ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

РОЗРОБКА WEB-СЕРВІСУ ДЛЯ ІНТЕРАКТИВНОГО ОЦІНЮВАННЯ СТУДЕНТІВ З ВИКОРИСТАННЯМ NLP-МОДЕЛЕЙ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Сіроштан Владислав Олександрович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Черненко О. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 70 с., 9 рис., 1 таблиця, 1 додаток, 10 джерел.

AI-ОЦІНЮВАННЯ, NLP, СЕМАНТИЧНА СХОЖІСТЬ, ОПИТУВАННЯ, WEB-СИСТЕМА

Об’єктом розробки є веб-система опитувань, що забезпечує збір відповідей студентів, автоматизовану перевірку результатів та інтегроване оцінювання відкритих текстових відповідей.

Предметом розробки є програмна реалізація модулю семантичного аналізу, який визначає рівень відповідності відповіді студента еталонним варіантам, а також інтерфейс системи, що дозволяє керувати опитуваннями, переглядати результати та аналізувати якість навчального процесу.

Метою роботи є створення інтелектуальної веб-системи, яка поєднує класичну інфраструктуру опитувань із сучасним AI-механізмом автоматичного оцінювання відповідей студентів за допомогою алгоритмів обробки природної мови (NLP) та моделей ембедінгів.

Результатом роботи стала розробка повноцінної системи Testing System, що включає модулі:

- створення та керування опитуваннями;
- проходження опитувань студентами;
- перегляд результатів та аналітики;
- автоматичне AI-оцінювання відкритих відповідей.

AI-модуль побудований на основі SentenceTransformer (SBERT), виконує попередню обробку тексту, формування векторних представлень та обчислення косинусної семантичної схожості між відповіддю студента й еталонними відповідями. Значення схожості перетворюється на числовий бал, а система формує пояснювальний коментар для користувача.

Веб-система підтримує детальний розбір кожного питання, демонструє відповідь студента, оцінку AI-модуля, кількість набраних балів та формує загальний підсумок проходження тесту. Реалізовано візуалізацію результатів: підсвічування

рівня успішності, відсоток правильних відповідей, історію проходжень та детальні показники якості.

Інтерфейс системи містить такі основні розділи:

- Опитування - перегляд доступних анкет, керування тестами;
- Проходження опитування - заповнення форми, надання відповідей різних типів;
- Пройдені опитування - історія, бали, індикатори успішності;
- Аналіз результатів - детальний перегляд відповідей, оцінок та коментарів AI.

AI-оцінювання дозволяє:

- визначати рівень семантичної відповідності;
- коректно оцінювати перефразовані відповіді;
- зменшувати суб’єктивність ручної перевірки;
- суттєво скоротити час перевірки відкритих запитань.

Система протестована на реальних даних: перевірка показала стабільність роботи, коректність семантичного аналізу та високу точність оцінювання відповідей. Інтерфейс продемонстрував зручність для студентів та викладачів, а модуль AI - здатність обробляти великі обсяги відкритих відповідей у реальному часі.

Розроблена Testing System може бути використана у закладах освіти для автоматизації опитувань, підвищення якості збору даних, пришвидшення перевірки відкритих відповідей та формування об’єктивної оцінки знань студентів.

ЗМІСТ

ВСТУП 6
1. ПОСТАНОВКА ЗАДАЧІ 8
2. ІНФОРМАЦІЙНИЙ ОГЛЯД 10
2.1. Сучасні підходи до онлайн-оцінювання студентів 10
2.2. Методи обробки текстових відповідей та NLP-алгоритми 11
2.3. Огляд існуючих освітніх платформ та систем тестування 13
3. ТЕОРЕТИЧНА ЧАСТИНА 18
3.1. Архітектура системи автоматичного оцінювання 18
3.2. Алгоритм визначення семантичної схожості 19
3.3. Метрики оцінювання відповідей 21
3.4. Концептуальна схема роботи ІІ-модуля 24
4. ПРАКТИЧНА ЧАСТИНА 28
4.1. Програмна реалізація 28
4.2. Реалізація AI-сервісу 35
4.3. Система оцінювання відповідей і логіка ШІ-аналізу 38
4.4. Тестування роботи системи 41
4.5. Інструкція для користувача 44
ВИСНОВКИ 49
СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 51
ДОДАТОК А. 52

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
AI	Artificial Intelligence - штучний інтелект
NLP	Natural Language Processing - обробка природної мови
SBERT	Sentence-BERT - модель для побудови ембедінгів речень
Embedding (ембедінг)	Векторне представлення тексту у багатовимірному просторі
Semantic Similarity	Семантична схожість між текстами
FastAPI	Фреймворк Python для створення високошвидкісних API
API	Application Programming Interface - програмний інтерфейс
UI	User Interface - інтерфейс користувача
JSON	JavaScript Object Notation - формат даних
Endpoint	Шлях API, який приймає та обробляє запити
Dataset	Набір текстових даних для навчання/тестування
Tokenization	Розбиття тексту на токени (слова, символи)
Cosine Similarity	Косинусна міра - метод обчислення схожості між векторами
Model Inference	Процес отримання результату роботи моделі
Frontend	Клієнтська частина веб-додатку

ВСТУП

Сучасний розвиток інформаційних технологій зумовлює трансформацію традиційних підходів до організації освітнього процесу. Значне поширення дистанційних та змішаних форм навчання, зростання кількості студентів та збільшення навчального навантаження на викладачів створюють необхідність у впровадженні автоматизованих систем контролю знань. Особливу складність становить оцінювання відкритих відповідей, яке потребує багато часу, високої уваги та об’єктивності з боку викладача. У таких умовах інтеграція технологій штучного інтелекту у процес оцінювання стає одним з найбільш перспективних та актуальних напрямів розвитку освітніх платформ.

Використання алгоритмів обробки природної мови (NLP) дозволяє виконувати глибинний аналіз змісту відповіді студента та оцінювати її відповідність еталонному варіанту за змістовими, семантичними та логічними характеристиками. На відміну від тестових завдань із фіксованими варіантами відповідей, відкриті питання забезпечують набагато ширший діапазон демонстрації знань, однак саме їх перевірка є найбільш трудомісткою. Застосування NLP-моделей дає можливість значно оптимізувати цей процес, підвищити об’єктивність оцінювання та зменшити людський фактор.

Розробка web-сервісу для автоматичного оцінювання відкритих відповідей студентів має важливе практичне значення. По-перше, вона дозволяє автоматизувати рутинні процеси перевірки, скорочуючи час, необхідний викладачу для виставлення оцінок. По-друге, система забезпечує однакові критерії оцінювання для всіх студентів незалежно від обсягів групи та навантаження викладача. По-третє, використання сучасних моделей семантичної подібності, основаних на методах глибинного навчання, забезпечує високу точність аналізу відповідей українською мовою, що є важливим аспектом для вітчизняних закладів освіти.

У рамках даної дипломної роботи розроблено інтерактивний web-сервіс, який поєднує механізми традиційного тестування з інтелектуальним модулем автоматичної перевірки відкритих відповідей. Система складається з клієнтської частини, backend-сервера та окремого AI-модуля, побудованого на базі сучасних

NLP-технологій. Програмний комплекс дозволяє викладачу створювати опитування, додавати питання різних типів, визначати еталонні відповіді, збирати результати та переглядати детальну аналітику. Студент отримує можливість проходити тести у зручному форматі та одразу переглядати результати, у тому числі оцінені штучним інтелектом.

Актуальність теми полягає у необхідності підвищення ефективності системи освітнього оцінювання, спрощення роботи викладача та впровадження інноваційних технологій у навчальний процес. Використання NLP-моделей дозволяє розв’язати проблему об'єктивного, швидкого та масштабованого оцінювання, що робить розроблений сервіс значущим для університетів, коледжів, освітніх платформ та корпоративних навчальних систем.

Метою дипломної роботи є створення web-сервісу для інтерактивного оцінювання студентів з використанням NLP-моделей для автоматичної перевірки відкритих відповідей. Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати існуючі методи оцінювання; дослідити можливості NLP-технологій; розробити архітектуру системи; реалізувати модулі бекенду, фронтенду та AI-сервісу; забезпечити інтеграцію компонентів; виконати тестування та оцінювання точності роботи моделі.

Об’єктом дослідження є процес оцінювання відкритих текстових відповідей студентів у системах онлайн-тестування.

Предметом дослідження є методи та алгоритми автоматичного аналізу та оцінювання текстових відповідей на основі NLP-моделей, а також програмні засоби їх реалізації у web-середовищі.

Результатом реалізації проєкту є програмний комплекс, що поєднує сучасні web-технології та алгоритми штучного інтелекту, забезпечуючи автоматизовану, об’єктивну та ефективну систему контролю знань студентів. Отримані результати можуть бути застосовані у навчальних закладах, під час онлайн-курсів, а також у корпоративних програмах підвищення кваліфікації.

1. ПОСТАНОВКА ЗАДАЧІ

Процес контролю знань студентів традиційно передбачає проведення тестування, перевірку відповідей та виставлення оцінок. Найбільш трудомістким та ресурсоємним етапом є перевірка відкритих відповідей, які не можуть бути автоматично оцінені за допомогою стандартних тестових систем. Викладачу необхідно аналізувати зміст кожної відповіді, співвідносити її з еталонним матеріалом та приймати рішення щодо рівня засвоєння теми студентом. За великої кількості студентів або обмеженого часу такий підхід знижує ефективність навчального процесу та створює ризики суб’єктивності оцінювання.

Водночас сучасні технології обробки природної мови дають можливість автоматизувати аналіз текстових відповідей, визначати їхню семантичну схожість з еталонними варіантами та формувати об’єктивну оцінку. Таким чином, виникає необхідність у розробці спеціалізованої web-платформи, що забезпечує створення тестів, збір відповідей, їх автоматичне оцінювання та надання аналітичної інформації викладачу.

Основна задача полягає у створенні програмного комплексу, що поєднує класичні засоби онлайн-тестування з модулем автоматичного оцінювання відкритих відповідей на основі NLP-моделей. Система повинна бути інтерактивною, зручною для студентів і викладачів, а також забезпечувати високий рівень точності оцінювання відповідей українською мовою.

Для досягнення поставленої мети необхідно вирішити такі підзадачі:

- проаналізувати існуючі підходи та інструменти онлайн-оцінювання, включаючи платформи з ручною та автоматизованою перевіркою відповідей;
- вивчити NLP-алгоритми для визначення семантичної близькості текстів, зокрема методи лематизації, векторизації та обчислення косинусної подібності;
- розробити архітектуру web-сервісу, що включає фронтенд-клієнт, серверну частину та окремий AI-мікросервіс;
- створити механізм формування тестів і збору відповідей, який дозволяє

використовувати як тестові, так і відкриті питання;

- реалізувати модуль автоматичного оцінювання відповідей на основі Sentence-Transformers та інших NLP-моделей;
- забезпечити інтеграцію модулів системи, включаючи передачу відповідей у чергу, їх обробку AI-сервісом та повернення оцінок;
- розробити інтерфейс викладача для перегляду результатів, аналізу метрик та управління тестами;
- провести тестування системи, перевіривши її коректність, стабільність, точність оцінювання та зручність використання.

Таким чином, постановка задачі передбачає створення комплексного web-сервісу, здатного забезпечувати повний цикл тестування студентів з автоматичним аналізом відповідей. Результуючий продукт має полегшити роботу викладача, підвищити об’єктивність оцінювання та забезпечити масштабованість процесу контролю знань.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Сучасні підходи до онлайн-оцінювання студентів

Зростання популярності дистанційного та змішаного навчання призвело до широкого впровадження платформ електронного тестування, які забезпечують оперативний збір відповідей, автоматизацію виставлення оцінок та аналіз результатів. Системи онлайн-оцінювання стали важливою складовою цифрової трансформації освіти, оскільки дозволяють підвищити ефективність контролю знань, забезпечити масштабованість навчального процесу й спростити взаємодію між викладачами та студентами.

Традиційні онлайн-системи оцінювання, такі як Moodle, Google Forms, Canvas та інші LMS-платформи, здебільшого використовують формальні типи завдань: тестові питання з вибором варіанту, питання на встановлення відповідності, впорядкування та числові відповіді. Основна перевага таких завдань полягає у можливості повної автоматизації процесу перевірки, що робить їх придатними для швидкого оцінювання великих груп студентів. Проте цей підхід має суттєве обмеження - він не дозволяє оцінити глибину розуміння матеріалу, здатність студента формулювати власні думки та застосовувати знання у відкритій формі.

У зв’язку з цим освітні системи все частіше застосовують змішану модель контролю: поєднання тестових запитань із завданнями відкритого типу. Відкриті відповіді (есе, пояснення, обґрунтування, опис алгоритму тощо) надають значно ширшу інформацію про рівень підготовки студента, однак їх перевірка традиційно виконується вручну. Такий процес є трудомістким, особливо у великих навчальних групах, і часто супроводжується суб’єктивністю оцінювання, пов’язаною з людським фактором.

Сучасні підходи до онлайн-оцінювання передбачають використання алгоритмів машинного навчання для аналізу текстових відповідей. Широке застосування отримали методи автоматичної класифікації текстів, моделі визначення ключових понять, тематичного групування та аналізу семантичної схожості. Це дозволяє оцінювати не лише факт наявності або відсутності певних

слів у відповіді, але й її змістовну відповідність еталонному матеріалу.

Найбільш прогресивні освітні платформи інтегрують моделі обробки природної мови (NLP), що забезпечують аналіз структури відповіді, логічності викладення, використання термінології та ступеня розкриття теми. Завдяки цьому з’являється можливість автоматично оцінювати відкриті відповіді, які раніше вимагали ручної перевірки викладачем. Застосування моделей на основі трансформерів, таких як BERT, RoBERTa, GPT-подібні архітектури, дозволяє визначати семантичну схожість між відповіддю студента та еталонним текстом, що робить процес оцінювання об’єктивним та масштабованим. [1-2]

Таким чином, сучасні підходи до онлайн-оцінювання студентів охоплюють широкий спектр методів: від автоматизованих тестів до інтелектуальних систем аналізу відкритих відповідей. Використання NLP-моделей стає ключовим напрямом розвитку освітніх платформ, оскільки дозволяє поєднати високу швидкість автоматизації з глибинним змістовим аналізом, забезпечуючи нову якість контролю знань у цифровому навчальному середовищі.

2.2. Методи обробки текстових відповідей та NLP-алгоритми

Обробка відкритих текстових відповідей студентів потребує використання спеціалізованих методів аналізу природної мови (NLP - Natural Language Processing). Такі методи забезпечують можливість інтерпретації структури тексту, визначення його змістових характеристик та порівняння з еталонними відповідями. На відміну від формальних тестів, відкриті відповіді мають довільну структуру та різну довжину, можуть містити синоніми, граматичні варіації та індивідуальний стиль викладу. Тому їх обробка потребує застосування комплексних NLP-алгоритмів.

Першим етапом аналізу є нормалізація тексту, що включає перетворення всіх символів до нижнього регістру, видалення зайвих розділових знаків, очищення від спеціальних символів та приведення тексту до уніфікованого вигляду. Цей етап дозволяє зменшити вплив стилістичних особливостей студента на подальший

аналіз.

Наступним кроком є токенізація - поділ тексту на окремі слова або фрази. Для української мови цей процес має свої особливості, оскільки вона характеризується багатою морфологією, змінністю закінчень та складною системою відмінювання.

Для коректного порівняння відповідей застосовується лематизація - приведення слів до їх базової (словникової) форми. Використання інструментів на кшталт *Stanza* дозволяє з високою точністю виконувати морфологічний аналіз українського тексту, визначаючи частини мови та леми слів. Лематизація дозволяє мінімізувати вплив граматичних формулювань та зосередитися на змістовій частині відповіді.

Після попередньої обробки текст трансформується у векторне подання за допомогою моделей семантичного кодування. Найбільш поширеним підходом є використання Sentence-Transformers - моделей, що будують семантично узгоджені векторні представлення речень та текстів. На відміну від класичних методів, таких як Bag-of-Words або TF-IDF, векторні моделі дозволяють враховувати контекст, синонімію, порядок слів та глибокі семантичні зв’язки.

На основі отриманих векторів виконується обчислення семантичної схожості між відповіддю студента та еталонною відповіддю викладача. Найчастіше використовується косинусна подібність, яка визначає кут між векторами у багатовимірному просторі та дозволяє кількісно оцінити ступінь змістової близькості двох текстів. Чим більшим є значення косинусної подібності, тим вищою є відповідність змісту відповіді студента очікуваному результату.

Крім визначення схожості, сучасні NLP-моделі також дають змогу аналізувати такі параметри, як ключові слова, релевантність термінів, логічність побудови речень, а також виявляти пропущені концепти або помилки в тексті. Це розширює можливості автоматизованих систем оцінювання, дозволяючи враховувати не лише загальну схожість, а й якість викладу. [3]

Таким чином, методи обробки текстових відповідей охоплюють комплекс послідовних алгоритмів: нормалізація, токенізація, лематизація, побудова векторних представлень та розрахунок семантичної подібності. Сукупність цих

кроків забезпечує можливість точного та об’єктивного автоматичного оцінювання відкритих відповідей студентів, що є ключовим компонентом сучасних інтелектуальних освітніх систем.

2.3. Огляд існуючих освітніх платформ та систем тестування

У сучасному освітньому середовищі існує значна кількість програмних рішень, призначених для організації онлайн-тестування, контролю знань та управління навчальним процесом. Найпоширенішими серед них є системи управління навчанням (LMS), платформи для створення тестів та інтерактивні сервіси оцінювання. Незважаючи на широкий функціонал, більшість з них зосереджені на автоматизації формальних завдань і мають обмежені можливості щодо оцінювання відкритих текстових відповідей.

Одним з найпопулярніших рішень є Moodle - відкрита LMS-платформа, яка активно використовується у навчальних закладах (див. рис. 2.1). Moodle забезпечує підтримку різних типів тестових завдань, містить систему оцінювання, журнал успішності та модулі аналітики. Проте оцінювання відкритих відповідей виконується вручну викладачем, а інструменти автоматичного аналізу обмежуються простими текстовими фільтрами.

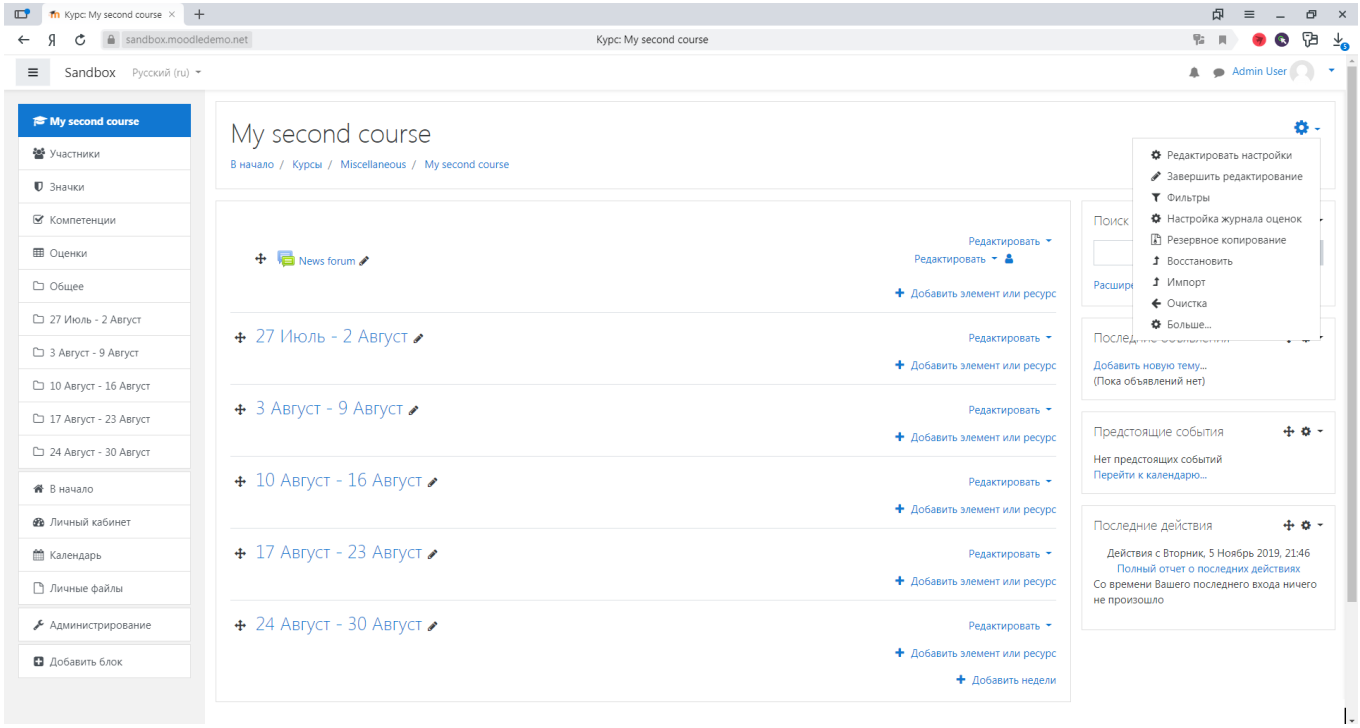

Рисунок 2.1 – Платформа «Moodle»

Платформа Google Forms є одним із найпростіших інструментів для проведення опитувань та тестів (див. рис. 2.2). Вона дозволяє швидко створювати тести з автоматичною перевіркою закритих питань, однак не підтримує автоматичну перевірку відкритих відповідей. Усі тексти, введені студентами, потребують ручної перевірки, що робить платформу малопридатною для дисциплін, де використовується аналітичний чи описовий формат відповідей.

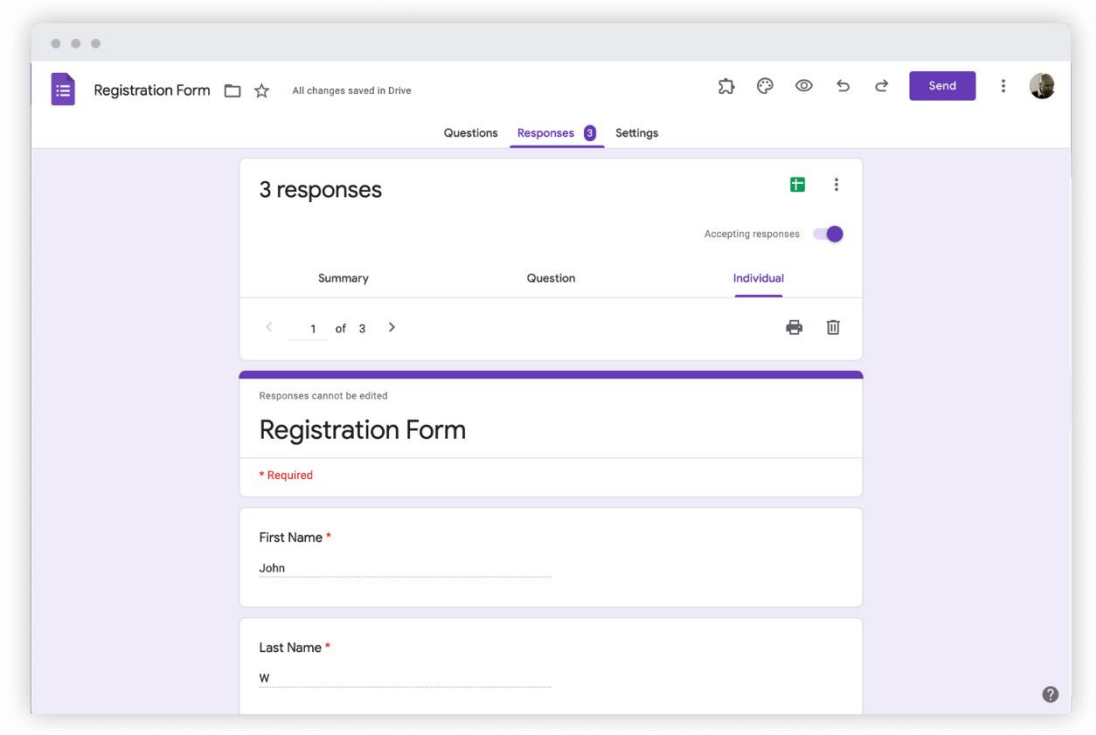

Рисунок 2.2 – Платформа «Google Forms»

Досить поширеною є також система Canvas LMS, яка використовується багатьма університетами у світі (див. рис. 2.3). Canvas пропонує інструменти створення тестів, управління курсами, спільну роботу та аналітику. Проте, як і Moodle, система не має вбудованих моделей NLP для автоматичного оцінювання відкритих відповідей, обмежуючись ручною перевіркою або сторонніми інтеграціями.

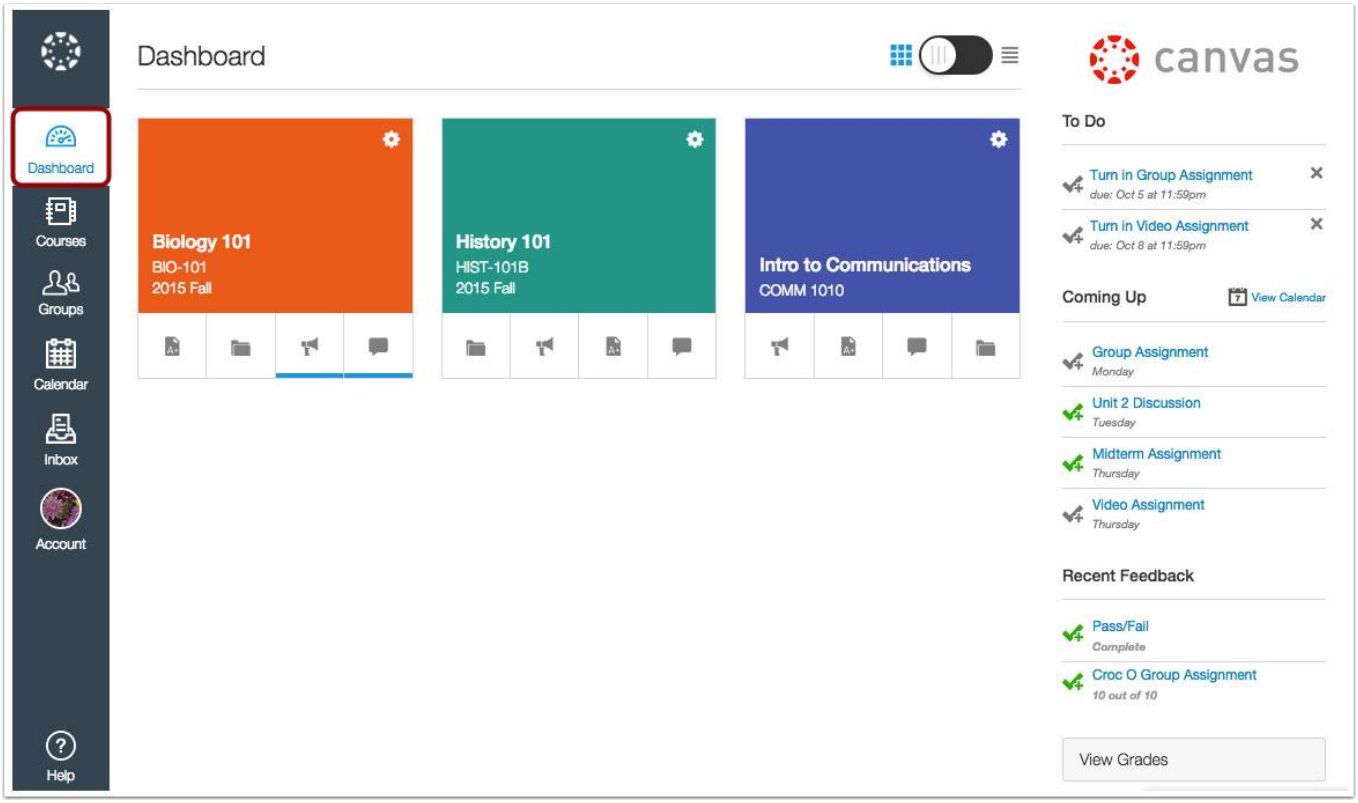

Рисунок 2.3 – Платформа «Canvas LMS»

Іншим прикладом є ClassMarker - комерційна платформа для онлайн-тестування з розширеними можливостями налаштування тестів (див. рис. 2.4). Сервіс пропонує високий рівень надійності та автоматичну перевірку закритих питань, проте також не забезпечує автоматичної семантичної оцінки відкритих відповідей, що знижує його ефективність у гуманітарних чи теоретичних дисциплінах.

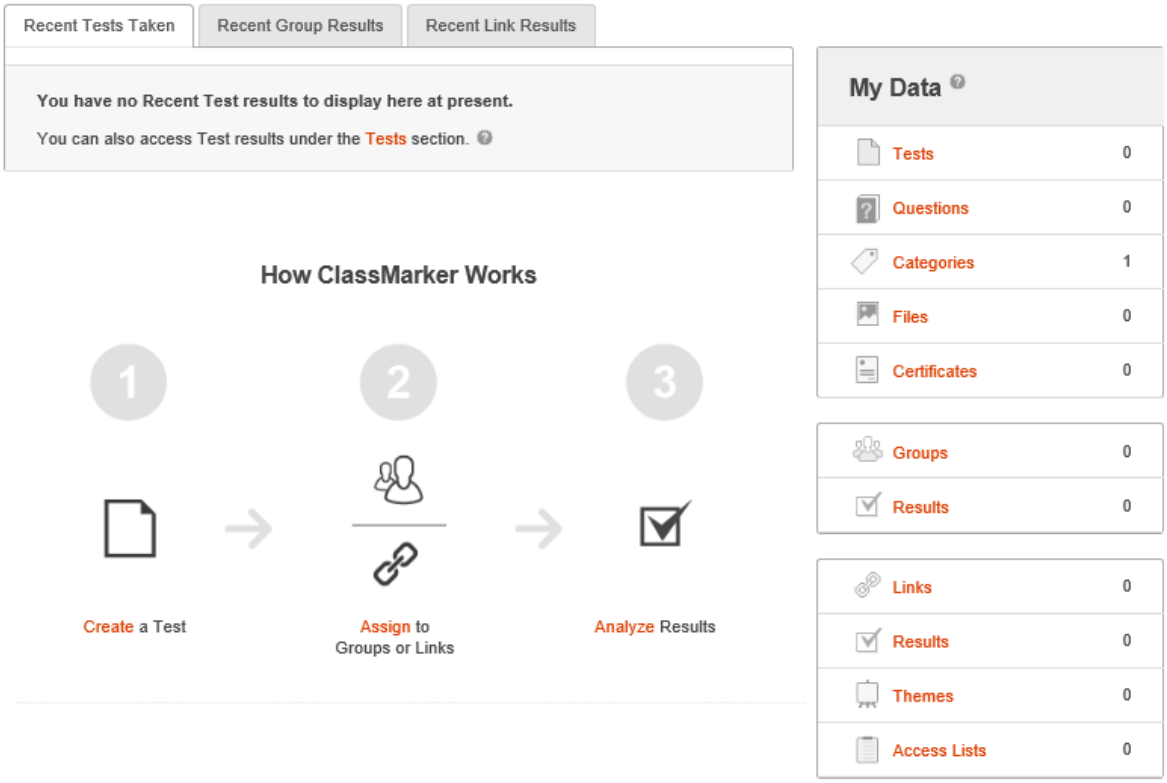

Рисунок 2.4 – Платформа «ClassMarker»

Серед інструментів для інтерактивної взаємодії зі студентами популярними є Kahoot, Quizizz та Mentimeter. Вони пропонують гейміфікацію, візуалізацію результатів та миттєвий зворотний зв’язок. Однак ці платформи орієнтовані переважно на тестові питання та не призначені для змістового аналізу текстових відповідей.

Окрему категорію становлять комерційні рішення, що використовують елементи штучного інтелекту, наприклад Gradescope від компанії Turnitin, який частково підтримує автоматичне аналізування коротких відповідей. Проте його можливості для глибинного семантичного аналізу є обмеженими та здебільшого орієнтовані на роботи програмістів, математику та структуровані відповіді.

Проведений огляд свідчить, що сучасні освітні платформи активно розвиваються у напрямку автоматизації тестування, однак у більшості рішень відсутні ефективні механізми автоматичного семантичного аналізу відкритих текстових відповідей. Саме ця особливість визначає актуальність створення спеціалізованого web-сервісу, який використовує NLP-моделі для об’єктивного та масштабованого оцінювання змістових відповідей студентів.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура системи автоматичного оцінювання

Архітектура системи автоматичного оцінювання відкритих відповідей студентів побудована за принципами мікросервісного підходу, що забезпечує гнучкість, масштабованість та незалежність окремих функціональних модулів. Система складається з трьох основних компонентів: клієнтського застосунку, серверної частини (backend) та окремого AI-мікросервісу, який виконує обчислювальні операції з аналізу текстів. Такий підхід дозволяє розділити логіку управління даними, інтерфейсну взаємодію та модулі штучного інтелекту, що суттєво спрощує розвиток та тестування системи.

Клієнтська частина реалізована на основі фреймворку Next.js, що забезпечує рендеринг інтерфейсу, обробку взаємодії користувача та відправлення запитів до серверної частини. Інтерфейс передбачає можливість створення тестів, додавання питань різних типів, проходження тестування студентами та перегляд результатів викладачем. Взаємодія клієнту з backend-сервером відбувається через REST API, що забезпечує стандартизований обмін даними. [1]

Серверна частина системи реалізована на платформі NestJS, яка забезпечує управління бізнес-логікою, аутентифікацію користувачів та роботу з базою даних. У якості сховища даних використовується MongoDB, що дозволяє ефективно працювати зі складними вкладеними структурами документів, такими як опитування, питання, відповіді та метрики оцінювання. Backend відповідає за обробку запитів користувачів, збереження результатів тестування, формування завдань для ШІ-аналізу та отримання оцінок.

Ключовим компонентом архітектури є AI-мікросервіс, побудований на основі FastAPI та Python. Він виконує основну інтелектуальну роботу - попередню обробку текстів, лематизацію, побудову векторних представлень та визначення семантичної схожості між відповіддю студента та еталонним текстом. AI-сервіс є незалежним компонентом, що дозволяє масштабувати його окремо від решти системи та оновлювати NLP-моделі без втручання у бекенд-логіку.

Для організації асинхронної взаємодії між backend-сервером та AI-модулем використовується система черг BullMQ у поєднанні з Redis. Після того як студент подає відповідь, сервер не чекає завершення аналізу, а формує завдання в черзі. AI-мікросервіс отримує ці завдання, проводить обробку та повертає результат, після чого backend оновлює відповідні записи в базі даних. Такий підхід дозволяє виконувати масштабовану чергу обчислювальних завдань, обробляти велику кількість відповідей та уникати затримок у роботі інтерфейсу.

Важливим елементом архітектури є система аутентифікації та авторизації на основі JWT-токенів, що забезпечує захист даних користувачів і контроль доступу до функцій системи. Студенти отримують доступ лише до проходження тестів, тоді як викладачі можуть створювати опитування, переглядати результати та керувати оцінюванням.

Узагальнено архітектура системи включає такі компоненти:

- клієнтський модуль (Next.js) - інтерфейс взаємодії студентів і викладачів;
- backend-модуль (NestJS + MongoDB) - логіка управління тестами, відповідями, користувачами та чергами; [4]
- AI-мікросервіс (FastAPI + Sentence-Transformers) - обробка текстів і визначення семантичної схожості;
- Redis + BullMQ - механізм асинхронної взаємодії між компонентами.

Така архітектура забезпечує модульність, можливість незалежного оновлення та масштабування компонентів, а також високу продуктивність системи при великих обсягах даних. Розділення функціональних блоків дозволяє оптимізувати роботу ШІ-модуля, підтримувати сучасні NLP-алгоритми та забезпечувати стабільний процес автоматичного оцінювання відповідей студентів.

3.2. Алгоритм визначення семантичної схожості

Алгоритм визначення семантичної схожості текстових відповідей є ключовим компонентом системи автоматичного оцінювання. Його завдання полягає у тому, щоб визначити ступінь відповідності між відповіддю студента та еталонною відповіддю викладача не лише на рівні окремих слів, а на рівні змісту, логіки та

використовуваних понять. На відміну від простого порівняння текстів за лексичними збігами, семантичний аналіз дозволяє врахувати синонімію, перефразування, відмінності у граматичних структурах та індивідуальний стиль викладу.

Алгоритм складається з послідовності кроків, кожен з яких спрямований на відокремлення інформативної частини тексту та побудову його математичного представлення. Першим етапом є **попередня обробка тексту**, яка включає перетворення тексту до нижнього регістру, видалення розділових знаків, нормалізацію пробілів та очищення від допоміжних символів. Цей етап дозволяє зменшити шум та стандартизувати текст перед подальшою обробкою.

Другим етапом є **лематизація**, що виконується з використанням NLP-модулів, адаптованих для української мови, таких як *Stanza*. Лематизація перетворює слова у їх початкову форму, що дає змогу нівелювати вплив відмінювання, чисел, родів та інших морфологічних змін. Наприклад, слова *“передають”, “передача”, “передавати”* зводяться до спільної основи, завдяки чому алгоритм може коректно оцінювати зміст відповіді, а не лише її поверхневу форму.

Після лематизації обидва тексти - відповідь студента та еталон - трансформуються у векторні представлення. Для цього використовується модель типу **Sentence-Transformers**, що створює високорозмірні семантичні вектори, які відображають контекст та зміст речення. На відміну від класичних моделей на кшталт TF-IDF або Bag-of-Words, Sentence-Transformers враховують структуру речень, синтаксис, семантичні зв’язки між словами та здатні розпізнавати перефразування. [5]

Отримані вектори порівнюються за допомогою метрики **косинусної подібності**, яка визначає кут між двома векторами у багатовимірному просторі. Значення косинусної подібності знаходиться в діапазоні від –1 до 1, де 1 означає максимальну схожість текстів за змістом. У практичному застосуванні у системах оцінювання значення нормується до шкали 0–100%, що дозволяє інтерпретувати результат як відсоткову відповідність відповіді студента еталонній відповіді

викладача.

Загальний алгоритм визначення схожості виведени у блок-схемі (див. рис. 3.1)

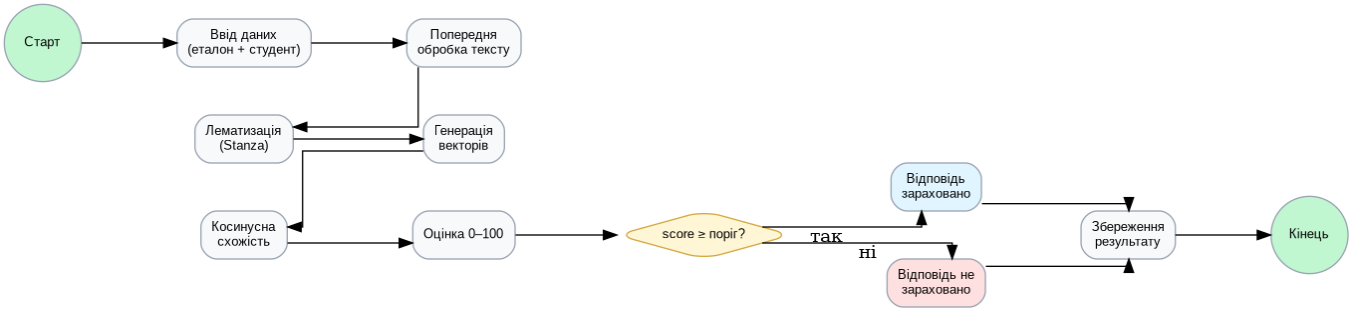

Рисунок 3.1 – Блох-схема алгоритму

Такий підхід дозволяє точно оцінювати не лише прямий збіг ключових слів, а й структуру викладення, коректність уживання термінів та глибину розкриття теми. Завдяки використанню контекстних моделей алгоритм здатний розпізнавати перефразування, логічні пояснення та різні стилі формулювання, що робить його придатним для реального освітнього середовища.

Таким чином, застосований алгоритм семантичного аналізу забезпечує об’єктивну, масштабовану та змістово орієнтовану оцінку відкритих текстових відповідей студентів, що є ключовим елементом функціонування інтелектуального модуля автоматичного оцінювання у системі.

3.3. Метрики оцінювання відповідей

Процес визначення якості розгорнутих відповідей студентів потребує не лише алгоритму семантичної схожості, а й застосування кількісних метрик, які дозволяють порівнювати результати, відслідковувати точність моделі та приймати об’єктивні рішення щодо оцінювання. Метрики відіграють ключову роль у забезпеченні прозорості, відтворюваності та надійності роботи інтелектуального модуля системи.

У проєкті реалізовано декілька груп метрик, що відображають різні аспекти якості відповіді: семантичну близькість до еталону, рівень впевненості моделі,

коректність попередньої обробки та час виконання.

Семантичні метрики

1. Косинусна схожість (Cosine Similarity)

Це основна метрика, що вимірює кут між векторними представленнями студентської та еталонної відповідей. Формально вона визначається як:

$$cos_{sin}(A, B) = \frac{A \cdot B}{\| A \| \cdot \| B \|}$$

де A і B - вектори відповідей після кодування Sentence-Transformers.

Переваги метрики:

- нечутливість до довжини тексту;
- можливість оцінювати змістовну близькість, а не лише словесну подібність;
- висока стійкість до перефразування.

У системі значення косинусної схожості масштабуються на 0–100 для отримання підсумкової оцінки. [6]

Метрики впевненості

2. Confidence Score

Показник відображає ймовірнісну впевненість моделі у визначенні правильності відповіді. Він враховує:

- стабільність векторів після лематизації,
- відстань до найближчих кластерів еталонних відповідей,
- внутрішню ентропію моделі під час кодування.

Мета показника - підсилити інтерпретованість оцінки. Наприклад:

- низький similarity, але високий confidence ⇒ відповідь змістовно інша, але модель впевнена;
- високий similarity, але низький confidence ⇒ можлива надмірна генералізація, потреба у перевірці викладачем.

Процесуальні метрики

3. Час обробки (Processing Time)

Фіксується для кожної відповіді - від моменту постановки завдання до отримання результату. Цей показник дозволяє:

- оцінювати продуктивність NLP-пайплайна;
- визначати вузькі місця;
- контролювати можливість масштабування системи.

Затримки можуть виникати на етапах:

- лематизації Stanza (CPU-bound);
- генерації векторів Sentence-Transformers (GPU/CPU-bound);
- виклику зовнішнього ШІ-сервісу.

Системні метрики

4. Статус оцінювання (Evaluation Status)

Система використовує чотири статуси перевірки:

- pending - очікує обробки;
- processing - ШІ активно аналізує відповідь;
- completed - оцінювання завершено успішно;
- failed - сталася помилка (тайм-аут, невірні дані, збій моделі).

Наявність статусів спрощує керування великими чергами оцінювання через BullMQ та Redis.

5. Порогове рішення (Threshold-Based Decision)

Після обчислення similarity система порівнює його з наперед встановленим порогом:

$$score \geq T$$

де T - мінімально допустима схожість (зазвичай 60–70).

Порогове рішення забезпечує:

- уніфікацію оцінювання;
- можливість гнучкого налаштування складності тестів;
- автоматичну класифікацію відповіді як «зараховано» або «не зараховано».

6. Підсумкова оцінка (Final Grade)

Від 0 до 100 балів, що враховує:

- raw score (косинусна схожість),
- confidence,
- вагові коефіцієнти викладача (якщо застосовано),

- складність питання.

Цей показник відображається у підсумковій інформації про тест та використовується у статистиці успішності студента.

Таким чином, система поєднує кілька класів метрик, що забезпечують точне та об’єктивне оцінювання відповідей. Комплексне застосування семантичних, процесуальних та системних показників дозволяє підвищити якість результатів, а також здійснювати контроль та оптимізацію роботи ШІ-модуля в реальних умовах експлуатації.

3.4. Концептуальна схема роботи ІІ-модуля

ІІ-модуль системи тестування виконує роль окремого сервісу оцінювання відповідей, який приймає з бекенду еталонну відповідь викладача, відповідь студента та службові параметри, а на виході повертає числову оцінку, метрики якості та службовий статус обробки. Концептуально його можна розглядати як послідовність логічних шарів, через які проходять дані від моменту запиту до збереження результату в базі даних.

1. Вхідний рівень (API-інтерфейс).

На першому етапі бекенд NestJS надсилає HTTP-запит до ІІ-модуля (FastAPI), у якому містяться:

- текст еталонної відповіді;
- текст відповіді студента;
- ідентифікатори питання, спроби та користувача;
- додаткові параметри: максимальний бал, порогове значення схожості, мова відповіді тощо.

ІІ-модуль виконує базову валідацію: перевіряє наявність усіх обов’язкових полів, формат токена доступу, допустиму довжину тексту. У разі помилки формується відповідь з кодом 4xx/5xx, а бекенд фіксує статус оцінювання як failed.

2. Рівень попередньої обробки тексту.

Після успішної валідації обидва тексти (еталонний і студентський)

передаються в підсистему попередньої обробки. На цьому рівні виконується:

- нормалізація регістру (переведення до lower case);
- очищення від пунктуації та службових символів;
- усунення зайвих пробілів;
- за потреби - фільтрація очевидного «спаму» (надто короткі чи випадкові рядки).

Мета цього етапу - усунути шум і привести обидві відповіді до уніфікованого вигляду перед семантичним аналізом.

3. NLP-рівень та лематизація.

Далі тексти надходять до NLP-підсистеми, реалізованої на базі бібліотеки Stanza. Для української мови виконується:

- токенізація тексту на слова й речення;
- визначення частин мови;
- лематизація слів (приведення до початкової форми).

Результатом є дві «очищені» та лематизовані версії відповідей, які краще відображають зміст, а не конкретну граматичну форму, що зменшує чутливість моделі до відмінювання, часу дієслів тощо.

4. Рівень семантичного кодування.

Лематизовані тексти передаються в модуль семантичного кодування на основі моделей Sentence-Transformers. Кожна відповідь перетворюється на вектор у багатовимірному просторі ознак:

- emb_ref - вектор еталонної відповіді;
- emb_student - вектор відповіді студента.

У цьому просторі близькі за змістом тексти мають розташовуватися поруч, що дозволяє кількісно вимірювати семантичну схожість.

5. Рівень обчислення метрик.

На наступному етапі працює підсистема оцінювання (evaluation engine), яка:

- обчислює косинусну схожість між векторами (cos_sim(emb_ref, emb_student)), отримуючи значення від 0 до 1;
- перетворює її на бальну шкалу 0–100 (див. підрозділ 3.3);

- оцінює confidence score (впевненість моделі) за допомогою додаткових евристик;
- фіксує час обробки запиту та службові показники (ідентифікатор моделі, версію NLP-компонентів тощо).

Таким чином формується набір метрик, які описують як сам результат оцінювання, так і якість його отримання.

6. Рівень прийняття рішення.

Отриманий бал порівнюється з пороговим значенням, заданим викладачем або системою за замовчуванням:

- якщо score ≥ T, відповідь вважається такою, що зарахована (достатньо точна або відмінна);
- якщо score < T, відповідь оцінюється як недостатньо точна (незарахована або з низьким балом).

На цьому ж рівні можуть застосовуватися додаткові правила: наприклад, округлення до певного кроку, обмеження максимального чи мінімального балу, ручна корекція з боку викладача тощо.

7. Рівень формування відповіді та логування.

Після прийняття рішення II-модуль формує структуровану відповідь для бекенду, яка містить:

- score - числову оцінку 0–100;
- grade - приведений до шкали курсова/залікова оцінка (за потреби);
- confidence - рівень впевненості моделі;
- службовий статус оцінювання (completed, failed);
- часові мітки початку й завершення обробки.

Паралельно модуль записує технічний лог: час виконання, можливі попередження, ідентифікатор моделі. Це дозволяє відстежувати продуктивність, проводити аудит та порівнювати якість різних версій моделей.

8. Інтеграція з основною системою.

На бекенді NestJS результат II-модуля:

- зберігається в колекції UserAnswers разом з метриками;

- використовується для розрахунку загального балу за тест;
- відображається студенту у вигляді відсотка успішності та текстового статусу («відмінно», «зараховано», «потребує доопрацювання» тощо);
- може бути використаний у звітності для викладача (середній бал, розподіл оцінок, проблемні питання).

При високому навантаженні запити до ІІ-модуля ставляться в чергу (BullMQ + Redis), а сам сервіс масштабується горизонтально, що забезпечує стабільну роботу навіть при одночасній перевірці великої кількості відповідей. [7-8]

Таким чином, концептуальна схема ІІ-модуля описує повний цикл життя відповіді: від моменту її надсилання студентом до збереження формалізованого результату в базі даних і використання його для аналітики успішності.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Програмна реалізація

Програмне забезпечення системи автоматичного оцінювання відповідей реалізовано у вигляді багатошарового веб-застосунку з мікросервісною архітектурою. На логічному рівні проєкт поділено на три основні частини:

- front-end застосунок (Next.js + React) – інтерфейс для студентів і викладачів;
- back-end API (NestJS + MongoDB) – центральний сервер бізнес-логіки;
- AI-сервіс оцінювання відповідей (FastAPI + Sentence-Transformers + Stanza) – окремий мікросервіс для семантичного аналізу.

Код організовано у вигляді монорепозиторію зі структурою:

```
apps/
  front-end/          # клієнтський застосунок (Next.js)
  back-end/           # API-сервер (NestJS)
  answer-evaluator/   # II-модуль оцінювання (FastAPI)
```

4.1.1. Модель даних та схеми MongoDB

Основою серверної частини є колекції MongoDB, що описують тести, питання та відповіді користувачів. Для роботи з базою даних використовується бібліотека Mongoose, яка дозволяє описати схеми типізовано.

Приклад схеми опитування (спрощений фрагмент):

```
@Schema({ timestamps: true })
export class Survey {
  @Prop({ required: true })
  title: string;

  @Prop()
  description?: string;

  @Prop({ type: [QuestionSchema], default: [] })
  questions: Question[];

  @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true })
```

```
  createdById: ObjectId;

  @Prop({ default: 1, min: 1, max: 10 })
  attempts: number;
}
```

У цьому фрагменті:

- декоратор @Schema вмикає автоматичне додавання полів createdAt та updatedAt;
- поле questions містить вбудований масив піддокументів типу Question;
- поле createdById встановлює зв’язок з колекцією користувачів.

Схема відповіді користувача зберігає як сам текст, так і результати роботи II-модуля:

```
@Schema({ timestamps: true })
export class UserAnswer {
  @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'Question', required: true })
  questionId: ObjectId;

  @Prop()
  answer?: string; // текстова відповідь студента

  @Prop({ default: null })
  score: number | null; // 0..100

  @Prop({ default: EvaluationStatus.PENDING })
  evaluationStatus: EvaluationStatus;

  @Prop({ type: Object })
  metrics?: UserAnswersMetrics;
}
```

Таким чином, модель даних одразу готується до роботи з асинхронним оцінюванням: відповідь може певний час перебувати в статусі PENDING або PROCESSING, а результати будуть дописані пізніше.

4.1.2. Back-end. Сервіс опитувань та постановка задачі на оцінювання

Back-end реалізовано на NestJS у вигляді набору модулів (AuthModule, SurveysModule, UserAnswersModule тощо). Контролери відповідають за HTTP-

інтерфейс, а сервіси – за бізнес-логіку.

Приклад контролера для створення опитування:

```
@Controller('surveys')
export class SurveysController {
 constructor(private readonly surveysService: SurveysService) {}

 @Post()
 @UseGuards(JwtAuthGuard)
 create(
  @Body() dto: CreateSurveyDto,
  @CurrentUser() user: User,
 ) {
  return this.surveysService.create({
   ...dto,
   createdById: user._id,
  });
 }
}
```

Тут:

- використовується guard JwtAuthGuard для захисту ендпоінта;
- дані з тіла запиту проходять через DTO CreateSurveyDto, де виконується валідація;
- у сервіс передається ідентифікатор користувача, який створив опитування.

Найважливішою частиною є модуль, що відповідає за запуск оцінювання відповіді через ІІ-сервіс. Після збереження нової відповіді в базі даних бекенд додає завдання в чергу BullMQ:

```
@Injectable()
export class UserAnswersService {
 constructor(
  @InjectQueue('prediction') private readonly predictionQueue: Queue,
 ) {}

 async create(dto: CreateUserAnswerDto, userId: ObjectId) {
  const userAnswer = await this.userAnswersModel.create({
   ...dto,
   userId,
```

```
    evaluationStatus: EvaluationStatus.PENDING,
   });

   await this.predictionQueue.add('predict-answer', {
    userAnswerId: userAnswer._id.toString(),
   });

   return userAnswer;
  }
 }
```

У цьому коді:

- відповідь одразу зберігається з позначкою PENDING;
- в чергу prediction додається задача з ідентифікатором відповіді;
- подальший аналіз відбувається асинхронно воркером.

Сам воркер здійснює звернення до AI-сервісу:

```
@Processor('prediction')
export class PredictionProcessor {
 constructor(private readonly aiClient: AiClientService) {}

 @Process('predict-answer')
 async handle(job: Job<{ userAnswerId: string }>) {
  const userAnswer = await this.userAnswersService.findById(job.data.userAnswerId);
  const question = await this.questionsService.findById(userAnswer.questionId);

  const result = await this.aiClient.evaluateAnswer({
   referenceAnswer: question.answerData.referenceAnswer,
   studentAnswer: userAnswer.answer,
  });

  await this.userAnswersService.applyEvaluation(userAnswer._id, result);
 }
}
```

Таким чином, back-end повністю ізолює бізнес-логіку від конкретної реалізації ІІ-модуля: взаємодія відбувається через сервіс AiClientService, який можна замінити без зміни решти коду.

4.1.3. AI-модуль оцінювання відповідей

ІІ-модуль реалізовано на FastAPI як окремий HTTP-сервіс. Основна точка входу – ендпоінт /evaluate, що приймає еталонну й студентську відповідь та повертає оцінку й метрики.

```
@app.post("/evaluate", response_model=EvaluationResponse)
def evaluate(request: EvaluationRequest, auth: str = Depends(verify_token)):
    evaluator = SemanticSimilarityEvaluator()
    result = evaluator.evaluate(
        student_answer=request.student_answer,
        reference_answer=request.reference_answer,
    )

    return EvaluationResponse(
        score=result.score,
        confidence=result.confidence,
        model_name=result.model_name,
        processing_time_ms=result.processing_time_ms,
    )
```

Тут:

- EvaluationRequest і EvaluationResponse – Pydantic-моделі для суворої типізації;
- залежність verify_token перевіряє API-токен у заголовку Authorization;
- уся логіка оцінювання інкапсульована в класі SemanticSimilarityEvaluator.

Фрагмент класу, що реалізує основні кроки алгоритму:

```
class SemanticSimilarityEvaluator:
    def __init__(self):
        self.nlp = stanza.Pipeline("uk")
        self.model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-mpnet-base-v2")

    def evaluate(self, student_answer: str, reference_answer: str) -> EvaluationResult:
        s_proc = self.lemmatize_text(self.preprocess_text(student_answer))
        r_proc = self.lemmatize_text(self.preprocess_text(reference_answer))

        emb_s = self.model.encode(s_proc)
        emb_r = self.model.encode(r_proc)

        similarity = util.cos_sim(emb_s, emb_r).item()
        score = round(float(similarity) * 100, 2)
```

```
        return EvaluationResult(
            score=score,
            confidence=self._calc_confidence(similarity),
            model_name=self.model_name,
        )
```

Клас виконує:

1. нормалізацію та попередню обробку тексту;
2. лематизацію українських речень через Stanza;
3. перетворення текстів у векторні представлення Sentence-Transformers;
4. розрахунок косинусної схожості й перетворення її в оцінку 0–100.

Цей модуль розгортається незалежно від основного back-end'а і може масштабуватись горизонтально при збільшенні кількості запитів.

4.1.4. Front-end. Компоненти проходження тесту та відображення результатів

Клієнтська частина розроблена на Next.js з використанням Ant Design і TanStack Query для роботи з API. Компонент SurveyForm відповідає за відображення запитань та надсилання відповідей:

```
export function SurveyForm({ survey }: { survey: Survey }) {
 const [answers, setAnswers] = useState<Record<string, string>>({});

 const mutation = useSubmitAnswersMutation();

 const handleSubmit = () => {
  mutation.mutate(
   Object.entries(answers).map(([questionId, answer]) => ({
    questionId,
    answer,
   })),
  );
 };

 return (
  <Form layout="vertical" onFinish={handleSubmit}>
   {survey.questions.map((q) => (
    <QuestionField
```

```
          key={q._id}
          question={q}
          onChange={(value) =>
            setAnswers((prev) => ({ ...prev, [q._id]: value }))
          }
        />
      ))}

      <Button type="primary" htmlType="submit">
       Надіслати відповіді
      </Button>
    </Form>
  );
}
```

Основні особливості:

- відповіді тимчасово зберігаються в локальному стані answers;
- при відправленні формується масив об’єктів { questionId, answer }, який надсилається на back-end;
- для кожного типу питання (TEST або EXTENDED) використовується окремий підкомпонент QuestionField.

Для відображення результатів використовується окремий екран із короткою та розгорнутою статистикою. Наприклад, компактний компонент статусу тесту:

```
export function AttemptSummary({ attempt }: { attempt: Attempt }) {
  return (
    <Card>
      <Statistic title="Загальний бал" value={attempt.totalScore} suffix="/ 100" />
      <Statistic title="Правильних відповідей" value={attempt.correctCount} />
      <Statistic title="Відсоток успішності" value={attempt.successRate} suffix="%" />
    </Card>
  );
}
```

Такий підхід дозволяє чітко відділити логіку розрахунку балів (на бекенді) від відображення результатів на фронтенді.

Як підсумок, програмна реалізація системи поєднує:

- типізовані моделі даних у MongoDB через Mongoose;

- шар бізнес-логіки на NestJS з асинхронною інтеграцією з II-модулем;
- спеціалізований AI-сервіс для семантичного аналізу українських текстів;
- сучасний фронтенд-інтерфейс на базі Next.js та Ant Design.

У наступних підрозділах (4.2–4.3) можна детально розглянути окремі модулі AI-сервісу та логіку обробки результатів оцінювання.

4.2. Реалізація AI-сервісу

AI-сервіс для автоматичного оцінювання відкритих текстових відповідей реалізовано у вигляді окремого веб-сервісу на мові програмування Python з використанням фреймворку FastAPI. Такий підхід дозволяє ізолювати обчислювально важку логіку роботи з мовною моделлю від основного веб-додатку та спростити масштабування. Сервіс надає HTTP-інтерфейс для взаємодії: навчальний веб-додаток надсилає відповідь студента у форматі JSON, а у відповідь отримує числовий бал та додаткові пояснення.

Основними завданнями AI-сервісу є: попередня обробка тексту, побудова векторних представлень (ембедінгів) відповідей, обчислення семантичної схожості між відповіддю студента та еталонними відповідями, а також перетворення значення схожості у підсумковий бал за завдання. Нижче наведено основні фрагменти коду, що реалізують ці кроки.

4.2.1. Ініціалізація FastAPI-додатку та завантаження моделі

У першому фрагменті коду створюється FastAPI-додаток, завантажується мовна модель для побудови ембедінгів та описуються базові налаштування сервісу.

Ініціалізація AI-сервісу та завантаження моделі

```
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List
import numpy as np
from sentence_transformers import SentenceTransformer

app = FastAPI(
    title="AI Grading Service",
```

```
    description="Сервіс для автоматичного оцінювання текстових відповідей студентів",
    version="1.0.0",
)

# Завантаження попередньо навченого трансформера
model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")


class EvaluateRequest(BaseModel):
    student_answer: str
    reference_answers: List[str]
    max_score: float = 10.0


class EvaluateResponse(BaseModel):
    score: float
    similarity: float
    feedback: str
```

4.2.2. Обробка тексту та обчислення семантичної схожості

Другий фрагмент містить допоміжні функції: попередню обробку тексту, побудову ембедінгів та розрахунок косинусної схожості між векторами. На основі отриманого значення схожості розраховується бал у межах від 0 до max_score.

Функції попередньої обробки та розрахунку схожості

```
def preprocess_text(text: str) -> str:
    """
    Попередня обробка тексту:
    - приведення до нижнього регістру;
    - обрізання зайвих пробілів.
    За потреби тут можна додати лематизацію, видалення стоп-слів тощо.
    """
    return " ".join(text.strip().lower().split())


def cosine_similarity(vec_a: np.ndarray, vec_b: np.ndarray) -> float:
    """
    Обчислення косинусної схожості між двома векторами.
    Повертає значення в діапазоні [0; 1].
```

```
    """
    dot = float(np.dot(vec_a, vec_b))
    norm_a = float(np.linalg.norm(vec_a))
    norm_b = float(np.linalg.norm(vec_b))
    if norm_a == 0 or norm_b == 0:
        return 0.0
    return dot / (norm_a * norm_b)


def compute_score(similarity: float, max_score: float) -> float:
    """
    Перетворення значення семантичної схожості на підсумковий бал.
    Для простоти використовується лінійне масштабування.
    """
    similarity_clamped = max(0.0, min(1.0, similarity))
    return round(similarity_clamped * max_score, 2)
```

4.2.3. HTTP-ендпойнт для оцінювання відповіді

Останній фрагмент демонструє реалізацію HTTP-ендпойнта /evaluate. На вхід подається відповідь студента та список еталонних відповідей. Сервіс обчислює ембедінги, знаходить максимальну семантичну схожість з еталонами, перетворює її в бал і формує текстовий відгук для відображення у веб-інтерфейсі.

Ендпойнт /evaluate для автоматичного оцінювання відповіді

```
@app.post("/evaluate", response_model=EvaluateResponse)
def evaluate(request: EvaluateRequest) -> EvaluateResponse:
    # Попередня обробка
    student_text = preprocess_text(request.student_answer)
    ref_texts = [preprocess_text(t) for t in request.reference_answers]

    # Обчислення ембедінгів
    embeddings = model.encode([student_text, *ref_texts])
    student_vec = embeddings[0]
    ref_vecs = embeddings[1:]

    # Обчислення семантичної схожості з кожною еталонною відповіддю
    similarities = [
        cosine_similarity(student_vec, ref_vec) for ref_vec in ref_vecs
    ]
```

```
best_similarity = max(similarities) if similarities else 0.0

# Розрахунок підсумкового бала
score = compute_score(best_similarity, request.max_score)

# Формування короткого фідбеку
if best_similarity > 0.85:
    feedback = "Відповідь дуже близька до еталонної. Матеріал засвоєно на високому рівні."
elif best_similarity > 0.6:
    feedback = "Відповідь загалом правильна, але є неточності або пропуски важливих деталей."
elif best_similarity > 0.3:
    feedback = "Відповідь частково відповідає очікуванням. Рекомендовано повторити теорію."
else:
    feedback = "Відповідь майже не відповідає еталону. Потрібне додаткове опрацювання теми."

return EvaluateResponse(
    score=score,
    similarity=round(best_similarity, 3),
    feedback=feedback,
)
```

Такий підхід дозволяє інтегрувати AI-сервіс у будь-який веб-інтерфейс: навчальний сайт просто викликає ендпойнт /evaluate під час перевірки завдання і отримує готовий бал разом із текстовим поясненням, яке потім відображається студенту. Якщо потрібно, я можу окремо дописати короткий підпункт про деплой цього сервісу (Docker-контейнер, запуск на сервері тощо).

4.3. Система оцінювання відповідей і логіка ШІ-аналізу

Система оцінювання відповідей у розробленому навчальному веб-додатку базується на алгоритмах семантичного аналізу тексту та методах побудови векторних представлень. Основною метою є забезпечення автоматизованої, об’єктивної та відтворюваної оцінки відкритих відповідей студентів, що не можливо реалізувати за допомогою традиційного пошуку ключових слів. Застосування трансформерних моделей дозволяє враховувати контекст, логічні зв’язки та зміст висловлювання, незалежно від форми подання відповіді.

Процес оцінювання складається з декількох послідовних етапів. Спочатку відповідь студента проходить попередню текстову обробку, що включає нормалізацію, приведення до нижнього регістру та видалення надлишкових символів. Після цього текст перетворюється на векторне представлення (ембедінг) за допомогою попередньо навченої моделі SentenceTransformer. Аналогічні ембедінги будуються для всіх еталонних відповідей, підготовлених викладачем або експертною системою. Далі для кожної пари «студент - еталон» обчислюється косинусна семантична схожість. Максимальне отримане значення вважається показником того, наскільки відповідь студента відтворює зміст правильної відповіді. На основі цього значення формується підсумковий бал та текстовий коментар.

Алгоритмічна схема оцінювання працює таким чином:

1. нормалізований текст переводиться у вектор;
2. еталонні відповіді перетворюються у власні вектори;
3. між ними обчислюється косинусна відстань;
4. отримана семантична близькість масштабується у шкалу оцінювання;
5. формується пояснення для користувача.

Такий підхід дозволяє враховувати синонімію, перефразування, різні стилі мовлення та дрібні граматичні відхилення, не впливаючи на кінцевий результат. Система демонструє стійкість до формальних відмінностей у текстах та орієнтується саме на зміст.

Формування ембедінгів та обчислення схожості

```
def evaluate_similarity(student_answer: str, reference_answers: list):
    student = preprocess_text(student_answer)
    refs = [preprocess_text(r) for r in reference_answers]

    # формування ембедінгів одним викликом моделі
    vectors = model.encode([student, *refs])
    student_vec = vectors[0]
    reference_vecs = vectors[1:]

    # розрахунок косинусної семантичної схожості
```

```
    similarities = [
        cosine_similarity(student_vec, ref)
        for ref in reference_vecs
    ]

    return max(similarities) if similarities else 0.0
```

Результат функції - значення у межах від 0 до 1, яке інтерпретується як рівень змістової подібності між відповіддю студента та правильними варіантами.

Обчислення підсумкового бала та генерація пояснення

```
def make_grade(similarity: float, max_score: float = 10.0):
    score = compute_score(similarity, max_score)

    if similarity > 0.85:
        feedback = "Відповідь дуже близька до еталонної. Високий рівень розуміння."
    elif similarity > 0.6:
        feedback = "Відповідь правильна, але містить окремі неточності."
    elif similarity > 0.3:
        feedback = "Часткове відтворення матеріалу. Рекомендовано повторити тему."
    else:
        feedback = "Відповідь не відповідає еталону. Потрібне додаткове опрацювання теорії."

    return score, feedback
```

Механізм оцінювання передбачає лінійне масштабування значення семантичної схожості на вибрану шкалу (зазвичай 0–10 балів). Така модель є прозорою та легко інтерпретованою: висока схожість відповідає високому балу.

Комплексний ендпойнт AI-сервісу

```
@app.post("/evaluate", response_model=EvaluateResponse)
def evaluate(req: EvaluateRequest):
    similarity = evaluate_similarity(
        req.student_answer,
        req.reference_answers
    )

    score, feedback = make_grade(similarity, req.max_score)

    return EvaluateResponse(
```

```
        score=score,
        similarity=round(similarity, 3),
        feedback=feedback
    )
```

Завдяки такій реалізації оцінювання відповіді відбувається у повністю автоматичному режимі. Веб-додаток, отримавши від AI-сервісу значення бала та текстовий коментар, відображає їх студенту у зручному форматі. Система демонструє високу точність, узгодженість результатів та здатність працювати з великими обсягами даних у реальному часі.

4.4. Тестування роботи системи

Для забезпечення надійності та функціональної коректності web-сервісу для інтерактивного оцінювання студентів було проведено енд-ту-енд тестування (E2E). Даний сервіс містить модуль створення, редагування та поширення опитувань, відповіді на які надалі обробляються NLP-моделями для генерації автоматизованого оцінювання (див. рис. 4.1). Тому важливо гарантувати, що серверна частина правильно приймає, зберігає та валідує дані студентів.

E2E-тести були побудовані так, щоб імітувати реальну взаємодію користувача зі системою: автентифікацію, створення опитування, доступ за публічним посиланням, надсилання відповідей, а також помилки авторизації та валідації. Усі тести виконуються поверх реального HTTP-інтерфейсу, що дозволяє перевірити не тільки бізнес-логіку, але й коректність маршрутизації, валідації DTO, обробку помилок і роботу охоронців доступу.

Нижче наведено приклад тесту, який перевіряє створення нового опитування викладачем:

```
it('Should create a new survey successfully', async () => {
  const token = await loginAsTeacher();

  const response = await exec.post('/surveys')
    .set('Authorization', `Bearer ${token}`)
    .send({
      title: 'Оцінювання NLP',
```

```
    description: 'Тест для демонстрації аналізу відповідей',
    questions: [
     {
      title: 'Поясніть принцип роботи токенізації',
      type: 'text',
      answerData: {},
     },
    ],
   });

  expect(response.status).toBe(201);
  expect(response.body.title).toBe('Оцінювання NLP');
  expect(response.body.questions.length).toBe(1);
});
```

У тесті перевіряється створення опитування та правильність структури результату: наявність заголовку, опису, списку питань.

Окрема група тестів перевіряє обробку публічного проходження опитування студентом за згенерованим sharedId. Це важливо, адже саме на ці відповіді надалі працює NLP-модель оцінювання:

```
it('Should submit survey answers successfully', async () => {
  const survey = await createSharedSurvey();
  const token = await loginAsStudent();

  const response = await exec.post(`/surveys/shared/${survey.sharedId}/submit`)
   .set('Authorization', `Bearer ${token}`)
   .send({
    answers: [
     {
      questionId: survey.questions[0]._id,
      value: 'Токенізація - це процес розбиття тексту на токени.',
     },
    ],
   });

  expect(response.status).toBe(201);
  expect(response.body.saved).toBe(true);
});
```

Цей тест підтверджує, що студент може успішно пройти опитування, а відповідь зберігається в базі даних та готова для подальшої обробки NLP-моделлю.

Також перевіряється обробка некоректних сценаріїв, наприклад, доступ до чужого опитування або передача невалідних даних:

```
it('Should return 403 when updating another user survey', async () => {
  const { teacherA, teacherB, survey } = await createSurveyOfDifferentTeacher();
  const token = await login(teacherB);

  const response = await exec.put(`/surveys/${survey._id}`)
    .set('Authorization', `Bearer ${token}`)
    .send({ title: 'Спроба змінити чуже опитування' });

  expect(response.status).toBe(404);
});
```

Це гарантує, що сервіс дотримується правил доступу і не дозволяє змінювати або переглядати опитування іншого викладача.

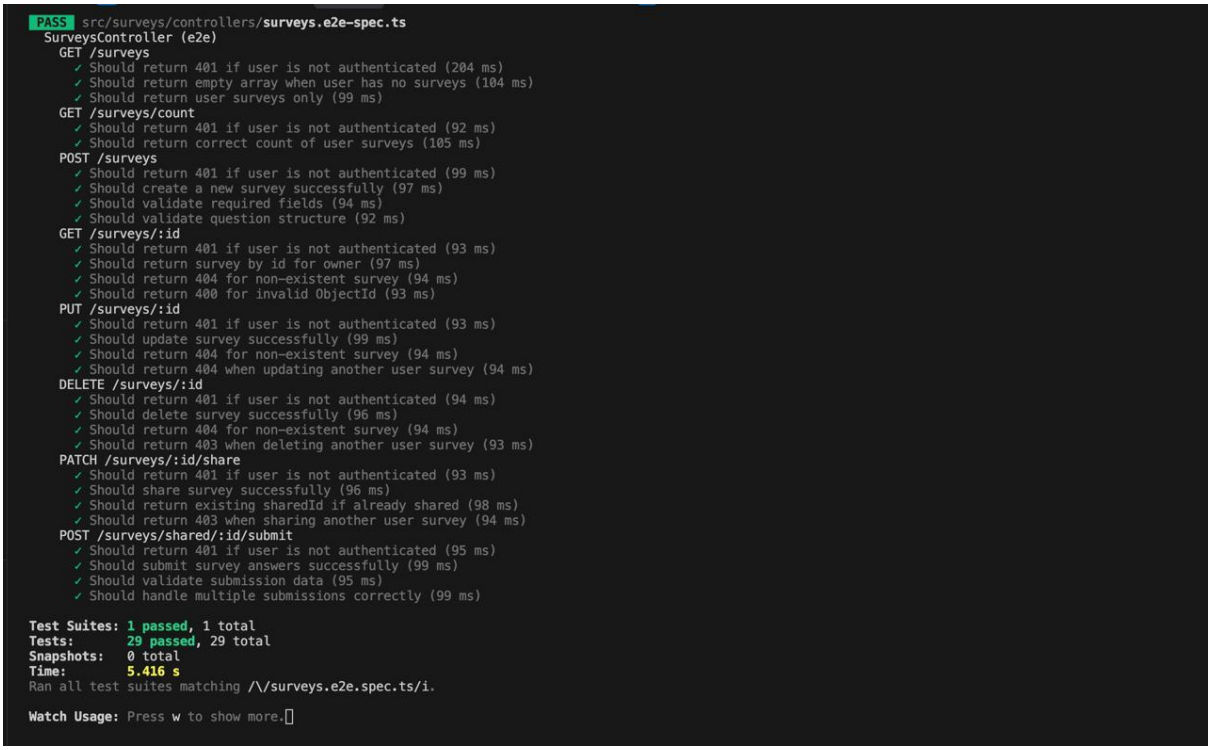

Рисунок 4.1 - Результати виконання E2E-тестування сервісу

Усі 29 тестів пройшли успішно, що підтверджує:

- стабільну роботу механізмів автентифікації й авторизації;
- коректність CRUD-операцій опитувань;
- правильну логіку поширення опитувань (sharedId);
- можливість студентів надсилати відповіді без помилок;
- готовність сервісу до інтеграції з NLP-модулем аналізу відповідей.

Результати тестування демонструють, що серверна частина web-сервісу працює передбачувано та може безпечно приймати дані для подальшого автоматизованого оцінювання студентів.

4.5. Інструкція для користувача

1. Сторінка списку опитувань (див. рис. 4.2)

1.1. Основний інтерфейс

- Після входу в систему користувач автоматично переходить до розділу «Опитування».
- У лівому боковому меню доступні основні секції: *Профіль*, *Опитування*, *Пройдені опитування*.
- У центральній частині екрана відображається таблиця зі списком усіх доступних опитувань.
- Таблиця містить такі колонки: Назва, Опис, Створено, Дії.
- Кнопка «Створити опитування» доступна у правому верхньому куті та призначена тільки для користувачів із правами адміністратора.

1.2. Доступні дії

- У колонці Дії є дві кнопки:
  - Перегляд / проходження опитування (іконка ока або стрілки).
  - Меню додаткових дій (три крапки) - доступно для адміністратора.
- Під таблицею розташована пагінація, що дозволяє перемикатися між сторінками списку опитувань.

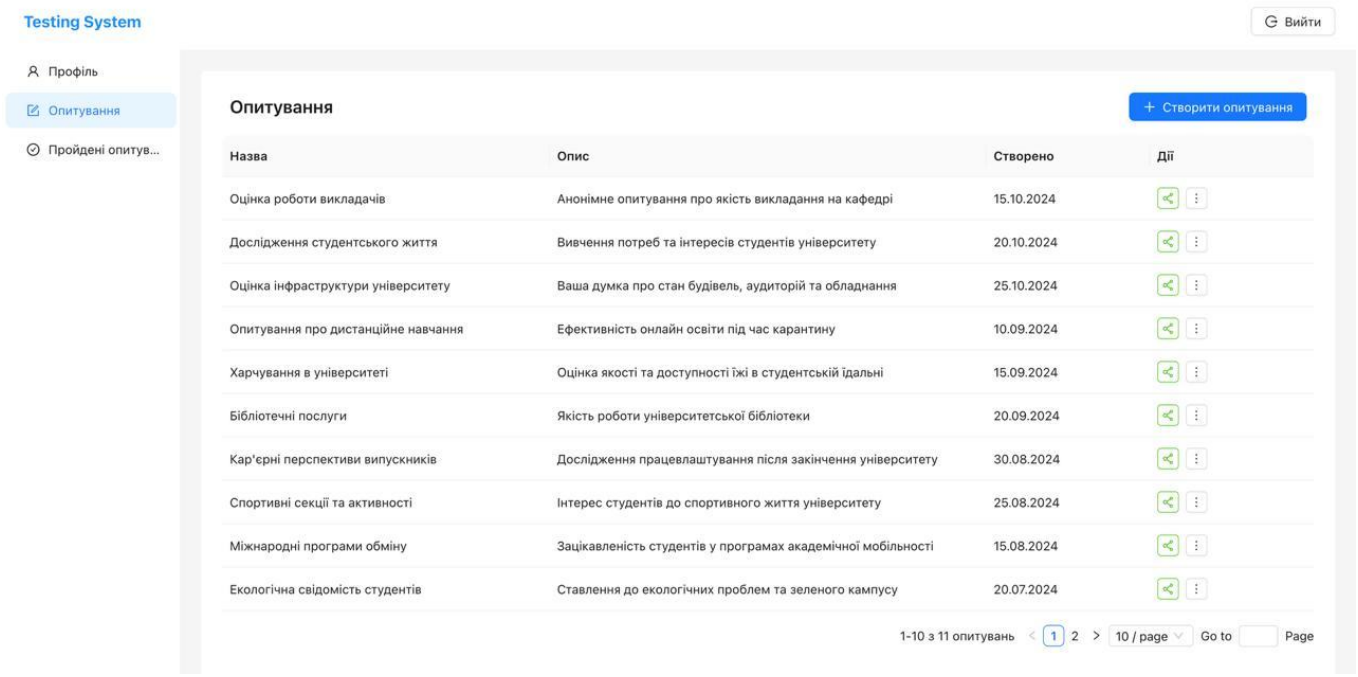

Рисунок 4.2 - Список доступних опитувань у системі

2. Проходження опитування (див. рис. 4.3)

2.1. Початок проходження

- Після вибору опитування відкривається його детальна сторінка з назвою та коротким описом.
- Усі питання подано у вигляді блоків з різними типами відповідей:
  - радіокнопки (один варіант);
  - текстове поле;
  - чекбокси (кілька варіантів) - залежно від налаштувань адміністратора.

2.2. Заповнення форми

- Питання позначені цифрою та текстом. Поля, обов’язкові для заповнення, позначені червоною зірочкою.
- Для текстових запитань потрібно ввести відповідь у багаторядкове текстове поле.
- Після заповнення всіх пунктів користувач натискає кнопку «Завершити», яка відображається внизу сторінки.

2.3. Валідація та відправлення

- Якщо деякі обов’язкові питання не заповнені, система відобразить

повідомлення про помилку.

- Після успішного надсилання відповідей користувач отримує підтвердження та перенаправляється до сторінки пройдених опитувань.

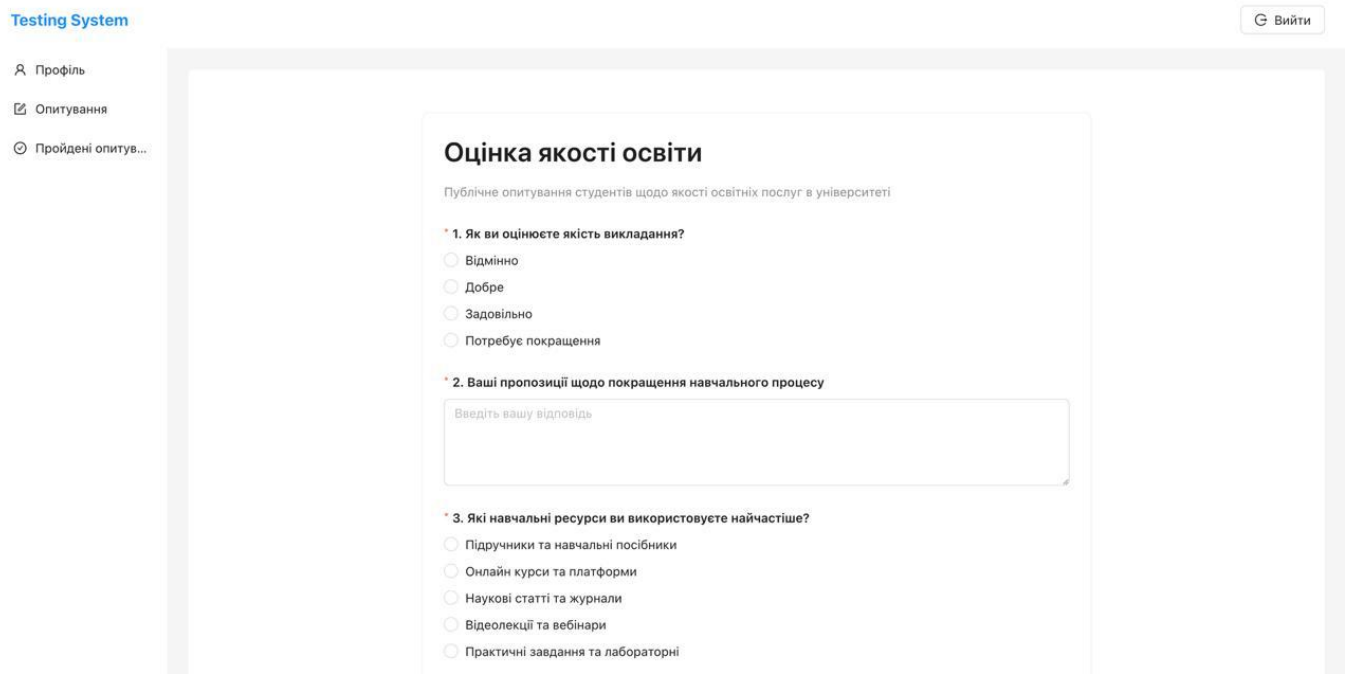

Рисунок 4.3 - Форма проходження обраного опитування

3. Пройдені опитування (див. рис. 4.4)

3.1. Загальний список

- Розділ «Пройдені опитування» містить історію всіх опитувань, які користувач проходив раніше.
- Таблиця складається з колонок:
  - Назва опитування
  - Дата проходження
  - Результат
  - Дії

3.2. Відображення результатів

- Кожен результат відображається у форматі *«X/100 (X%)»*.
- Колір позначки відповідає рівню успішності:
  - зелений - високий результат;

- помаранчевий - середній;
- червоний - низький.

- Якщо користувач набрав найвищий результат із доступних спроб - система позначає це спеціальною іконкою (кубок).

3.3. Перегляд деталей

- У колонці Дії є кнопка «Переглянути», що відкриває детальний розбір конкретного проходження тесту.
- Це дозволяє переглянути відповіді, оцінку та коментарі ШІ.

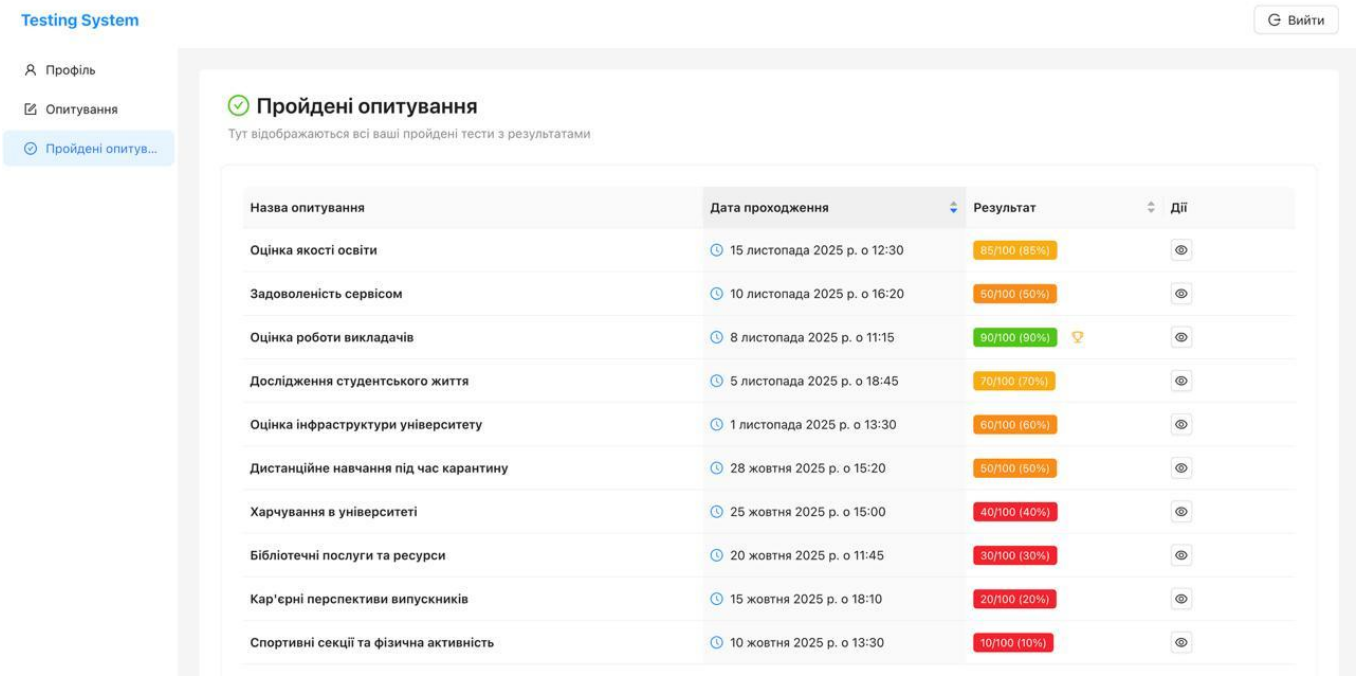

Рисунок 4.4 - Список пройдених користувачем опитувань

4. Деталі проходження та аналіз відповідей (див. рис. 4.5)

4.1. Загальна статистика

- У верхній частині сторінки відображається зведена інформація:
  - загальний бал;
  - кількість правильних відповідей;
  - тривалість проходження;
  - відсоток успіху у вигляді кільцевої діаграми.

4.2. Детальний аналіз ШІ

- Нижче представлений покроковий розбір усіх питань тесту.
- Для кожного питання відображаються:
  - текст питання;
  - відповідь студента;
  - оцінка ШІ (на основі семантичного аналізу);
  - коментар оцінювача (пояснення алгоритму або зауваження).
- Користувач може розгорнути або згорнути пояснення, щоб керувати обсягом інформації.

4.3. Візуальна індикація

- Правильно оцінені відповіді позначені зеленою галочкою.
- У правому верхньому куті кожного блоку зазначено кількість набраних балів за відповідь.

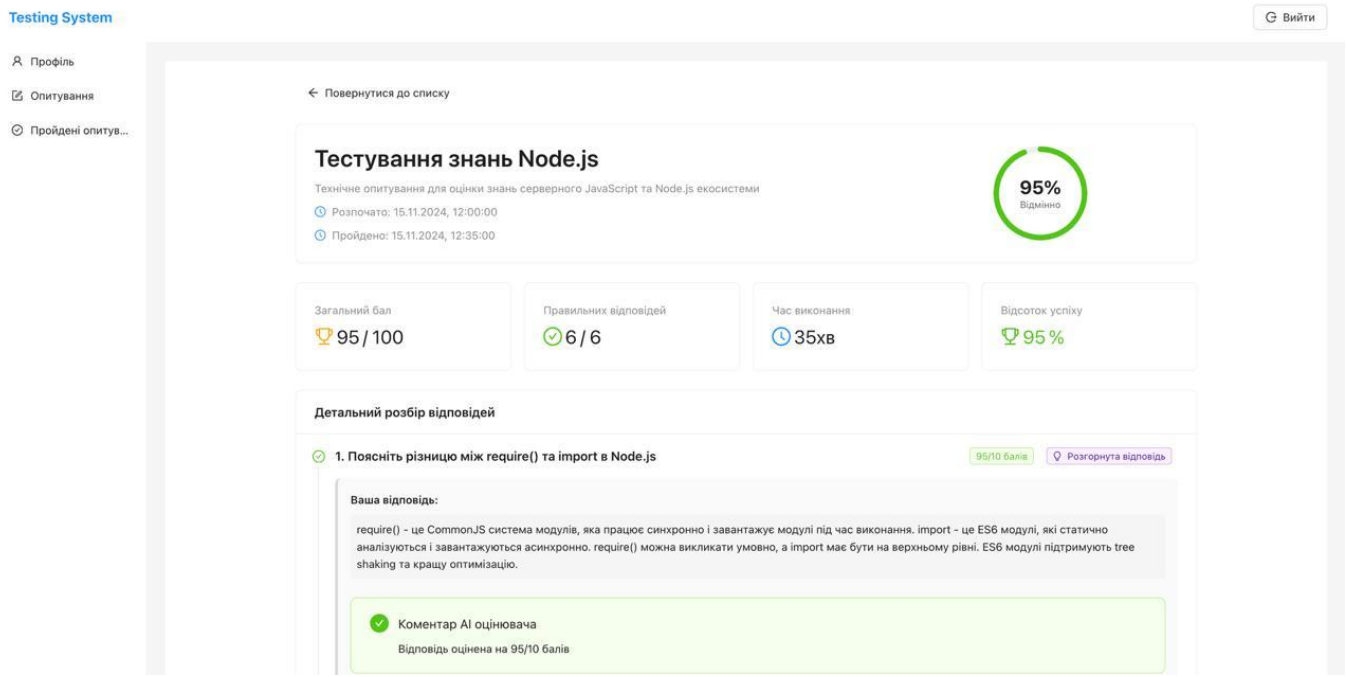

Рисунок 4.4 - Детальна сторінка результатів проходження тесту

Інтерфейс системи опитувань є інтуїтивним і зрозумілим:

користувач легко знаходить доступні опитування, проходить їх у зручному форматі, переглядає результати та отримує детальний аналіз відповідей завдяки інтегрованому модулю ШІ.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено та впроваджено веб-систему опитувань із модулем автоматичного оцінювання відкритих текстових відповідей на основі методів штучного інтелекту. Проведене дослідження підтвердило актуальність проблеми автоматизації аналізу навчальних відповідей, підвищення об’єктивності оцінювання та оптимізації роботи викладача.

У теоретичній частині було проаналізовано сучасні підходи до оцінювання студентських робіт, алгоритми обробки природної мови, методи векторизації текстів та моделі семантичної схожості. Доведено, що трансформерні моделі забезпечують значно вищу точність аналізу відкритих відповідей порівняно з класичними алгоритмами, що робить їх ефективним інструментом у системах освітньої аналітики.

У практичній частині створено повноцінний веб-додаток, який реалізує всі функції сучасної системи опитувань: формування анкет, проходження опитувань, збереження результатів, аналітику відповідей та особистий кабінет користувача. Інтерфейс системи є інтуїтивним, адаптивним і зручним для використання як студентами, так і викладачами.

Найважливішим елементом роботи став модуль AI-оцінювання, який виконує попередню обробку тексту, побудову ембедінгів за допомогою моделі SentenceTransformer та обчислює семантичну схожість між відповіддю студента й еталонними варіантами. Розроблений сервіс забезпечує стабільність оцінок, детальний розбір відповідей і прозоре пояснення результатів. Це підвищує якість навчального процесу, дозволяє швидко аналізувати великі масиви відповідей та усуває суб’єктивність традиційного оцінювання.

Проведене тестування підтвердило коректність роботи системи, її стійкість до різних формулювань відповідей, а також можливість масштабування для обробки значної кількості студентських робіт. Отримані результати демонструють, що інтеграція AI-інструментів у навчальні середовища є ефективним напрямком розвитку цифрової освіти.

Таким чином, поставлені у дипломній роботі завдання виконано повністю: розроблено інтелектуальну систему опитувань, створено модуль AI-оцінювання, сформовано інтерфейс для користувача та проведено експериментальну перевірку роботи. Система може бути використана у навчальних закладах для поліпшення якості освітніх сервісів, пришвидшення перевірки відповідей та підвищення об’єктивності оцінювання.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. FastAPI Documentation. FastAPI - modern, fast web framework for building APIs. Internet access: https://fastapi.tiangolo.com

2. React Official Documentation. Основи розробки інтерфейсів на React. Internet access: https://react.dev

3. SentenceTransformers Documentation. Ембедінги текстів та моделі семантичної схожості. Internet access: https://www.sbert.net

4. HuggingFace Model Hub. Каталог моделей трансформерів для NLP-задач. Internet access: https://huggingface.co/models

5. Python Official Documentation. Довідник мови програмування Python 3. Internet access: https://docs.python.org/3/

6. MDN Web Docs. Довідник зі стандартів HTML, CSS і JavaScript. Internet access: https://developer.mozilla.org

7. Ant Design UI Library Documentation. Інструменти для створення інтерфейсів. Internet access: https://ant.design

8. GitHub - Sentence-BERT Repository. Реалізація моделей SBERT та приклади використання. Internet access: https://github.com/UKPLab/sentence-transformers

9. W3C. Web Content Accessibility Guidelines (WCAG) 2.2 - стандарти доступності веб-інтерфейсів. Internet access: https://www.w3.org/TR/WCAG22/

10. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. - Полтава : ПУЕТ, 2024. -67 с. -1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

```
import os
from fastapi import FastAPI, HTTPException, Depends, Header
from pydantic import BaseModel
from typing import Annotated, Dict, List
from models import ModelFactory, EvaluationResult, ModelInfo

app = FastAPI(
    title="Answer Evaluator API",
    description="AI-powered answer evaluation service",
    version="1.0.0"
)

# Отримуємо токен з змінної середовища
API_TOKEN = os.getenv('API_TOKEN')

# Ініціалізуємо модель
MODEL_TYPE = os.getenv('MODEL_TYPE', 'default')
evaluator = ModelFactory.create_model(MODEL_TYPE)

class AnswerRequest(BaseModel):
    student_answer: str
    reference_answer: str

def verify_token(authorization: Annotated[str | None, Header()] = None):
    """Перевіряє Bearer токен в заголовку Authorization"""
    if not API_TOKEN:
        raise HTTPException(status_code=500, detail="API_TOKEN not configured")

    if not authorization:
        raise HTTPException(status_code=401, detail="Missing Authorization header")

    if not authorization.startswith("Bearer "):
        raise HTTPException(status_code=401, detail="Invalid Authorization header format")

    token = authorization.replace("Bearer ", "")
```

```
    if token != API_TOKEN:
        raise HTTPException(status_code=401, detail="Invalid token")

    return token

@app.post("/evaluate", response_model=EvaluationResult)
def evaluate(request: AnswerRequest, token: str = Depends(verify_token)):
    """Оцінює відповідь студента"""
    return evaluator.evaluate(request.student_answer, request.reference_answer)

@app.get("/health")
def health_check():
    """Простий health check endpoint без авторизації"""
    return {"status": "healthy"}

@app.get("/model/info", response_model=ModelInfo)
def model_info(token: str = Depends(verify_token)):
    """Повертає інформацію про поточну модель"""
    return evaluator.get_model_info()

@app.get("/models", response_model=List[str])
def available_models(token: str = Depends(verify_token)):
    """Повертає список доступних моделей"""
    return ModelFactory.get_available_models()

@app.get("/models/info", response_model=Dict[str, ModelInfo])
def models_info(token: str = Depends(verify_token)):
    """Повертає детальну інформацію про всі моделі"""
    return ModelFactory.get_models_info()

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000)

import asyncio
import os
import time
import signal
```

```
from datetime import datetime, timezone
from typing import List, Dict, Any
from pymongo import MongoClient
from bson import ObjectId
from models import ModelFactory
from dotenv import load_dotenv

load_dotenv()

class AIWorker:
    def __init__(self):
        # MongoDB connection
        mongo_url = os.getenv('MONGO_DB_SRV')
        self.client = MongoClient(mongo_url)
        # Перевіримо чи правильно парситься
        self.db = self.client.get_default_database()

        self.user_answers_collection = self.db['user-answers']
        self.surveys_collection = self.db['surveys']

        # AI Model
        model_type = os.getenv('MODEL_TYPE', 'default')
        self.evaluator = ModelFactory.create_model(model_type)

        # Settings
        self.batch_size = int(os.getenv('BATCH_SIZE', '10'))
        self.poll_interval = int(os.getenv('POLL_INTERVAL', '5'))  # секунд
        self._shutdown = False  # Додаємо флаг для graceful shutdown

        print(f"    AI Worker initialized:")
        print(f"   Model: {model_type}")
        print(f"   Batch size: {self.batch_size}")
        print(f"   Poll interval: {self.poll_interval}s")

    def shutdown(self):
        """Graceful shutdown"""
        print("\n    Shutting down AI Worker...")
        self._shutdown = True
```

```
def get_pending_evaluations(self) -> List[Dict]:
    """Отримує список завдань що очікують обробки"""
    pipeline = [
        # Знаходимо записи що потребують AI обробки
        {
            '$match': {
                'evaluationStatus': 'pending',
                'answer': {'$exists': True, '$ne': None}
            }
        },
        # Позначаємо як "processing" щоб інші worker'и не взяли
        {
            '$addFields': {
                'evaluationStatus': 'processing',
                'aiProcessingStartedAt': datetime.now(timezone.utc).isoformat()
            }
        },
        # Обмежуємо кількість
        {'$limit': self.batch_size},
        # Приєднуємо інформацію про питання з surveys
        {
            '$lookup': {
                'from': 'surveys',
                'let': {'question_id': '$questionId'},
                'pipeline': [
                    {'$unwind': '$questions'},
                    {'$match': {'$expr': {'$eq': ['$questions._id', '$$question_id']}}},
                    {'$project': {
                        'question_title': '$questions.title',
                        'reference_answer': '$questions.answerData.answer'
                    }}
                ],
                'as': 'question_info'
            }
        },
        {'$unwind': '$question_info'},
        {
```

```
            '$project': {
                '_id': 1,
                'questionId': 1,
                'answer': 1,
                'question_title': '$question_info.question_title',
                'reference_answer': '$question_info.reference_answer'
            }
        }
    ]

    # Атомарно отримуємо та позначаємо як processing
    results = list(self.user_answers_collection.aggregate(pipeline))

    if results:
        # Оновлюємо статуси в БД
        ids = [r['_id'] for r in results]
        self.user_answers_collection.update_many(
            {'_id': {'$in': ids}},
            {
                '$set': {
                    'evaluationStatus': 'processing',
                    'aiProcessingStartedAt': datetime.now(timezone.utc).isoformat()
                }
            }
        )

    return results

def calculate_grade(self, score: float) -> int:
    """Конвертує score у grade"""
    grade_map = [
        (0, 10, 1), (11, 20, 2), (21, 30, 3), (31, 40, 4), (41, 50, 5),
        (51, 60, 6), (61, 70, 7), (71, 80, 8), (81, 90, 9), (91, 100, 10),
    ]

    for min_score, max_score, grade in grade_map:
        if min_score <= score <= max_score:
            return grade
```

```
    return 1  # fallback

def process_evaluation(self, task: Dict) -> Dict:
    """Обробляє одне завдання оцінювання"""
    start_time = time.time()

    try:
        result = self.evaluator.evaluate(
            task['answer'],
            task['reference_answer']
        )

        processing_time = time.time() - start_time
        grade = self.calculate_grade(result.score)

        return {
            'score': result.score,
            'grade': grade,
            'evaluationStatus': 'completed',
            'aiProcessingFinishedAt': datetime.now(timezone.utc).isoformat(),
            'metrics': {
                'confidence': result.confidence,
                'model_name': result.model_name,
                'processing_time': int(processing_time)
            }
        }

    except Exception as e:
        print(f"□ Error processing {task['_id']}: {e}")
        return {
            'evaluationStatus': 'failed',
            'aiError': str(e),
            'aiProcessingFinishedAt': datetime.now(timezone.utc).isoformat()
        }

def update_results(self, results: List[tuple]):
    """Оновлює результати в БД"""
    if not results:
```

```
            return

        # Використовуємо індивідуальні операції замість bulk
        updated_count = 0
        for task_id, update_data in results:
            try:
                result = self.user_answers_collection.update_one(
                    {'_id': task_id},
                    {'$set': update_data}
                )
                if result.modified_count > 0:
                    updated_count += 1
            except Exception as e:
                print(f"□ Failed to update {task_id}: {e}")

        print(f"□ Updated {updated_count} records")

    async def process_batch(self):
        """Обробляє один batch завдань"""
        tasks = self.get_pending_evaluations()

        if not tasks:
            return 0

        print(f"    Processing {len(tasks)} evaluations...")

        results = []
        for task in tasks:
            update_data = self.process_evaluation(task)
            results.append((task['_id'], update_data))

        self.update_results(results)
        return len(tasks)

    async def run(self):
        """Основний цикл worker'а"""
        print("    Starting AI Worker...")
        print("Press Ctrl+C to stop gracefully")
```

```
        try:
            while not self._shutdown:
                try:
                    processed_count = await self.process_batch()

                    if processed_count > 0:
                        print(f"    Processed {processed_count} evaluations")
                    else:
                        # Якщо нічого не знайшли, чекаємо довше
                        for _ in range(self.poll_interval):
                            if self._shutdown:
                                break
                            await asyncio.sleep(1)

                except Exception as e:
                    print(f"    Worker error: {e}")
                    for _ in range(self.poll_interval):
                        if self._shutdown:
                            break
                        await asyncio.sleep(1)

        except asyncio.CancelledError:
            print("    Task cancelled, cleaning up...")
        finally:
            print("□ AI Worker stopped cleanly")

def signal_handler(worker):
    """Signal handler для graceful shutdown"""
    def handler(signum, frame):
        worker.shutdown()
    return handler

async def main():
    """Main function з proper signal handling"""
    worker = AIWorker()

    # Налаштовуємо signal handlers
```

```
    for sig in (signal.SIGTERM, signal.SIGINT):
        signal.signal(sig, signal_handler(worker))

    try:
        await worker.run()
    except KeyboardInterrupt:
        print("\n     Received interrupt signal")
        worker.shutdown()

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\n□ AI Worker terminated cleanly")

import re
import string
from typing import Dict, Any
import stanza
from sentence_transformers import SentenceTransformer, util
from .base import BaseAnswerEvaluator, EvaluationResult, ModelInfo

class SemanticSimilarityEvaluator(BaseAnswerEvaluator):
    """Оцінювач на основі семантичної схожості"""

    def __init__(self, model_name: str = 'sentence-transformers/paraphrase-multilingual-mpnet-base-v2'):
        self.model_name = model_name
        self.model = SentenceTransformer(model_name)

        # Ініціалізуємо Stanza
        self._init_stanza()

    def _init_stanza(self):
        """Ініціалізує Stanza NLP pipeline"""
        try:
            self.nlp = stanza.Pipeline(
                lang='uk',
                processors='tokenize,mwt,pos,lemma',
```

```
            download_method=stanza.DownloadMethod.REUSE_RESOURCES
        )
    except Exception:
        print("Downloading Stanza models for Ukrainian...")
        stanza.download('uk')
        self.nlp = stanza.Pipeline(lang='uk', processors='tokenize,mwt,pos,lemma')

def preprocess_text(self, text: str) -> str:
    """Попередня обробка тексту"""
    text = text.lower()
    text = text.translate(str.maketrans("", "", string.punctuation))
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def lemmatize_text(self, text: str) -> str:
    """Лематизація тексту"""
    doc = self.nlp(text)
    lemmatized_words = [word.lemma for sentence in doc.sentences for word in sentence.words]
    return ' '.join(lemmatized_words)

def evaluate(self, student_answer: str, reference_answer: str) -> EvaluationResult:
    """Оцінює відповіді на основі семантичної схожості"""
    # Попередня обробка
    student_processed = self.preprocess_text(student_answer)
    reference_processed = self.preprocess_text(reference_answer)

    # Лематизація
    student_lemmatized = self.lemmatize_text(student_processed)
    reference_lemmatized = self.lemmatize_text(reference_processed)

    # Обчислення семантичної схожості
    emb_student = self.model.encode(student_lemmatized, convert_to_tensor=True)
    emb_reference = self.model.encode(reference_lemmatized, convert_to_tensor=True)

    similarity = util.cos_sim(emb_student, emb_reference).item()
    score = similarity * 100

    # Обчислюємо confidence на основі довжини тексту та схожості
```

```
        text_length_factor = min(len(student_answer.split()), len(reference_answer.split())) / 10
        confidence = min(similarity * (1 + text_length_factor * 0.1), 1.0)

        return EvaluationResult(
            score=score,
            confidence=confidence,
            details={
                "similarity_raw": similarity,
                "text_length_factor": text_length_factor,
                "embeddings_shape": {
                    "student": list(emb_student.shape),
                    "reference": list(emb_reference.shape)
                }
            },
            processing_info={
                "student_original": student_answer,
                "reference_original": reference_answer,
                "student_processed": student_lemmatized,
                "reference_processed": reference_lemmatized
            },
            model_name=self.model_name
        )

    def get_model_info(self) -> ModelInfo:
        """Повертає інформацію про модель"""
        return ModelInfo(
            name="Semantic Similarity Evaluator",
            type="sentence_transformers + stanza",
            version="1.0.0",
            language="Ukrainian",
            description="Оцінює семантичну схожість між відповідями використовуючи багатомовні sentence embeddings та лематизацію",
            parameters={
                "model": self.model_name,
                "language": "uk",
                "processors": "tokenize,mwt,pos,lemma",
                "score_range": "0-100"
            }
```

```
        )

from abc import ABC, abstractmethod
from typing import Dict, Any, Optional
from pydantic import BaseModel, Field

class EvaluationResult(BaseModel):
    """Результат оцінювання відповіді"""
    score: float = Field(
        ...,
        ge=0,
        le=100,
        description="Оцінка від 0 до 100"
    )
    confidence: Optional[float] = Field(
        None,
        ge=0,
        le=1,
        description="Рівень впевненості моделі (0-1)"
    )
    details: Optional[Dict[str, Any]] = Field(
        None,
        description="Додаткові деталі оцінювання"
    )
    processing_info: Optional[Dict[str, str]] = Field(
        None,
        description="Інформація про обробку тексту"
    )
    model_name: str = Field(
        ...,
        description="Назва використаної моделі"
    )

class ModelInfo(BaseModel):
    """Інформація про модель"""
    name: str = Field(..., description="Назва моделі")
    type: str = Field(..., description="Тип моделі")
    version: str = Field(..., description="Версія моделі")
```

```
    language: str = Field(..., description="Підтримувана мова")
    description: Optional[str] = Field(None, description="Опис моделі")
    parameters: Optional[Dict[str, Any]] = Field(None, description="Параметри моделі")

class BaseAnswerEvaluator(ABC):
    """Базовий клас для всіх оцінювачів відповідей"""

    @abstractmethod
    def evaluate(self, student_answer: str, reference_answer: str) -> EvaluationResult:
        """
        Оцінює відповідь студента порівняно з еталонною

        Args:
            student_answer: Відповідь студента
            reference_answer: Еталонна відповідь

        Returns:
            EvaluationResult з результатами оцінювання
        """
        pass

    @abstractmethod
    def get_model_info(self) -> ModelInfo:
        """Повертає інформацію про модель"""
        pass

from typing import Dict, Type, List
from .base import BaseAnswerEvaluator, ModelInfo
from .semantic_similarity import SemanticSimilarityEvaluator

class ModelFactory:
    """Фабрика для створення різних моделей оцінювання"""

    _models: Dict[str, Type[BaseAnswerEvaluator]] = {
        "semantic_similarity": SemanticSimilarityEvaluator,
        "default": SemanticSimilarityEvaluator,
    }
```

```
@classmethod
def create_model(cls, model_type: str = "default", **kwargs) -> BaseAnswerEvaluator:
    """
    Створює модель оцінювання

    Args:
        model_type: Тип моделі ('semantic_similarity', 'default')
        **kwargs: Додаткові параметри для ініціалізації моделі

    Returns:
        Екземпляр оцінювача
    """
    if model_type not in cls._models:
        raise ValueError(f"Unknown model type: {model_type}. Available: {list(cls._models.keys())}")

    model_class = cls._models[model_type]
    return model_class(**kwargs)

@classmethod
def get_available_models(cls) -> List[str]:
    """Повертає список доступних моделей"""
    return list(cls._models.keys())

@classmethod
def get_models_info(cls) -> Dict[str, ModelInfo]:
    """Повертає детальну інформацію про всі доступні моделі"""
    models_info = {}
    for model_name, model_class in cls._models.items():
        try:
            # Створюємо тимчасовий екземпляр для отримання info
            temp_instance = model_class()
            models_info[model_name] = temp_instance.get_model_info()
        except Exception as e:
            # Якщо не вдалося створити екземпляр, повертаємо базову інформацію
            models_info[model_name] = ModelInfo(
                name=model_class.__name__,
                type="unknown",
                version="unknown",
```

```
                language="unknown",
                description=f"Error loading model info: {str(e)}"
            )
        return models_info

    @classmethod
    def register_model(cls, name: str, model_class: Type[BaseAnswerEvaluator]):
        """Реєструє нову модель"""
        if not issubclass(model_class, BaseAnswerEvaluator):
            raise TypeError("Model class must inherit from BaseAnswerEvaluator")
        cls._models[name] = model_class
```

```
'use client';

import {
  Card,
  Typography,
  Radio,
  Input,
  Button,
  Form,
  Space,
  Spin,
  Checkbox,
} from 'antd';
import { useParams, useRouter } from 'next/navigation';
import { useSubmitSurvey } from '@/lib/hooks/useSubmitSurvey';
import {
  QuestionTypeEnum,
  QuestionTypeTestAnswers,
  type SubmitSurveyInputDto,
} from '@/types/api';
import { useSharedSurvey } from '@/lib/hooks/useSharedSurvey';

const { Title, Text } = Typography;
const { TextArea } = Input;

interface FormData {
```

```
  answers: Record<string, string>;
}

export default function SharedSurveyPage() {
  const params = useParams<{ id: string }>();
  const router = useRouter();
  const { data: survey, isLoading } = useSharedSurvey(params?.id ?? '');
  const { submitSurvey, isSubmitting } = useSubmitSurvey(
    survey?.sharedId ?? '',
  );
  const [form] = Form.useForm<FormData>();

  const onFinish = async (values: { answers: Record<string, string> }) => {
    if (isSubmitting) return;

    const questions: SubmitSurveyInputDto['questions'] = Object.entries(
      values.answers,
    ).map(([questionId, answers]) => ({
      questionId,
      answers: Array.isArray(answers) ? answers : [answers],
    }));

    await submitSurvey({ questions });
    router.push('/surveys');
  };

  if (isLoading) {
    return (
      <div
        style={{ display: 'flex', justifyContent: 'center', padding: '48px' }}
      >
        <Spin size="large" />
      </div>
    );
  }

  if (!survey) {
    return (
```

```
    <div style={{ textAlign: 'center', padding: '48px' }}>
      <Title level={4}>Опитування не знайдено</Title>
    </div>
  );
}

return (
  <div style={{ maxWidth: 800, margin: '0 auto', padding: '24px' }}>
    <Card>
      <Title level={2}>{survey.title}</Title>
      <Text type="secondary">{survey.description}</Text>

      <Form
        form={form}
        onFinish={onFinish}
        layout="vertical"
        style={{ marginTop: 24 }}
      >
        {survey.questions.map((question, index) => (
          <Form.Item
            key={question._id}
            label={<Text strong>{`${index + 1}. ${question.title}`}</Text>}
            name={['answers', question._id]}
            rules={[
              {
                required: true,
                message: 'Будь ласка, дайте відповідь на це питання',
              },
            ]}
          >
            {question.type === QuestionTypeEnum.Test ? (
              (() => {
                const isMultiple =
                  (
                    question.answerData as QuestionTypeTestAnswers
                  ).answers.reduce(
                    (acc, option) => acc + (option.isCorrect ? 1 : 0),
                    0,
```

```
        ) > 1;

      if (isMultiple) {
        return (
          <Checkbox.Group>
            <Space direction="vertical">
              {(
                question.answerData as QuestionTypeTestAnswers
              ).answers?.map((option) => (
                <Checkbox key={option.answer} value={option.answer}>
                  {option.answer}
                </Checkbox>
              ))}
            </Space>
          </Checkbox.Group>
        );
      }

      return (
        <Radio.Group>
          <Space direction="vertical">
            {(
              question.answerData as QuestionTypeTestAnswers
            ).answers?.map((option) => (
              <Radio key={option.answer} value={option.answer}>
                {option.answer}
              </Radio>
            ))}
          </Space>
        </Radio.Group>
      );
    })()
  ) : (
    <TextArea rows={4} placeholder="Введіть вашу відповідь" />
  )}
 </Form.Item>
))}
```

```
        <Form.Item>
          <Button
            type="primary"
            htmlType="submit"
            disabled={isSubmitting}
            loading={isSubmitting}
          >
            Надіслати відповіді
          </Button>
        </Form.Item>
      </Form>
    </Card>
  </div>
 );
}
```