання:

– розробити алгоритм автоматичного аналізу навчального контенту для виокремлення ключових понять, термінів і основних ідей тексту;

– реалізувати механізм генерації тестових запитань різних типів (з вибором відповіді) на основі отриманої інформації;

– забезпечити формування варіантів відповідей із можливістю визначення їхньої складності та присвоєння вагових коефіцієнтів для обліку частково правильних відповідей;

– розробити функціонал автоматичного створення глосарію основних термінів теми для подальшого використання студентами;

– передбачити можливість імпорту навчальних матеріалів у поширених форматах (.txt, .pdf, .docx, .pptx) та експорту результатів у структурованому вигляді (наприклад, .xml) для інтеграції з освітніми платформами (наприклад, Moodle);

– реалізувати інтерфейс користувача, що дозволяє попередньо переглядати, редагувати та зберігати згенеровані тести й глосарії, забезпечити підтримку української мови та адаптацію системи до вимог національної освітньої програми;

– реалізувати зберігання згенерованих матеріалів на платформі для подальшого доступу та редагування.

Отже, поставлене завдання охоплює розробку інструменту, здатного автоматично перетворювати навчальні тексти на готові освітні елементи – тести та глосарії з можливістю їх редагування, збереження та використання в електронному навчальному середовищі. Передбачені функції системи узгоджуються з актуальними потребами сучасної освіти та створюють основу для ефективної реалізації сформульованої мети.

2.4 Архітектура системи та обґрунтування вибору технологій

У якості основної мови програмування для реалізації серверної частини платформи обрано C#. Цей вибір зумовлений тим, що мова є об’єктно-орієнтованою, що дозволяє створювати добре структурований, надійний і легко масштабований код. Вона має широку підтримку серед сучасних бібліотек і фреймворків, зокрема фреймворку .NET, який забезпечує високу продуктивність і стабільність роботи серверних застосунків. Завдяки цьому C# є ефективним інструментом для реалізації бізнес-логіки, обробки запитів користувачів, управління сесіями та взаємодії з базою даних.

Для розробки веб-додатку використовується ASP.NET Core [8] – сучасний фреймворк, який підтримує побудову архітектури MVC, REST API та забезпечує розробку продуктивних і масштабованих серверних рішень. Однією з важливих переваг ASP.NET Core є його кросплатформеність: фреймворк працює як у середовищі Windows, так і на Linux або macOS, що дозволяє обирати зручні умови для розгортання та подальшої підтримки застосунку.

Для роботи з базою даних обрано Entity Framework Core – технологію ORM, яка дозволяє здійснювати взаємодію з базами даних через об'єктно-орієнтовану модель [9]. Це суттєво спрощує роботу з даними і знижує

потребу в написанні SQL-запитів вручну. У межах цього проєкту планується використання Microsoft SQL Server як основної системи управління базами даних. SQL Server зарекомендував себе як надійне рішення для роботи з великими обсягами інформації, має розвинену підтримку транзакцій, потужні засоби захисту даних і високу продуктивність. У контексті даного проєкту він використовується для збереження інформації про користувачів, курси, теми, тести, глосарії термінів і результати проходження тестів.

Однією з ключових функцій платформи є автоматичне створення навчального контенту на основі завантажених текстів. Для реалізації цієї функціональності інтегрується зовнішній API мовної моделі ChatGPT від OpenAI, яка дозволяє автоматично генерувати тестові запитання, формувати варіанти відповідей, визначати рівень складності кожного запитання, присвоювати вагові коефіцієнти відповідям, а також будувати глосарії основних термінів. Запити до API обробляються серверною частиною платформи, що дає змогу централізовано керувати процесом генерації та обробки результатів.

Клієнтську частину застосунку реалізовано за допомогою React – сучасної JavaScript-бібліотеки, яка широко застосовується для створення односторінкових веб-додатків (SPA). Використання React дозволяє створити інтуїтивно зрозумілий та адаптивний інтерфейс користувача, що швидко реагує на дії без необхідності повного перезавантаження сторінки. Компонентна архітектура React забезпечує зручність супроводу проєкту, його масштабування та повторного використання частин інтерфейсу. Взаємодія з серверною частиною реалізується через REST API: клієнт надсилає запити, отримує відповіді, та відображає дані в інтерфейсі.

Таким чином, використання мови програмування C# у поєднанні з фреймворком ASP.NET Core, ORM-технологією Entity Framework Core та системою управління базами даних Microsoft SQL Server для реалізації серверної частини, а також React для побудови клієнтської частини, дає змогу створити сучасну, надійну та масштабовану програмну платформу.

Така архітектура забезпечує ефективну взаємодію між усіма компонентами системи, стабільну обробку даних і зручний користувацький інтерфейс (рисунок 2.1).

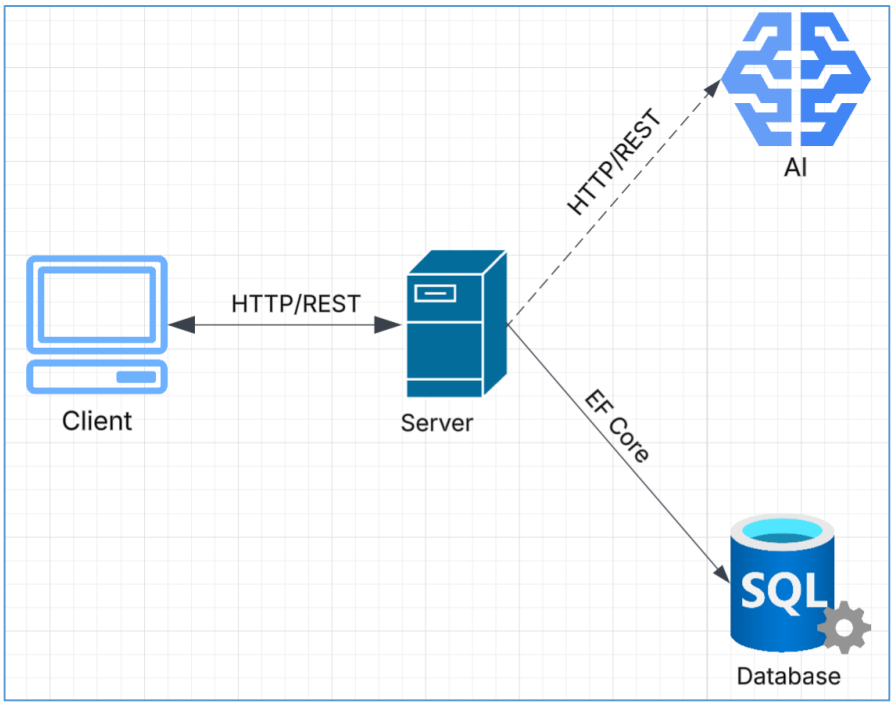

Рисунок 2.1 – Архітектура системи

Інтеграція з мовною моделлю ChatGPT дозволяє автоматизувати створення навчального контенту, що, у свою чергу, суттєво розширює функціональні можливості платформи та сприяє досягненню цілей.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Трирівнева клієнт-серверна архітектура системи

На початковому етапі проєктування було сформовано архітектурну модель системи, яка відповідає функціональним вимогам та передбачає можливість подальшого масштабування. З метою досягнення гнучкості, надійності та структурованості була обрана трирівнева клієнт-серверна архітектура, що включає презентаційний рівень (інтерфейс користувача), логічний рівень (серверну частину) та рівень зберігання даних (базу даних).

На рівні інтерфейсу реалізовано зручний вебзастосунок на основі бібліотеки React, який забезпечує взаємодію користувача з системою відповідно до його ролі – викладача або студента. Такий підхід дозволяє легко створювати, переглядати й редагувати навчальні курси, теми, тести та глосарії, а також переглядати результати тестування. React було обрано за його компонентну структуру, гнучкість та підтримку односторінкових застосунків, що дозволяє оновлювати вміст без перезавантаження сторінки.

Серверну частину побудовано з використанням C# і ASP.NET Core. Вона відповідає за виконання бізнес-логіки, керування користувачами, авторизацію й автентифікацію, обробку запитів до бази даних та взаємодію з зовнішнім API мовної моделі (наприклад, ChatGPT від OpenAI). Сервер обробляє запити від клієнта, передає їх у базу або до мовної моделі, а результати надсилає назад у зручному для відображення форматі.

Для обміну даними між клієнтом і сервером використовується REST API з форматом JSON. Така технологія забезпечує гнучкість у майбутній інтеграції з іншими сервісами, зокрема системами дистанційного навчання (наприклад, Moodle), і відкриває можливість підключення мобільних додатків.

Важливою складовою платформи є мовна модель, що здійснює аналіз навчального матеріалу та на його основі автоматично генерує тестові завдання, дистрактори (альтернативні, але правдоподібні відповіді) та глосарні статті. Зв’язок із ChatGPT або аналогічною LLM відбувається через HTTP-запити до відповідного API, після чого результати генерації проходять додаткову обробку відповідно до внутрішньої логіки системи – наприклад, за рівнем складності або ваговими коефіцієнтами.

Розроблена архітектура дозволяє чітко розмежувати обов’язки між компонентами системи, що полегшує її підтримку, подальший розвиток і забезпечує стабільну роботу при збільшенні кількості користувачів чи розширенні обсягу навчального контенту. Такий підхід дає змогу створити гнучку та сучасну освітню платформу, яка відповідає актуальним потребам цифрової трансформації освіти.

3.2 Схема бази даних

Важливим етапом у процесі розробки стало створення логічної моделі бази даних, яка забезпечує ефективне зберігання, доступ і обробку всієї інформації, необхідної для роботи ключових компонентів системи. Для реалізації було використано реляційну модель із дотриманням принципів нормалізації, що дозволило уникнути дублювання даних та забезпечити цілісність зв’язків між сутностями.

Центральне місце в структурі бази займає сутність User – користувач, який може виконувати роль викладача або студента. Користувачі взаємодіють із курсами, темами, тестами та результатами проходження, а для реалізації зв’язку «багато до багатьох» між користувачами та курсами використано проміжну таблицю CourseUser.

Кожен курс (Course) містить щонайменше одну тему (Topic), яка структурує навчальний матеріал. Теми є ключовими одиницями, до яких прив’язуються завантажені файли, згенеровані тести й термінологічні

словники. Така структура дозволяє впорядковувати контент та забезпечує точкову генерацію навчальних матеріалів (рисунок 3.1).

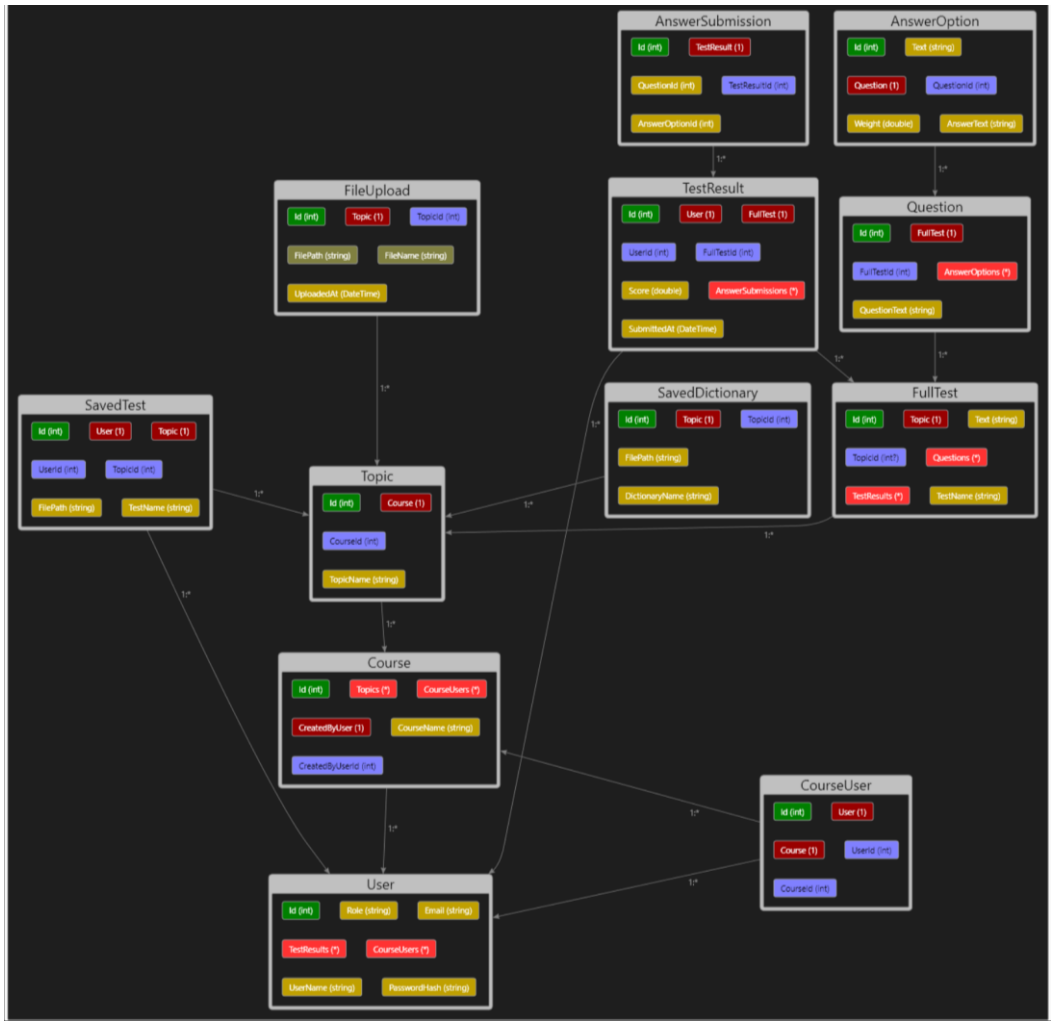

Рисунок 3.1 – Схема бази даних

Файли додаються через сутність FileUpload і можуть мати формат .txt, .pdf, .docx або .pptx. На основі цих файлів створюються тести (SavedTest або FullTest) та словники (SavedDictionary), які прив’язуються до відповідної теми.

У системі реалізовано два підходи до збереження тестових матеріалів: перший передбачає збереження тесту у вигляді згенерованого файлу (SavedTest), другий – у структурованому форматі (FullTest), де тест поділено на окремі запитання (Question) з відповідними варіантами

відповідей (AnswerOption), кожен із яких має визначений ваговий коефіцієнт.

Під час проходження тесту студент формує відповіді, що зберігаються в таблиці AnswerSubmission. Остаточні результати зберігаються в TestResult – вона містить інформацію про користувача, тест, дату проходження та оцінку.

Словники термінів представлені через сутність SavedDictionary і прив’язуються до теми так само, як і тести. Це забезпечує контекстний доступ до глосаріїв лише в межах відповідного навчального блоку, який вивчає студент.

Розроблена модель бази даних є гнучкою, масштабованою та повністю відповідає функціональним вимогам платформи. Вона дозволяє зберігати й організовувати всі необхідні дані – від навчальних курсів до результатів тестування та словників – у чітко структурованому вигляді, підтримуючи як індивідуальну, так і колективну взаємодію користувачів у межах системи.

3.3 Використані технології та компоненти системи

3.3.1 Реалізація клієнтської частини

Клієнтська частина системи реалізована з використанням JavaScript-бібліотеки React, яка забезпечує створення односторінкового застосунку із компонентною архітектурою [10]. Уся логіка інтерфейсу поділена на незалежні компоненти, що відповідають за окремі аспекти взаємодії з користувачем. Такий підхід дозволяє забезпечити гнучкість у розробці, полегшує масштабування та підтримку системи.

Основу навігаційної логіки складає компонент App, який керує переходами між усіма частинами інтерфейсу. Залежно від стану, встановленого через хуки useState, відображаються відповідні компоненти:

CoursesPage, TopicsPage, GeneratorPage, EditPage, TestEditorPage, TestPassingPage або TestHistoryPage. Перевірка авторизації користувача (наявність userId у localStorage) виконується в useEffect, і в разі її відсутності завантажується сторінка LoginPage. Замість використання сторонніх бібліотек маршрутизації (наприклад, React Router), усі переходи реалізовані через умовний рендеринг, що забезпечує просту й надійну логіку перемикання між сторінками без перезавантаження.

Окремі компоненти відповідають за такі завдання:

– CoursesPage відповідає за керування курсами та приєднання до них;

– TopicsPage відповідає за створення та перегляд тем у межах курсу;

– EditPage відповідає за редагування збережених тестів або словників;

– TestEditorPage відповідає за створення структурованих тестів із запитаннями та ваговими відповідями;

– TestPassingPage відповідає за проходження тестів студентами з підрахунком оцінки;

– TestHistoryPage відповідає за перегляд результатів проходжень;

– LoginPage відповідає за реєстрацію та авторизація користувача з вибором ролі.

Центральним елементом взаємодії користувача з платформою є компонент GeneratorPage, який відповідає за керування навчальними матеріалами та взаємодію з серверною частиною системи. У межах цього компонента реалізовано завантаження навчальних файлів, зчитування їх вмісту, а також ініціювання генерації тестів і словників. При цьому, ключова логіка формування контенту (аналіз тексту, побудова тестових запитань тощо) повністю реалізована на сервері, а клієнтська частина лише надсилає відповідні HTTP-запити до API.

Зв’язок із сервером відбувається через функцію fetch, з використанням REST-архітектури. Для передачі текстових даних застосовується формат JSON, а для завантаження файлів – FormData. Компонент також обробляє відповіді сервера та забезпечує візуалізацію згенерованих результатів, з

можливістю їх попереднього перегляду, збереження до бази даних або експорту в форматах .txt та .xml.

Після вибору теми користувач завантажує навчальний файл, вказує бажані параметри – кількість запитань і рівень складності та ініціює генерацію тесту. Ці дані надсилаються до серверного маршруту у форматі POST-запиту, а отриманий результат зберігається у локальному стані компонента та виводиться в інтерфейсі (рисунок 3.2).

```
const handleGenerateTest = async () => {
  if (!fileText || !questionCount) {
    alert("Заповніть усі поля!");
    return;
  }

  const response = await fetch("http://localhost:5048/api/test-generation/generate", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      text: fileText,
      questionCount: parseInt(questionCount),
      difficulty,
    }),
  });

  const result = await response.json();
  setTestResult(result.test);
};
```

Рисунок 3.2 – POST-запит до API на генерацію тесту

Генерація словника відбувається за схожим принципом, але без зазначення додаткових параметрів. Збереження результатів реалізовано через модальне вікно, де користувач вводить назву об’єкта. Залежно від типу (тест або словник), формується відповідний запит на API (рисунок 3.3). Перед відправленням запиту дані приводяться до необхідного формату та доповнюються метаданими, зокрема ідентифікатором теми та користувача. У разі успіху відображається повідомлення, а локальний стан оновлюється шляхом повторного завантаження пов’язаних з темою файлів.

```
const handleSave = async () => {
  const endpoint = isSavingTest ? "test-generation" : "dictionary-generation";
  const contentToSave = isSavingTest ? testResult : dictionaryResult;

  const response = await fetch(`http://localhost:5048/api/${endpoint}/save`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "UserId": localStorage.getItem("userId")
    },
    body: JSON.stringify({
      name: saveName,
      content: contentToSave,
      topicId: topic.id,
    }),
  });

  if (response.ok) {
    alert("Успішно збережено!");
    setShowModal(false);
    setSaveName("");
    fetchSavedFiles();
  } else {
    alert("Помилка при збереженні!");
  }
};
```

Рисунок 3.3 – Збереження згенерованого матеріалу через API-запит

Реалізовано також можливість створення повноцінного структурованого тесту (FullTest) на основі текстового варинту, шляхом попереднього зчитування вмісту файлу та надсилання його на сервер для створення об’єкта з ідентифікованими полями. Цей об’єкт надалі можна редагувати, проходити або експортувати у формат XML для імпорту в систему Moodle та інші системи тестування.

Рольова логіка враховується в інтерфейсі під час рендерингу компонентів. Залежно від збереженої ролі користувача, динамічно відображаються або приховуються певні елементи керування та дії. Це дозволяє реалізувати умовний доступ до функцій платформи безпосередньо на рівні клієнтської частини. На рисунку 3.4 наведено приклад такої умовної логіки відображення елементів інтерфейсу.

```jsx
<div>
  {localStorage.getItem("role") === "Teacher" && (
    <>
      <button className="icon" onClick={() => onEditTest(test.id)}>✏️</button>
      <button className="icon" onClick={async () => {
        const confirmed = window.confirm("Видалити цей повноцінний тест?");
        if (!confirmed) return;
        await fetch(`http://localhost:5048/api/fulltests/${test.id}`, { method: "DELETE" });
        fetchSavedFiles();
      }}>🗑️</button>
    </>
  )}
  <button className="icon" onClick={() => onPassTest(test.id)}>📝</button>
</div>
</div>
```

Рисунок 3.4 – Реалізація рольового доступу в інтерфейсі

Додатковою особливістю є можливість експорту тестових завдань та словників у форматах .txt та .xml. Експорт у форматі XML дозволяє інтегрувати створений контент з системами керування навчанням, такими як наприклад Moodle та інші. Уся логіка експорту реалізована у вигляді асинхронних запитів на відповідні маршрути API з подальшою генерацією тимчасового посилання для завантаження файлу (рисунок 3.5).

```jsx
const handleDownloadDictionaryXml = async (dictionaryName) => {
  try {
    const response = await fetch(`http://localhost:5048/api/dictionary-generation/extract-text/${encodeURIComponent(dictionaryName)}`);
    if (!response.ok) throw new Error("Файл не знайдено");

    const data = await response.json();
    const text = data.text;

    const xmlResponse = await fetch("http://localhost:5048/api/moodle-export/generate-dictionary", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(text),
    });

    if (!xmlResponse.ok) throw new Error("Помилка генерації XML для словника");

    const blob = await xmlResponse.blob();
    const url = URL.createObjectURL(blob);
    const link = document.createElement("a");
    link.href = url;
    link.download = `${dictionaryName}.xml`;
    document.body.appendChild(link);
    link.click();
    link.remove();
    URL.revokeObjectURL(url);
  } catch (error) {
    console.error("Помилка при створенні або завантаженні XML словника:", error);
    alert("Помилка при створенні або завантаженні XML словника");
  }
};
```

Рисунок 3.5 – Завантаження словника у форматі XML через API

Візуальне оформлення інтерфейсу реалізоване за допомогою CSS-файлів, що забезпечують єдиний стиль для всіх компонентів: кнопок, блоків виводу, заголовків, модальних вікон тощо. Кольорова схема та компонування інтерфейсу були підібрані з урахуванням зручності користувача, читабельності контенту та логіки взаємодії. Основна увага приділялась тому, щоб інтерфейс був інтуїтивно зрозумілим як для викладачів, так і для студентів, з можливістю швидко орієнтуватися у доступному функціоналі.

Таким чином, клієнтська частина системи не лише здійснює технічну взаємодію з сервером через REST API, а й забезпечує інтуїтивно зрозумілий та зручний інтерфейс для користувача, що відповідає вимогам сучасного освітнього програмного забезпечення. Вона підтримує генерацію, збереження, редагування й експорт навчального контенту, а також гнучку адаптацію до ролі користувача, створюючи зручне й ефективне середовище для навчання та викладання.

3.3.2 Реалізація серверної частини

Серверна частина системи реалізована з використанням багаторівневої архітектури, яка охоплює контролери, об’єкти передачі даних (DTO), контекст бази даних та окремі сервіси, призначені для взаємодії з API зовнішньої мовної моделі. Такий підхід забезпечує структуровану організацію коду, розділення відповідальностей і гнучкість при розширенні функціональності. Контролери відповідають за обробку вхідних HTTP-запитів, DTO-моделі – за стандартизовану передачу інформації між клієнтською частиною та сервером, а сервіси – за формування запитів до LLM та обробку відповідей. Контекст бази даних забезпечує доступ до збережених сутностей через ORM-технологію Entity Framework Core, а моделі, що його описують, використовуються для

збереження основних об’єктів: курсів, тем, тестів, користувачів, результатів тощо.

У межах контролерного рівня реалізовано повний спектр функціональних можливостей системи, кожен з яких відповідає за обробку конкретної категорії запитів.

Зокрема, контролер CoursesController реалізує повний цикл керування курсами. Він надає можливість викладачу створювати курси (рисунок 3.6), студентам – приєднуватися до них, переглядати список підключених курсів, а також залишати курс або видаляти його повністю. Логіка перевірки ролі користувача, обробка ID курсу та користувача, а також взаємодія з таблицею зв’язку CourseUsers реалізовані з використанням асинхронних методів доступу до бази даних через Entity Framework Core.

```
[HttpPost]
public async Task<IActionResult> CreateCourse([FromBody] CreateCourseDto dto)
{
    if (string.IsNullOrWhiteSpace(dto.CourseName))
        return BadRequest(new { message = "Назва курсу не може бути порожньою" });

    var user = await _context.Users.FindAsync(dto.CreatedByUserId);
    if (user == null || user.Role != "Teacher")
        return BadRequest("Недійсний викладач");

    var course = new Course
    {
        CourseName = dto.CourseName,
        CreatedByUserId = dto.CreatedByUserId
    };

    _context.Courses.Add(course);
    await _context.SaveChangesAsync();

    return Ok(course);
}
```

Рисунок 3.6 – Реалізація методу створення нового курсу

Контролер FilesController відповідає за завантаження та обробку навчальних матеріалів. Після завантаження файлу (у форматах .pdf, .docx, .pptx, .txt) він зберігається на сервері та фіксується в базі даних із

прив’язкою до теми. Особливістю є можливість витягування тексту з цих файлів для подальшої обробки – наприклад, для генерації тестів або словників. Для кожного типу документа використовується відповідна технологія парсингу, що дозволяє забезпечити стабільну підтримку різноманітних форматів (рисунок 3.7).

```
private string ExtractTextFromPdf(string filePath)
{
    var text = new StringBuilder();
    using (var document = PdfDocument.Open(filePath))
    {
        foreach (var page in document.GetPages())
        {
            text.AppendLine(ContentOrderTextExtractor.GetText(page));
        }
    }
    return text.ToString();
}

private string ExtractTextFromWord(string filePath)
{
    using var stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    using var wordDoc = WordprocessingDocument.Open(stream, false);
    var body = wordDoc.MainDocumentPart?.Document.Body;
    return body?.InnerText ?? "";
}

private string ExtractTextFromPowerPoint(string filePath)
{
    using var stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    using var presentation = PresentationDocument.Open(stream, false);
    var textBuilder = new StringBuilder();

    var slideParts = presentation.PresentationPart?.SlideParts;
    if (slideParts != null)
    {
        foreach (var slidePart in slideParts)
        {
            var texts = slidePart.Slide.Descendants<A.Text>();
            foreach (var text in texts)
            {
                textBuilder.AppendLine(text.Text);
            }
        }
    }

    return textBuilder.ToString();
}
```

Рисунок 3.7 – Реалізація методів вилучення тексту з pdf-, docx- та pptx- документів

Контролер FullTestController реалізує операції створення, редагування, видалення та перегляду повноцінних тестів. Він дозволяє зберігати структуру тесту у вигляді списку запитань із варіантами відповідей, кожен з яких має ваговий коефіцієнт. Крім того, в контролері реалізовано функції збереження результатів проходження тесту студентом, а також перегляду оцінок, статистики й історії проходжень (рисунок 3.8). Для коректного обліку оцінки враховується не лише факт правильності відповіді, а й часткова відповідність – завдяки використанню числових ваг.

```
[HttpPost("submit")]
public async Task<IActionResult> SubmitTest([FromBody] SubmitTestDto dto)
{
    var fullTest = await _context.FullTests
        .Include(t => t.Questions)
        .ThenInclude(q => q.AnswerOptions)
        .FirstOrDefaultAsync(t => t.Id == dto.FullTestId);

    if (fullTest == null)
    {
        return NotFound("Тест не знайдено");
    }

    double totalScore = 0;

    foreach (var submission in dto.AnswerSubmissions)
    {
        var selectedOption = fullTest.Questions
            .SelectMany(q => q.AnswerOptions)
            .FirstOrDefault(a => a.Id == submission.AnswerOptionId);

        if (selectedOption != null)
        {
            totalScore += selectedOption.Weight;
        }
    }

    var testResult = new TestResult
    {
        UserId = dto.UserId,
        FullTestId = dto.FullTestId,
        SubmittedAt = dto.SubmittedAt,
        Score = Math.Round(totalScore, 2),
        AnswerSubmissions = dto.AnswerSubmissions.Select(a => new AnswerSubmission
        {
            QuestionId = a.QuestionId,
            AnswerOptionId = a.AnswerOptionId
        }).ToList()
    };

    _context.TestResults.Add(testResult);
    await _context.SaveChangesAsync();
    return Ok("Результат збережено!");
}
```

Рисунок 3.8 – Метод збереження результату проходження тесту

Окремої уваги заслуговує контролер MoodleExportController, який забезпечує конвертацію згенерованих тестів або глосаріїв у формат XML, сумісний із платформою Moodle (рисунок 3.9). Він обробляє структуровані текстові дані, перетворюючи їх у відповідні XML-теги відповідно до вимог системи тестування Moodle. Це дозволяє викладачам експортувати створені тести для подальшого використання у зовнішньому середовищі навчання.

```
private List<DictionaryEntryModel> ParseDictionaryText(string text)
{
    var entries = new List<DictionaryEntryModel>();

    var lines = text.Split(new[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

    foreach (var line in lines)
    {
        var parts = line.Split(new[] { " – " }, 2, StringSplitOptions.None);
        if (parts.Length == 2)
        {
            entries.Add(new DictionaryEntryModel
            {
                Term = parts[0].Trim(),
                Definition = parts[1].Trim()
            });
        }
    }

    return entries;
}
```

Рисунок 3.9 – Метод парсингу згенерованого словника

Контролер UsersController реалізує функціональність, пов’язану з реєстрацією, авторизацією та участю користувачів у навчальному процесі (рисунок 3.10). Під час реєстрації система перевіряє унікальність електронної адреси та зберігає базову інформацію про користувача, включно з його роллю. Авторизація здійснюється шляхом перевірки наданих облікових даних, після чого повертаються ідентифікатор і роль користувача. Окремий метод забезпечує приєднання до курсу за допомогою передачі ідентифікаторів користувача та курсу, а також перевірку наявності вже існуючого зв’язку. Реалізовано також можливість отримання переліку курсів, до яких приєднався конкретний користувач.

```
[HttpPost("register")]
public async Task<IActionResult> Register([FromBody] RegisterUserDto dto)
{
    if (_context.Users.Any(u => u.Email == dto.Email))
        return BadRequest("Email already exists");

    var user = new User
    {
        UserName = dto.UserName,
        Email = dto.Email,
        PasswordHash = dto.Password,
        Role = dto.Role
    };

    _context.Users.Add(user);
    await _context.SaveChangesAsync();

    return Ok(user);
}
```

Рисунок 3.10 – Реєстрація нового користувача в системі

Контролер TopicsController відповідає за керування тематичним наповненням курсів. Він надає інтерфейс для створення, перегляду та видалення тем, які належать до конкретного курсу. Перед створенням нової теми здійснюється перевірка існування курсу, що забезпечує цілісність зв’язків у структурі даних. Теми є ключовим елементом ієрархії навчального контенту, оскільки саме до них прив’язуються завантажені навчальні матеріали, тести та глосарії.

Окремим напрямом роботи є контролери, пов’язані з інтеграцією мовної моделі для автоматизованої генерації навчального контенту. Зокрема, TestGenerationController і DictionaryGenerationController відповідають за обробку запитів користувача, формування промптів на основі наданих матеріалів та ініціалізацію запитів до зовнішнього API моделі GPT. Отримані результати повертаються у вигляді тексту, придатного для подальшого збереження, редагування або експорту в інші системи. Таким чином, ці контролери відіграють ключову роль у реалізації

інтелектуальної частини системи, делегуючи обчислювальні завдання сервісам, зокрема класам TestGenerator та DictionaryGenerator.

Крім контролерів, важливу роль у структурі застосунку відіграють сервіси, винесені в окрему частину проєкту. Серед них – TestGenerator і DictionaryGenerator, які відповідають за взаємодію з зовнішнім API мовної моделі. Вони використовуються для генерації тестів і глосаріїв на основі завантажених навчальних матеріалів. Ці сервіси реалізовані як автономні компоненти, що не залежать від конкретних контролерів, а можуть викликатися з різних частин системи. Така реалізація відповідає принципам чистої архітектури та дозволяє гнучко масштабувати проєкт у майбутньому.

Для зручної та безпечної взаємодії між клієнтом і сервером використовуються об’єкти передачі даних (DTO). Вони дозволяють чітко структурувати запити й відповіді, відокремлюючи зовнішні дані від внутрішніх моделей бази. Це не лише підвищує надійність взаємодії між компонентами, але й спрощує обробку вхідної інформації, роблячи API зрозумілим, контрольованим і легко підтримуваним.

Загалом, реалізація серверної частини поєднує класичні підходи до побудови API з інструментами сучасної обробки даних та інтеграції зі сторонніми сервісами. Така структура забезпечує чітке розділення обов’язків між компонентами, зручність масштабування та гнучкість у розвитку проєкту. Продумана архітектура серверної логіки дозволяє ефективно обробляти запити, зберігати й керувати освітнім контентом, а також інтегрувати інтелектуальні функції без порушення загальної стабільності системи.

3.4 Генерація дистракторів великою мовною моделлю

Одним із важливих етапів створення якісних тестових завдань із множинним вибором є формування дистракторів – варіантів відповідей, які є помилковими, але виглядають логічно й переконливо. Якщо правильна

відповідь повинна спиратися на факти, викладені в навчальних матеріалах, то дистрактори мають бути достатньо схожими, аби створювати когнітивну складність для студента, водночас не підриваючи довіру до системи оцінювання загалом. У традиційній практиці ця задача вимагає досвіду та інтуїції викладача, оскільки потребує врахування типових помилок, рівня складності теми, а також лінгвістичної й змістової узгодженості всіх варіантів відповіді.

З появою великих мовних моделей (LLM), таких як GPT, відкриваються нові можливості для автоматизації цього процесу. Такі моделі здатні аналізувати контекст запитання, правильну відповідь і навчальний текст, а потім формувати альтернативні варіанти, які здаються логічними, але фактично є неправильними. Основою такого підходу є здатність моделі відтворювати мовні і логічні зв’язки, які вона «вивчила» з мільярдів прикладів під час попереднього навчання. Однак варто розуміти, що мовна модель не володіє справжнім знанням – вона не розрізняє істину від вигадки, а лише передбачає найбільш ймовірну послідовність слів на основі заданого контексту. Саме це й зумовлює здатність моделі створювати фрази, які виглядають переконливо, але не мають підґрунтя в реальних фактах.

У запропонованому підході генерація дистракторів відбувається як частина загального процесу автоматизованого створення тестових завдань на основі навчального тексту. Мовна модель отримує на вхід лише текстовий матеріал і загальну інструкцію сформувати тестові питання з варіантами відповідей. У результаті модель самостійно виокремлює ключові концепти, формулює запитання, обирає правильні відповіді та генерує дистрактори – неправдиві, але логічно схожі варіанти. Оскільки модель не має доступу до зовнішніх баз знань і працює на основі ймовірнісного передбачення наступних слів, згенеровані дистрактори часто є результатом так званих «галюцинацій» – тверджень, які звучать правдоподібно, але не мають достовірного підтвердження. У контексті

тестування це явище набуває функціональної цінності: подібні варіанти підвищують складність завдань і дозволяють перевірити глибину розуміння матеріалу, а не лише здатність до запам’ятовування фактів.

Особливу увагу заслуговує ситуація, коли студенти під час проходження таких тестів звертаються до LLM як до «помічника». У цьому випадку існує високий ризик, що модель, аналізуючи сформоване запитання, обере дистрактор як правильну відповідь – адже саме вона (або подібна до неї відповідь) могла бути згенерована іншою LLM. Обидва випадки – і створення дистрактора, і його «вибір» – базуються на одних і тих самих статистичних механізмах генерації, а не на перевірці фактів. Це підкреслює важливість критичного мислення та усвідомленого підходу до роботи з мовними моделями – як під час створення тестів, так і при спробах отримати з їхньою допомогою відповіді.

Отже, застосування великих мовних моделей для генерації дистракторів – це не лише інструмент автоматизації, але й приклад того, як властивості таких технологій, зокрема здатність до правдоподібних помилок (галюцинацій), можуть бути використані з педагогічною метою. Удало сформовані дистрактори роблять тестування більш ефективним і справедливим, одночасно демонструючи як потужність, так і межі сучасних мовних технологій.

3.5 Реалізація генерації тестів і глосаріїв

У межах реалізованої програмної системи автоматичне створення тестів і глосаріїв здійснюється з використанням великих мовних моделей (LLM), зокрема GPT-4 Turbo, через інтеграцію з API OpenAI. Зв’язок із мовною моделлю реалізовано за допомогою клієнта OpenAIService, який ініціалізується в кожному з сервісів (TestGenerator та DictionaryGenerator) із використанням ключа доступу (ApiKey). Модель працює в режимі ChatCompletion, де користувацький запит (prompt)

передається у вигляді структури ChatCompletionCreateRequest, що включає повідомлення від системи та користувача. Саме у цих повідомленнях задається інструкція до генерації – створити тестові завдання або глосарій на основі вхідного тексту. У відповідь від моделі повертається сформований текст, який система передає користувачеві або зберігає на сервері.

Сервіс TestGenerator призначений для створення тестів на основі текстових навчальних матеріалів (рисунок 3.11).

```
public async Task<string> GenerateTestFromText(string text, int questionCount, string difficulty)
{
    string prompt =
    $"Прочитай наступний текст і створи {questionCount} запитань для тесту. " +
    $"Рівень складності: {difficulty}. " +
    "Кожне запитання повинно мати 5 варіантів відповідей, де одна або кілька відповідей правильні.
    "Кожне варіант відповідей має мати вагу правильності, тобто якщо за тестове завдання коштує од
    "Додай рівень складності до кожного запитання (простий, середній, складний). " +
    "Знизу під відповідями обовьязково пиши правильні відповіді. " +
    "Роби тестові завдання обовьязково лише в такому форматі : Які властивості визначають стандарт
    "**Рівень складності:** складний" +
    "A) `margin` встановлює внутрішні відступи блоку" +
    "B) `padding` встановлює внутрішні відступи блоку від його зовнішніх меж до контенту(0,3)" +
    "C) `border-width` визначає ширину рамки елемента(0,4)" +
    "D) `height` та `width` встановлюють зовнішні розміри боксу, включаючи рамку та внутрішні/зовн
    "E) `margin` встановлює зовнішні відступи блоку від його меж до сусідніх елементів або контейне
    "**Правильні відповіді:**" +
    "B, C, E " +
    $"Ось текст лекції:\n{text}";

    var chatRequest = new ChatCompletionCreateRequest
    {
        Messages = new List<ChatMessage>
        {
            new ChatMessage("system", "Ти створюєш тест на основі прочитаного тексту."),
            new ChatMessage("user", prompt)
        },
        Model = Models.Gpt_4_turbo
    };

    var response = await _openAiService.ChatCompletion.CreateCompletion(chatRequest);

    if (response.Successful)
    {
        return response.Choices[0].Message.Content;
    }
    else
    {
        return "Помилка генерації тесту: " + response.Error?.Message;
    }
}
```

Рисунок 3.11 – Метод для створення тесту за допомогою GPT-4 Turbo

На вхід метод GenerateTestFromText приймає три параметри: текст лекції, кількість запитань (questionCount) та рівень складності (difficulty). У

тілі запиту формується детальний prompt, який інструктує мовну модель створити фіксовану кількість запитань у форматі multiple-choice із п’ятьма варіантами відповідей. Кожна відповідь має свою вагу правильності, наприклад, 0.3, 0.4. А загальна вага правильних варіантів повинна дорівнювати 1. Крім цього, вказується складність кожного запитання та перелік правильних варіантів.

Сервіс DictionaryGenerator виконує генерацію глосарію з термінами на основі переданого тексту (рисунок 3.12).

```
public async Task<string> GenerateDictionaryFromText(string text)
{
    string prompt =
    $"Прочитай наступний текст і створи словник на основі прочитаного текст
    $"Словник має складатися з ключів – основних понять або термінів тексту
    $"Слово – визначення\n" +
    $"Наприклад:\n" +
    $"Алгоритм – набір послідовних дій для розв'язання задачі.\n" +
    $"Ось текст :\n{text}";

    var chatRequest = new ChatCompletionCreateRequest
    {
        Messages = new List<ChatMessage>
        {
            new ChatMessage("system", "Ти створюєш словник на основі прочит
            new ChatMessage("user", prompt)
        },
        Model = Models.Gpt_4_turbo
    };

    var response = await _openAiService.ChatCompletion.CreateCompletion(cha

    if (response.Successful)
    {
        return response.Choices[0].Message.Content;
    }
    else
    {
        return "Помилка генерації словника: " + response.Error?.Message;
    }
}
```

Рисунок 3.12 – Метод для створення словника за допомогою GPT-4 Turbo

Запит до LLM містить інструкцію проаналізувати навчальний матеріал та сформувати список ключових понять у форматі «термін – коротке визначення». Цей формат дозволяє швидко формувати

термінологічну базу для теми, що вивчається. Зразок prompt-інструкції наведено на рис. 3.12.

Взаємодія з мовною моделлю організована через REST API-контролери TestGenerationController та DictionaryGenerationController, реалізовані на основі ASP.NET Core. Вони обробляють HTTP-запити у форматі JSON, перевіряють коректність отриманих даних і передають їх до відповідного сервісу генерації. У випадку успішної відповіді результат обробки LLM – згенерований тест або словник – повертається клієнту, а також може бути збережений на сервері у вигляді текстового файлу. Метод генерації тесту у контролері TestGenerationController продемонстрована на рисунку 3.13.

```
[HttpPost("generate")]
public async Task<IActionResult> GenerateTest([FromBody] TestRequest request)
{
    if (string.IsNullOrWhiteSpace(request.Text))
        return BadRequest(new { message = "Текст не може бути порожнім!" });

    if (request.QuestionCount < 1 || request.QuestionCount > 20)
        return BadRequest(new { message = "Кількість питань повинна бути від 1 до 20!" });

    if (!new[] { "простий", "середній", "складний" }.Contains(request.Difficulty?.ToLower()))
        return BadRequest(new { message = "Рівень складності має бути 'простий', 'середній' або 'складний'!" });

    try
    {
        string result = await _testGenerator.GenerateTestFromText(request.Text, request.QuestionCount, request.Difficulty!);
        return Ok(new { message = "Тест згенеровано", test = result });
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "Помилка при генерації тесту", error = ex.Message });
    }
}
```

Рисунок 3.13 – Метод генерації тесту у контролері

Для забезпечення зручної роботи з матеріалами система підтримує можливість їх збереження, редагування, перегляду та видалення. Ця функціональність реалізована відповідними методами контролерів, які працюють з локальними файлами у визначених директоріях (SavedTests і SavedDictionaries). Таким чином, користувачі можуть зберігати результати генерації для подальшого використання, а також здійснювати повторний

доступ до збережених матеріалів у зручному форматі. Логіка збереження продемонстрована на рисунку 3.14.

```
[HttpPost("save")]
public async Task<IActionResult> SaveDictionary([FromBody] SaveDictionaryRequest request)
{
    if (string.IsNullOrWhiteSpace(request.Name) || string.IsNullOrWhiteSpace(request.Content))
        return BadRequest("Назва або зміст словника не може бути порожнім");

    if (!_context.Topics.Any(t => t.Id == request.TopicId))
        return BadRequest("Тема не знайдена");

    var filePath = Path.Combine(_uploadPath, $"{request.Name}.txt");

    await System.IO.File.WriteAllTextAsync(filePath, request.Content);

    var savedDictionary = new SavedDictionary
    {
        DictionaryName = request.Name,
        FilePath = filePath,
        TopicId = request.TopicId
    };

    _context.Dictionaries.Add(savedDictionary);
    await _context.SaveChangesAsync();

    return Ok(new { message = "Словник збережено", name = request.Name });
}
```

Рисунок 3.14 – Метод збереження згенерованого словника у контролері

Запропонований підхід вдало поєднує можливості великих мовних моделей із зручністю використання через веб-інтерфейс.

Завдяки чітко організованій структурі контролерів і сервісів, генерація тестових завдань і тематичних словників відбувається автоматично, з мінімальною участю користувача.

Усі обчислення виконуються асинхронно, а результати можуть зберігатися у вигляді окремих файлів, що спрощує їх повторне використання або подальше редагування.

Така інтеграція LLM у навчальний процес не лише зменшує обсяг рутинної роботи для викладача, а й створює умови для масштабованої розробки якісних матеріалів, максимально адаптованих до конкретного курсу.

3.6 Інтерфейс користувача та система ролей

Графічний інтерфейс користувача реалізовано у вигляді веб-застосунку, створеного за допомогою React. Інтерфейс забезпечує доступ до основних функцій системи, таких як завантаження навчальних матеріалів, генерація тестів і глосаріїв, збереження результатів, редагування та перегляд згенерованого контенту. Особливу увагу приділено простоті користування: дизайн витримано в єдиному стилі з акцентом на фіолетову кольорову гаму, округлі кнопки, модальні вікна та централізоване розміщення елементів. Завдяки цьому інтерфейс інтуїтивно зрозумілий навіть для нових користувачів та не вимагає додаткових інструкцій.

Залежно від ролі, яка присвоєна користувачеві під час реєстрації або авторизації, інтерфейс адаптується до його потреб та рівня доступу. У системі реалізовано два типи ролей: викладач та студент, кожна з яких має власний набір функцій.

На рівні взаємодії користувача з системою передбачено два окремі інтерфейси: форма входу та форма реєстрації. Обидва компоненти реалізовано з урахуванням єдиного стилю оформлення: заокруглені кнопки, фіолетова кольорова гама, тінь для кнопок і панелей, адаптивне розміщення полів (рисунок 3.15).

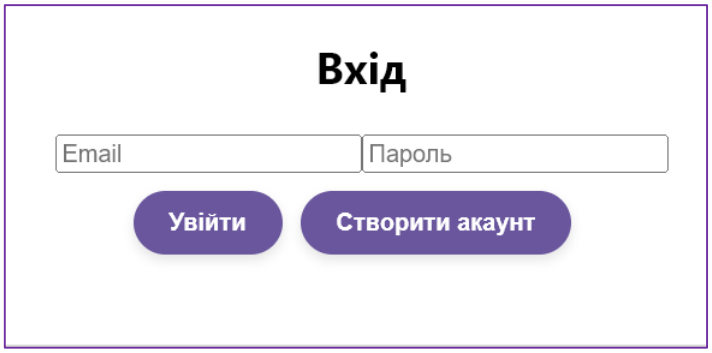

Рисунок 3.15 – Форма входу до системи

Форма реєстрації містить додаткове поле для вибору ролі користувача (викладач або студент), що дозволяє визначити рівень доступу до функціоналу одразу під час створення облікового запису (рисунок 3.16).

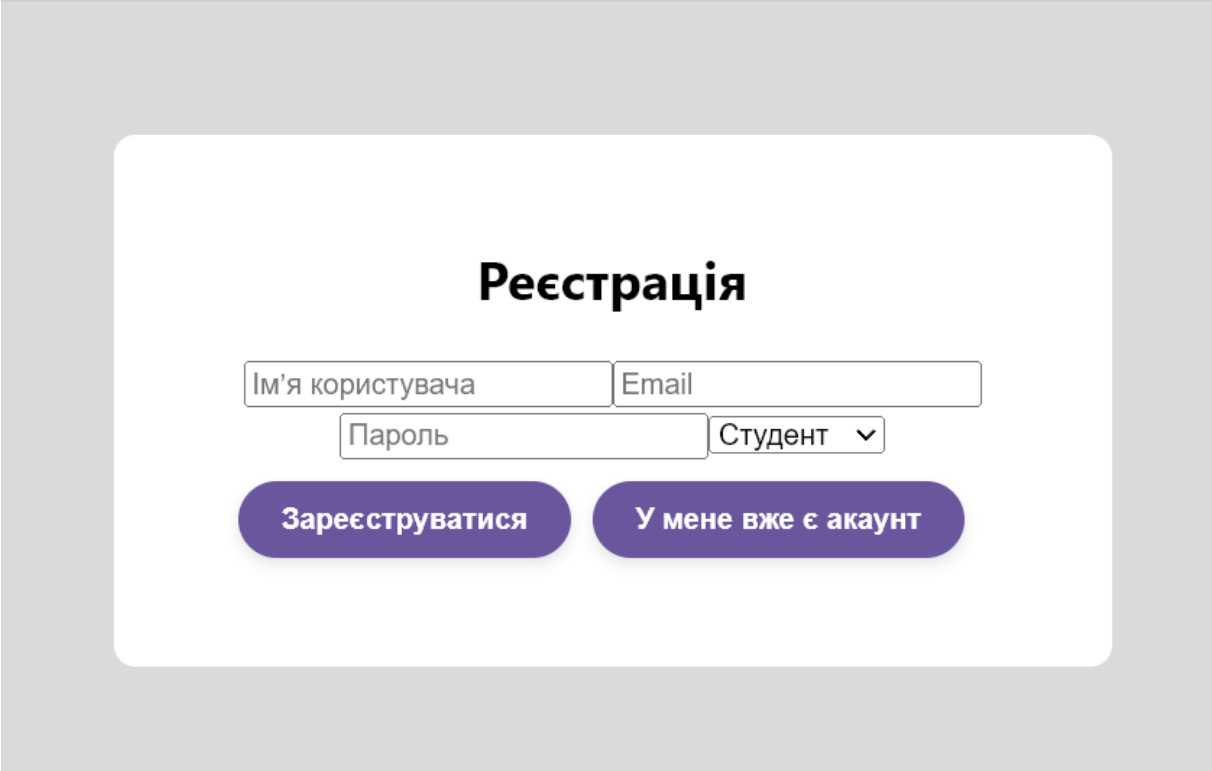

Рисунок 3.16 – Форма реєстрації користувача

Після успішної авторизації користувач потрапляє на сторінку з курсами (рисунок 3.17). У залежності від ролі, наданої під час реєстрації, інтерфейс набуває різного вигляду. Для студентів система надає можливість приєднатися до наявного курсу за кодом. Після введення правильного коду користувач отримує доступ до структури курсу, яка складається з окремих тем. У межах кожної теми студент може переглядати відкриті викладачем навчальні матеріали – зокрема, тести для проходження та відповідні глосарії. У разі потреби студент також може скористатися кнопкою «Покинути курс», яка дозволяє видалити поточну прив'язку до курсу та вийти з нього.

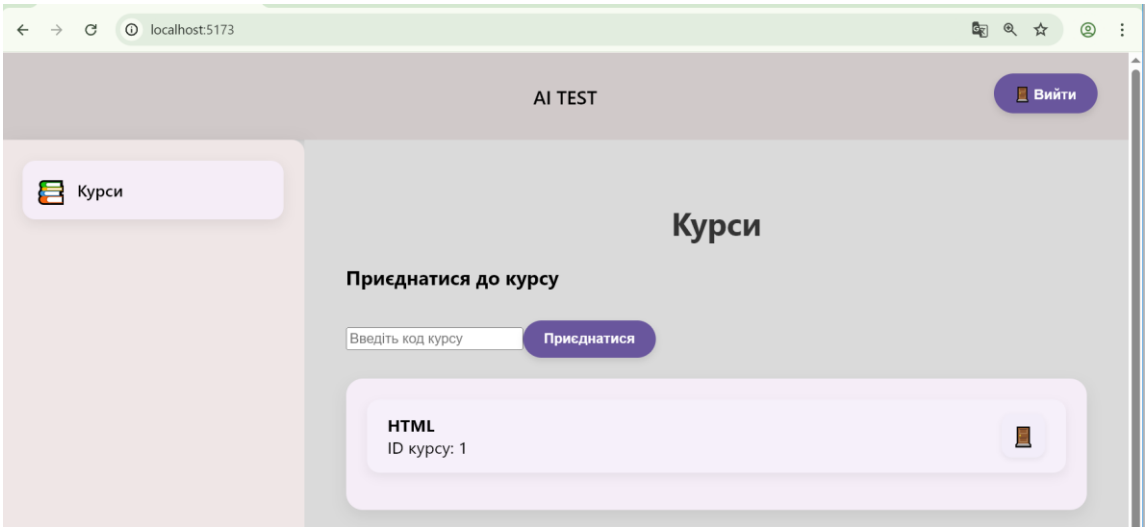

Рисунок 3.17 – Інтерфейс приєднання до курсу для студента

У свою чергу, викладачі мають доступ до розширеного функціоналу. Після входу в систему вони бачать список створених курсів і можуть додавати нові за допомогою відповідної кнопки (рисунок 3.18). Інтерфейс також дозволяє видаляти існуючі курси, якщо вони більше не потрібні. Всі дії виконуються через зручні інтерактивні елементи – кнопки з іконками створення та видалення. Кожен курс відображається у вигляді окремого блока зі стилізованим оформленням.

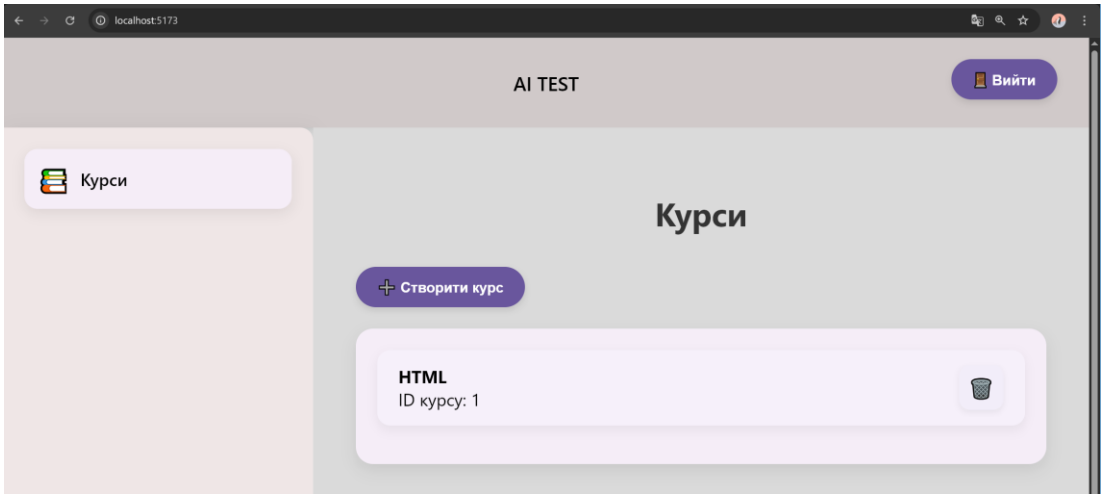

Рисунок 3.18 – Інтерфейс викладача з можливістю створення нового курсу та керування існуючими

Після вибору курсу користувач переходить на сторінку перегляду тем, пов’язаних із цим курсом. Кожна тема відображається у вигляді окремого візуального блоку з єдиним оформленням, що відповідає загальному стилю застосунку. Список тем є основною точкою входу до наступних функцій – таких як генерація тестів, словників або проходження контрольних завдань.

Для студентів інтерфейс сторінки тем має спрощений вигляд, орієнтований на зручне сприйняття інформації (рисунок 3.19). Користувач бачить перелік тем, які входять до вибраного курсу. Кожна тема подана у вигляді окремого візуального елемента з чистим мінімалістичним оформленням. Усі елементи редагування, створення або видалення відсутні, що дозволяє зосередити інтерфейс виключно на навчальному контенті. Такий підхід виключає можливість випадкового втручання в структуру курсу та забезпечує чітке розмежування функціоналу відповідно до ролі користувача.

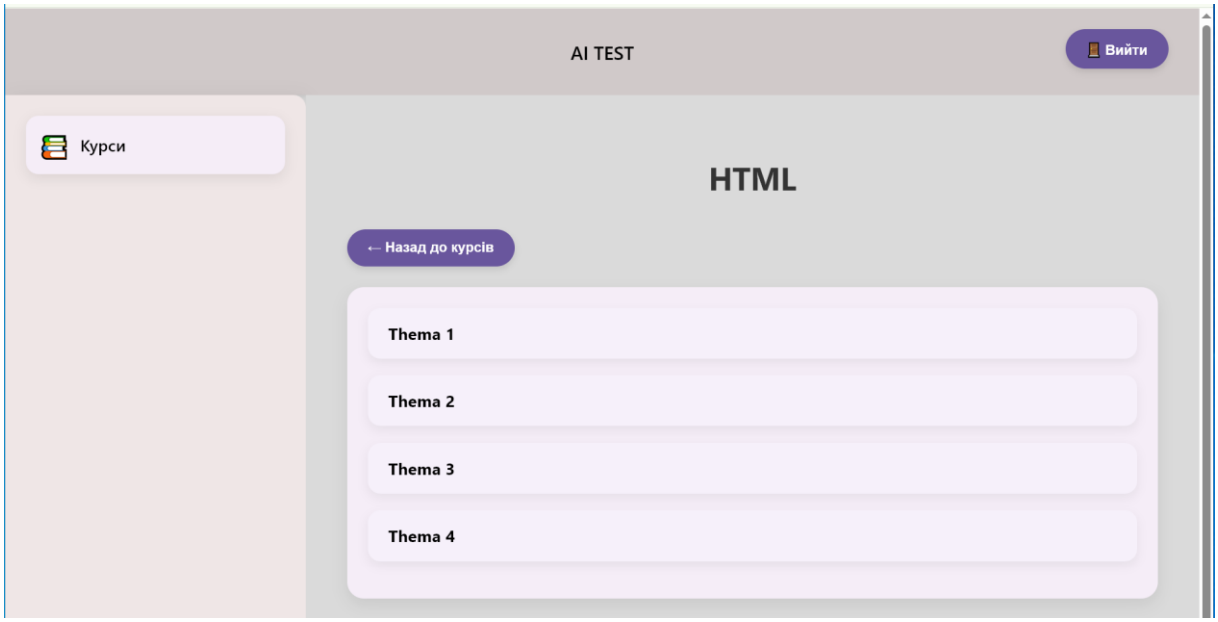

Рисунок 3.19 – Інтерфейс студента зі списком тем обраного курсу

На відміну від студентського інтерфейсу, викладач отримує повний контроль над структурою курсу. Він може не лише переглядати наявні теми,

але й додавати нові за допомогою кнопки «Створити тему», яка відкриває модальне вікно з полем введення назви (рисунок 3.20). Після створення нова тема одразу з’являється у списку без потреби оновлення сторінки (рисунок 3.21). Крім того, біля кожної теми розміщено іконку видалення, що дозволяє швидко редагувати склад тем курсу. Така гнучкість дає змогу викладачу динамічно формувати зміст курсу відповідно до навчальної програми або поточних потреб.

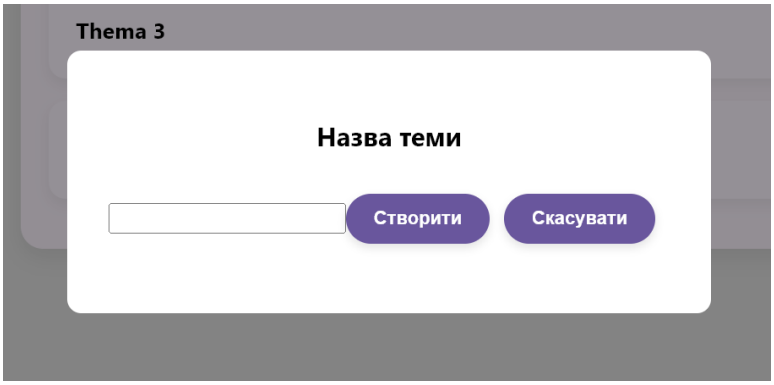

Рисунок 3.20 – Модальне вікно створення нової теми

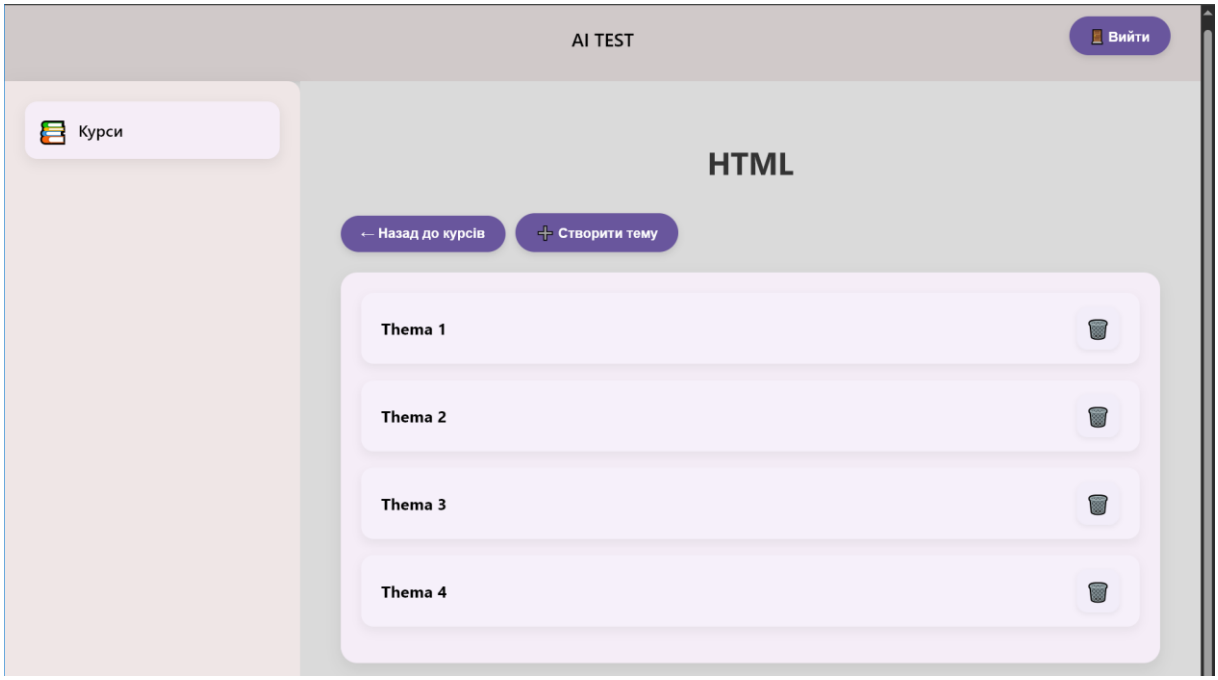

Рисунок 3.21 – Інтерфейс викладача зі списком тем обраного курсу

Після вибору конкретної теми користувач потрапляє на окрему сторінку, присвячену роботі з матеріалами цієї теми. На сторінці зібрано всі доступні функції: перегляд опублікованих тестів, робота з власними збереженими тестами та словниками, а також інструменти для генерації нових матеріалів на основі навчального тексту. Залежно від ролі користувача – студента чи викладача – інтерфейс цієї сторінки дещо відрізняється за набором доступних дій (рисунок 3.22).

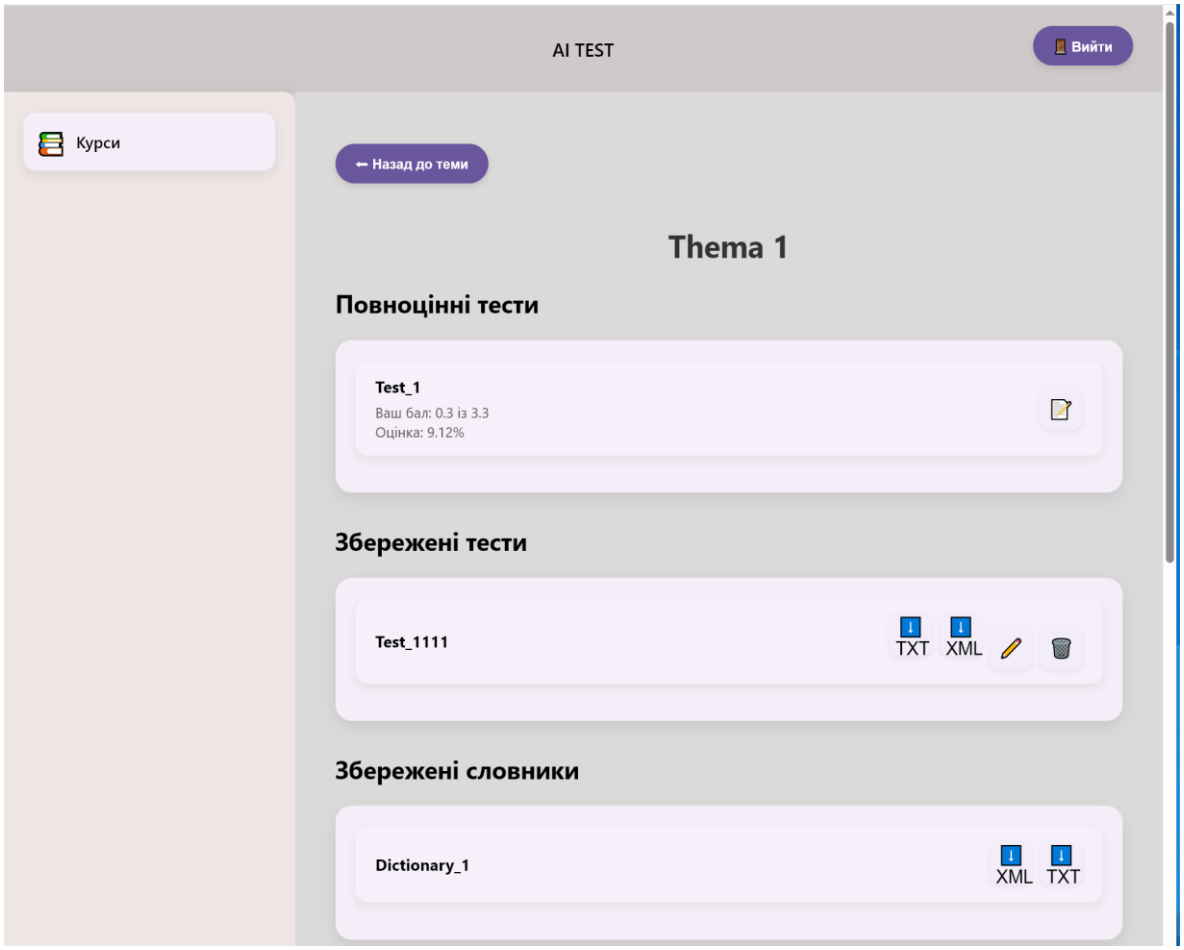

Рисунок 3.22 – Інтерфейс студента зі згенерованими матеріалами теми

У верхній частині сторінки розміщено блок «Повноцінні тести» – це тести, які були опубліковані викладачем для проходження. Студент бачить назву тесту та, якщо він уже пройшов його раніше, – інформацію про набраний бал і відсоток правильних відповідей. Біля кожного тесту

доступна кнопка для проходження тесту – після натискання відкривається інтерфейс виконання завдань.

Нижче розташовано секцію «Збережені тести», яка містить тести, що були згенеровані студентом самостійно. Такі тести не публікуються викладачем і використовуються для індивідуальної підготовки. Користувач може редагувати або видаляти ці тести, а також завантажувати їх у форматах .txt або .xml для подальшої роботи поза системою.

Останній блок «Збережені словники» містить глосарії, створені викладачем для підтримки навчального процесу.

Кожен словник представлений у вигляді окремого елемента з назвою, поруч з якою розміщені кнопки для завантаження у форматах .xml та .txt. Це дозволяє студенту зручно зберегти термінологічний матеріал для подальшого використання в автономному режимі або під час підготовки до тестування.

У нижній частині сторінки теми розташовано функціональний блок генерації навчальних матеріалів, який дозволяє студенту створити власний тест або словник на основі завантаженого викладачем навчального файлу.

У секції «Менеджер файлів» відображається список доступних документів, що були прикріплені до теми – зазвичай це лекції у форматі .pdf. Після вибору конкретного файлу стають активними два незалежних блоки: «Генерація тесту» та «Генерація словника».

У блоці генерації тесту користувач має можливість ввести кількість запитань та обрати бажаний рівень складності зі списку (простий, середній, складний), після чого натискає кнопку для створення тестового матеріалу.

Нижче розташовано кнопку для створення словника, яка запускає процес виділення ключових термінів із тексту.

До моменту вибору файлу обидві кнопки залишаються неактивними, що запобігає генерації без джерела контенту (рисунок 3.23).

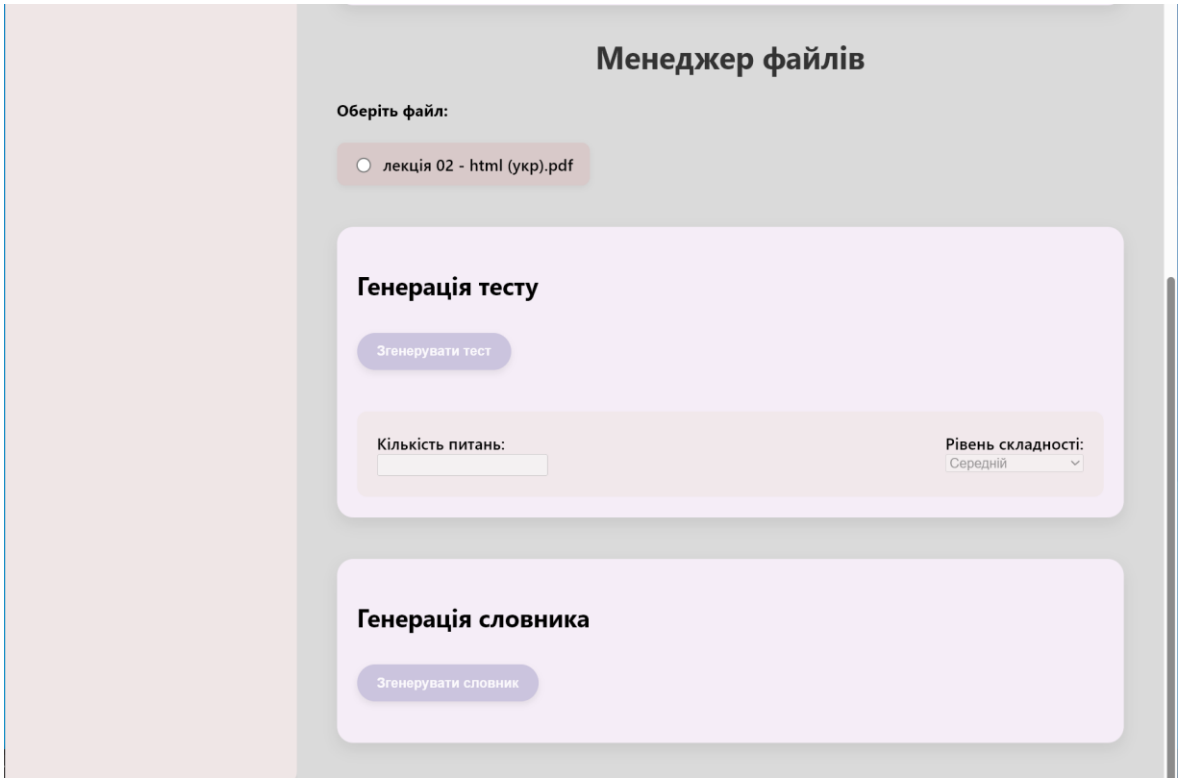

Рисунок 3.23 – Інтерфейс студента генерація тесту і словника

Інтерфейс проходження тесту реалізований у простій і зручній для користувача формі (рисунок 3.24).

У верхній частині сторінки виводиться текст тесту з усіма запитаннями та варіантами відповідей. Кожне запитання супроводжується зазначенням рівня складності та п’ятьма варіантами відповідей, з яких студент може обрати один або кілька, залежно від типу завдання.

Візуальне оформлення забезпечує чітке розділення запитань, зручне групування відповідей і виразні елементи керування.

У нижній частині сторінки розміщується кнопка «Завершити тест», яка надсилає відповіді на перевірку.

Після завершення тесту студент автоматично переходить на сторінку результатів, де відображається отриманий бал та відсоток правильних відповідей (рисунок 3.25).

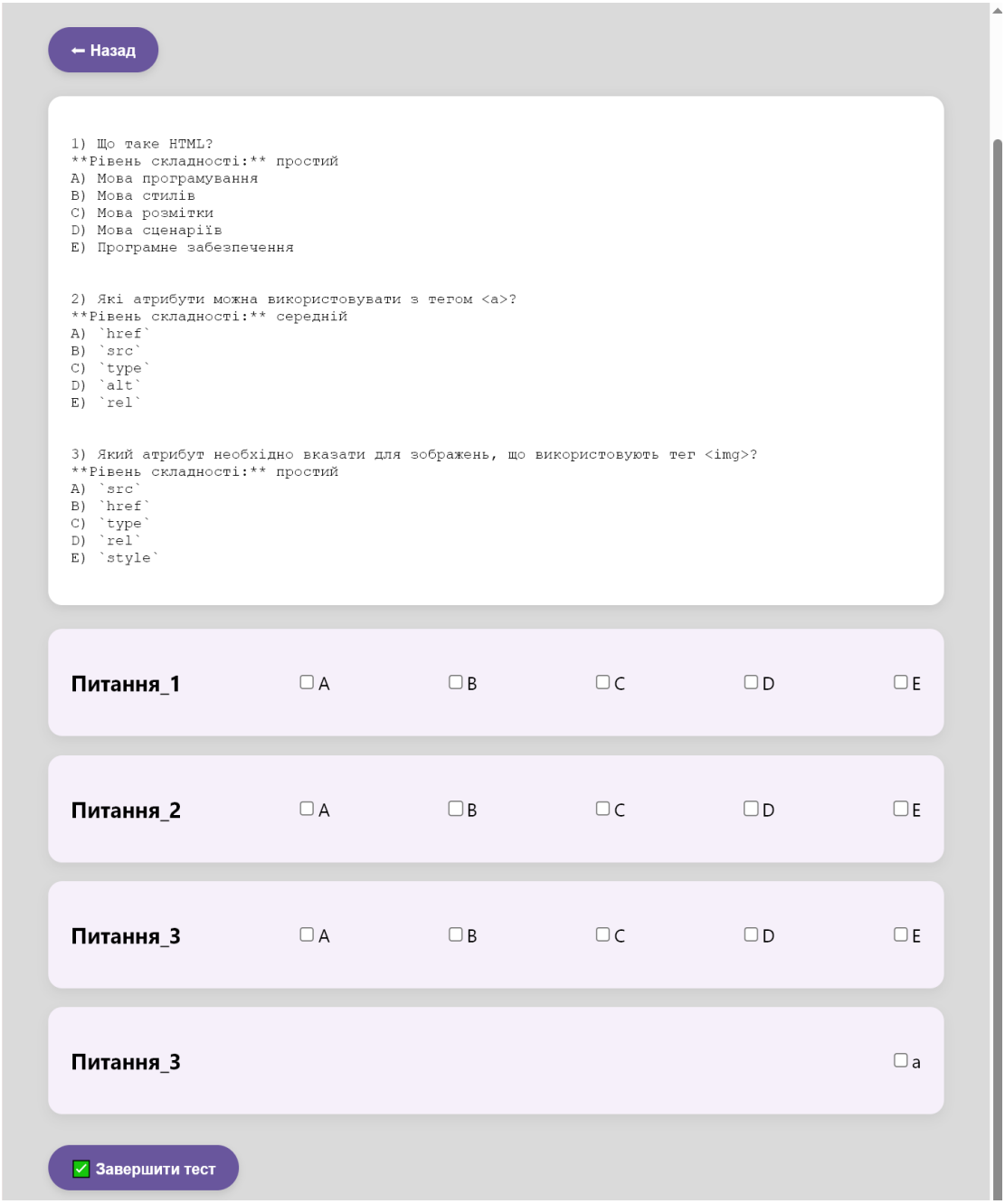

Рисунок 3.24 – Інтерфейс проходження тесту

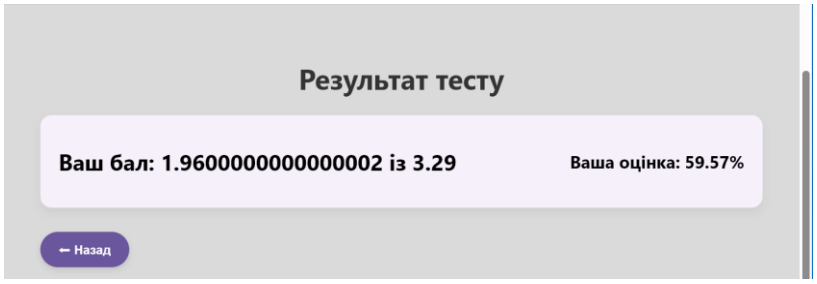

Рисунок 3.25 – Відображення результату після завершення тесту

У блоці «Повноцінні тести» викладачу, на відміну від студента, доступні додаткові функції, він може не лише переглядати тести, а й редагувати або повністю видаляти їх. Це дозволяє оперативно вносити зміни в опубліковані тести або прибирати застарілі варіанти. Для кожного тесту передбачено набір керуючих кнопок, зокрема іконки редагування, видалення та попереднього перегляду.

У секції «Збережені тести», окрім можливостей редагування, завантаження та видалення, викладач має ще одну важливу функцію – створення повноцінного тесту на основі збереженого. Це дозволяє швидко перевести згенерований тест у формат, готовий до публікації для студентів. Такий підхід спрощує роботу викладача: він може попередньо згенерувати кілька варіантів, оцінити якість і лише після цього опублікувати найкращий.

Окрему частину інтерфейсу займає менеджер файлів (рисунок 3.26), де викладач може завантажити навчальні матеріали (наприклад, лекції у форматі PDF), щоб студенти мали змогу використовувати їх для генерації тестів і словників.

Завантажені файли автоматично стають доступними всім користувачам курсу, і їх можна обрати для подальшої обробки у відповідному блоці генерації (рисунок 3.27).

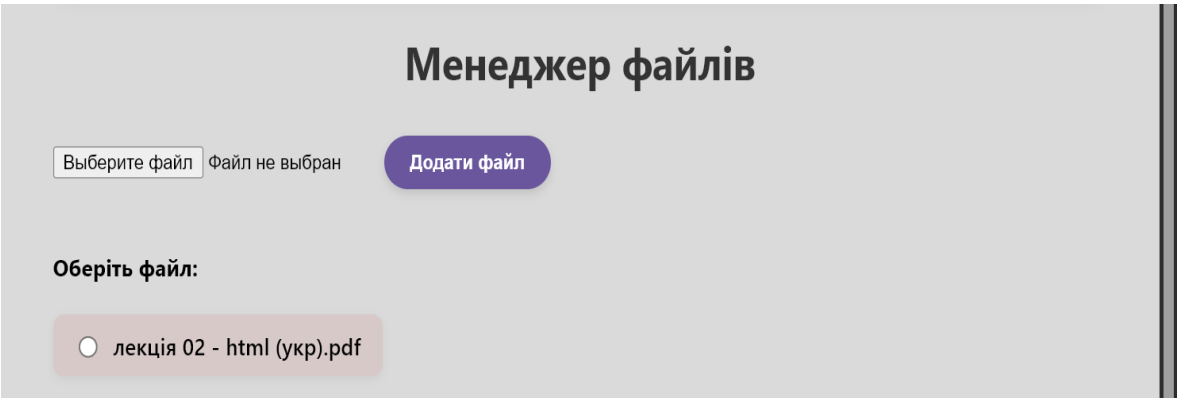

Рисунок 3.26 – Менеджер файлів викладача з можливістю додавання навчальних матеріалів

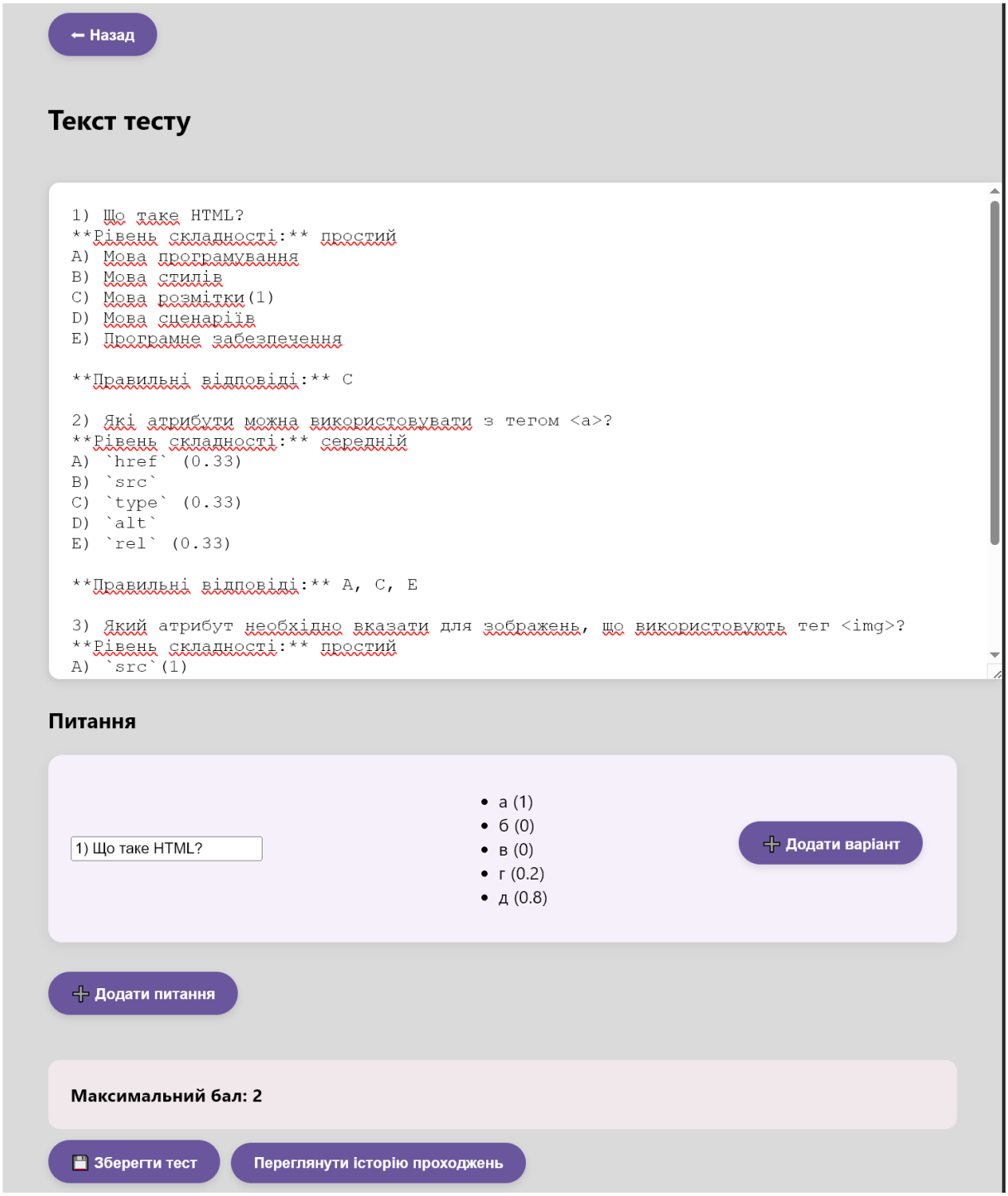

Рисунок 3.27 – Інтерфейс створення та редагування повноцінного тесту

Додатковою функцією, доступною викладачу, є перегляд історії проходжень опублікованих тестів. У спеціальному вікні відображаються всі спроби студентів із зазначенням імені користувача, адреси електронної пошти, отриманого балу та дати проходження [11] (рисунок 3.28). Кожен

запис супроводжується кнопкою видалення, що дозволяє викладачу при потребі очищати історію або прибирати окремі результати з аналізу.

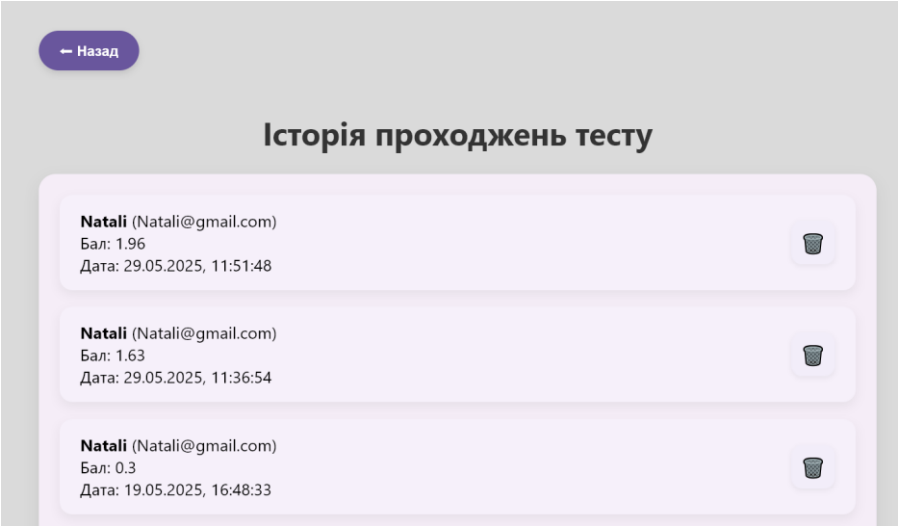

Рисунок 3.28 – Менеджер файлів викладача з можливістю додавання навчальних матеріалів

Підсумовуючи, інтерфейс клієнтської частини платформи розроблено так, щоб він був зручним, зрозумілим і логічним для кожного користувача – як для студента, так і для викладача. Завдяки чітко продуманому поділу ролей, кожен бачить саме ті функції, які йому потрібні: студенти можуть зосередитись на навчанні, проходженні тестів і підготовці, а викладачі – керувати матеріалами, створювати завдання, аналізувати результати. Усе побудовано так, щоб робота із системою не викликала зайвих труднощів і не відволікала від головного – якісного навчального процесу.

3.7 Тестування системи

Після завершення розробки функціональних компонентів системи було проведено тестування ключових можливостей, що охоплюють генерацію, збереження, експорт та імпорт навчальних матеріалів. Основна мета тестування – переконатися, що система працює стабільно, а результат

генерації є коректним і придатним для подальшого використання в освітньому середовищі.

На першому етапі перевірено генерацію тестових завдань та словників на основі текстових матеріалів. Користувач завантажує файл (наприклад, лекцію у форматі PDF) та вибирає файл з якого хочу зробити тест після чого активуються поля генерації. При створенні тесту можна задати кількість питань і рівень складності, а результат виводиться у текстовому форматі з позначенням правильних відповідей та їхніх ваг (рисунок 3.29).

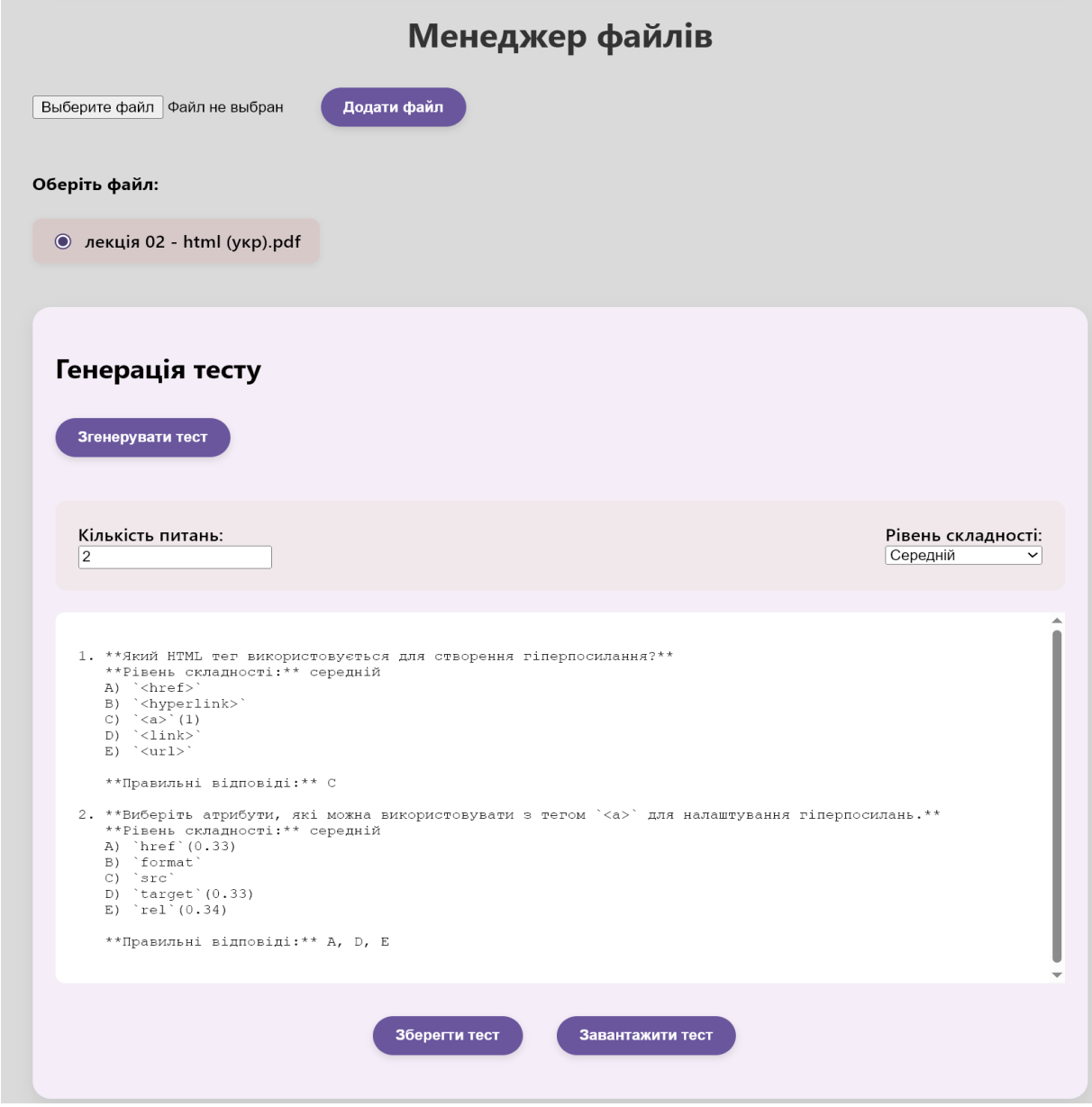

Рисунок 3.29 – Приклад згенерованого тесту з параметрами

Аналогічно реалізовано генерацію тематичного словника. На основі поданого тексту система автоматично формує список термінів із короткими визначеннями (рисунок 3.30).

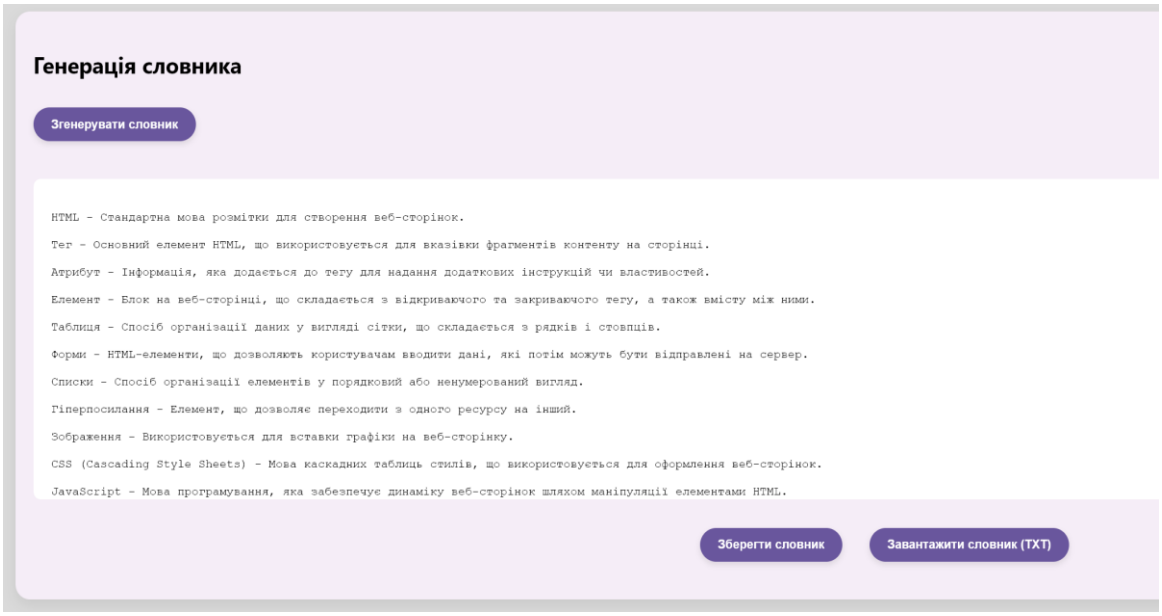

Рисунок 3.30 – Приклад згенерованого словника

Далі результати генерації можно зберегти на платформі або у форматах .txt і .xml (рисунок 3.31).

Рисунок 3.31 – Приклад згенерованого XML документа(тест)

Особлива увага приділялась структурі XML-документів, оскільки їх передбачалося використовувати для імпорту в Moodle – популярну систему дистанційного навчання (рисунок 3.32).

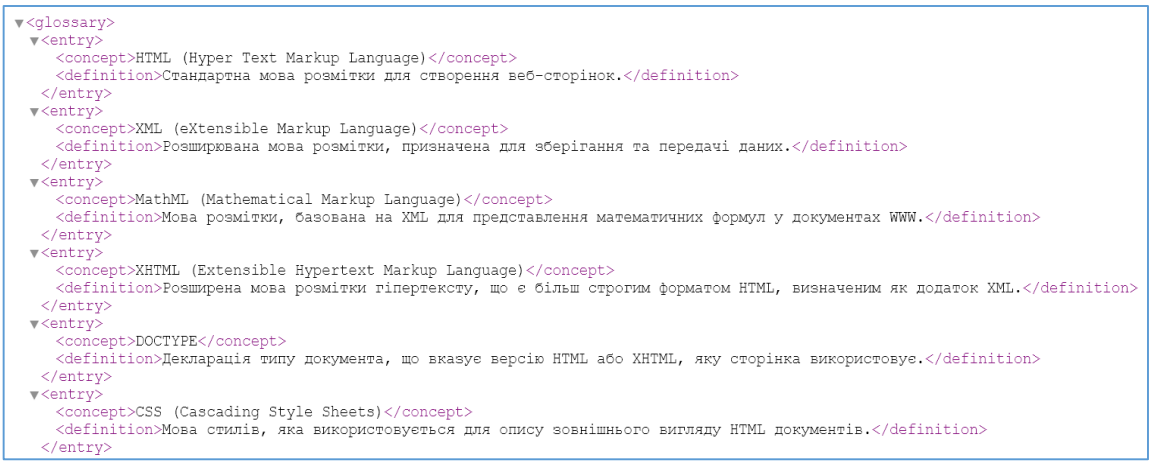

Рисунок 3.32 – Приклад згенерованого XML документа (словник)

Для перевірки сумісності згенерованих тестових завдань зі стандартами електронного навчання було проведено тестування у середовищі Moodle Sandbox 5.0, що є офіційною демонстраційною версією системи управління навчанням Moodle. Ця платформа дозволяє симулювати роботу повноцінного навчального курсу без необхідності локального розгортання серверу, що особливо зручно для цілей експериментального аналізу. Moodle підтримує імпорт структурованих тестів у власному форматі XML, що дозволяє легко інтегрувати зовнішні інструменти генерації завдань у навчальні процеси [12].

Було створено тестовий курс, до якого через інтерфейс банку запитань було імпортовано XML-файл, згенерований розробленою системою. Під час імпортування система автоматично здійснює валідацію структури файлу, типів питань і відповідей. На рисунках 3.33 – 3.34 показано процес успішного імпорту тестових запитань. Moodle коректно розпізнав типи завдань, варіанти відповідей, параметри одно- або множинного вибору, а також правильні значення. Це підтверджує відповідність структури файлу

вимогам формату Moodle XML format та придатність згенерованих даних для використання в освітньому середовищі.

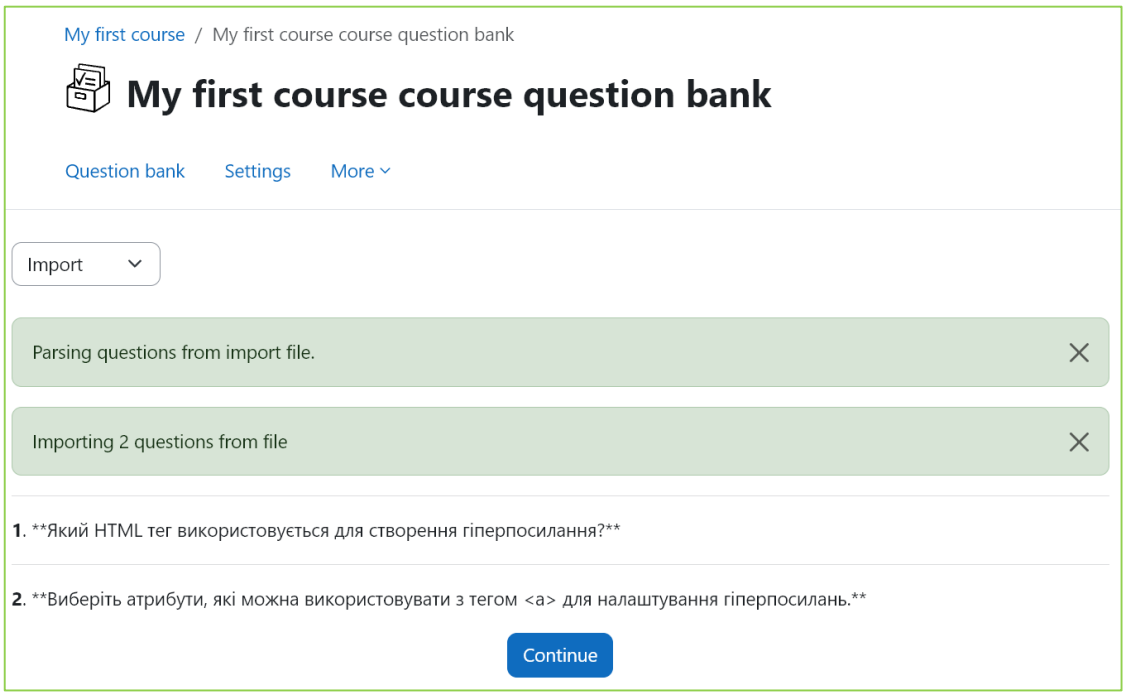

Рисунок 3.33 – Успішний імпорт тестових запитань

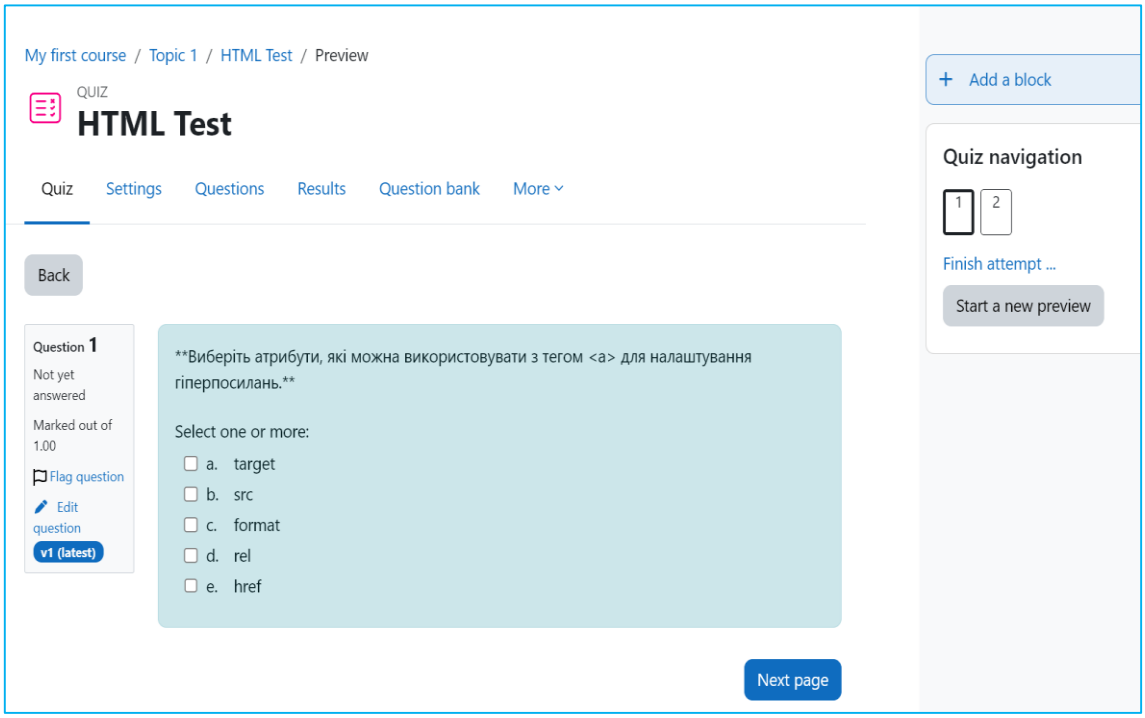

Рисунок 3.34 – Попередній перегляд імпортованого тесту в Moodle

Результати тестування підтвердили головну функціональну перевагу розробленої системи – можливість автоматично генерувати тестові завдання та тематичні словники на основі текстових навчальних матеріалів. Система забезпечує зручний інтерфейс, у якому користувач може завантажити документ, задати параметри генерації, отримати готовий матеріал у зручному форматі та за потреби зберегти його для подальшого використання.

Згенеровані тести й словники можуть бути використані як у паперовому вигляді, так і в електронних системах дистанційного навчання. Для перевірки цього було здійснено імпорт згенерованих XML-файлів до середовища Moodle Sandbox 5.0, яке успішно розпізнало структуру тестів та коректно їх відобразило. Це свідчить про сумісність формату й підтверджує можливість інтеграції системи у навчальний процес.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було проведено комплексне дослідження можливостей застосування великих мовних моделей для автоматизованої генерації тестових завдань на основі навчальних матеріалів. Аналіз сучасного стану освіти та існуючих інструментів контролю знань показав, що традиційні методи створення тестів є трудомісткими, маломасштабованими та не завжди здатні адаптуватися до швидких змін у навчальних програмах. В умовах зростання обсягів навчальної інформації та переходу до дистанційних і змішаних форматів навчання проблема ефективного й об’єктивного оцінювання знань набуває особливої актуальності [13].

Розроблена в межах цієї роботи система автоматизованого створення тестових завдань із використанням LLM дозволяє значно спростити процес підготовки тестів, зменшити навантаження на викладачів і забезпечити більш об’єктивне оцінювання знань студентів. Система здатна автоматично аналізувати навчальні матеріали, формувати різні тестові запитання із варіантами відповідей, а також створювати глосарії ключових термінів. Завдяки цьому забезпечується індивідуалізація навчального процесу, можливість адаптації завдань до рівня підготовки конкретного студента та підвищення мотивації до навчання.

Особливу увагу в роботі приділено питанням якості автоматично згенерованих тестів. Було визначено основні критерії, яким мають відповідати тестові завдання: відповідність педагогічним стандартам, точність, зрозумілість, відсутність двозначностей і коректність формулювань. Проведене тестування системи показало, що використання LLM дозволяє досягти високого рівня релевантності та різноманітності запитань, а також формувати правдоподібні варіанти відповідей, що підвищує об’єктивність оцінювання.

Впровадження подібних рішень у навчальний процес відкриває нові можливості для масштабування освітніх платформ, оптимізації роботи викладачів і підвищення якості контролю знань [14]. Автоматизовані системи на основі LLM можуть бути використані як у традиційних, так і в дистанційних чи змішаних формах навчання, забезпечуючи гнучкість і адаптивність до різних освітніх сценаріїв. При цьому роль викладача не нівелюється, а трансформується, він отримує можливість зосередитися на методичній, аналітичній роботі, інтерпретації результатів тестування та наданні індивідуальної підтримки студентам.

Незважаючи на значний потенціал, впровадження систем автоматизованої генерації тестів із використанням LLM супроводжується низкою викликів. Серед них – необхідність забезпечення якості й достовірності згенерованих завдань, уникнення надмірної стандартизації, врахування етичних аспектів і захисту даних. Для подолання цих викликів важливо поєднувати автоматизовані інструменти з експертною оцінкою викладача, а також постійно вдосконалювати алгоритми генерації тестів.

Проведене дослідження підтвердило доцільність використання великих мовних моделей для автоматизації створення тестових завдань у сучасній освіті. Запропонована система може стати основою для подальшого розвитку інноваційних освітніх технологій, сприяти підвищенню якості навчального процесу та формуванню конкурентоспроможних фахівців у цифрову епоху. Перспективними напрямами подальших досліджень є розширення функціональності системи, інтеграція з іншими освітніми платформами, а також удосконалення методів адаптивного оцінювання знань із урахуванням індивідуальних потреб студентів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кравченко О. О. Використання штучного інтелекту для автоматизації оцінювання знань студентів. *Комп'ютер у школі та сім'ї*. 2023. № 1. С. 12–17.

2. API Documentation: Chat Completions. *OpenAI*. URL: https://platform.openai.com/docs/api-reference/chat/completions (дата звернення: 23.05.2025).

3. API Documentation: Chat Completions. *Perplexity AI*. URL: https://docs.perplexity.ai/api-reference/chat-completions (дата звернення: 23.06.2025).

4. Artificial Intelligence and Education: Guidance for Policy-makers 2022. *UNESCO*. URL: https://unesdoc.unesco.org/ark:/48223/pf0000367823 (дата звернення: 24.05.2025).

5. Fillout | Forms that do it all. *Fillout | Forms that do it all*. URL: https://www.fillout.com/ (дата звернення: 24.05.2025).

6. Free Test Generator - easily create an online quiz. *Free Test Generator - easily create an online quiz*. URL: https://www.flexiquiz.com/ (дата звернення: 24.05.2025).

7. Гуржи, Т. С., Гриньова, О. Є. Генерація тестових завдань з текстових навчальних матеріалів. *29-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті»*. Зб. матеріалів форуму. Т.6., Харків: ХНУРЕ. 2025. С. 21–23. URL: https://openarchive.nure.ua/entities/publication/003f6f00-3cb0-4019-8191-7f889ed9cf8a (дата звернення: 01.06.2025).

8. ASP.NET documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core (дата звернення: 26.05.2025).

9. Overview of Entity Framework Core - EF Core. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 26.05.2025).

10. React. *React*. URL: https://reactjs.org/ (дата звернення: 03.06.2025).

11. Baker R. S., Inventado P. S. Educational Data Mining and Learning Analytics. *Learning Analytics*. Springer, 2014. P. 61–75.

12. Moodle XML format - MoodleDocs. *MoodleDocs*. URL: https://docs.moodle.org/401/en/Moodle_XML_format (дата звернення: 03.06.2025).

13. Al-Okaily M., Bataineh A., Abu-Shanab E. AI in Education: Benefits and Challenges. *International Journal of Emerging Technologies in Learning*. 2021. Vol. 16, No. 14. P. 58–76.

14. Hurzhy T., Hrynova O. Leveraging LLMS for automated test and glossary generation in education. *Proceedings of the International Workshop "Chatbot, Gaming & AI Techniques Applied in Student Digital Education"* (May, 12-13, 2025, Uzhhorod, Ukraine). Uzhhorod: Uzhhorod National University, 2025. 28–32p. URL: https://dspace.uzhnu.edu.ua/jspui/handle/lib/74262 (дата звернення: 03.06.2025).