Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ ГЕНЕРУВАННЯ ТЕСТІВ НАВЧАЛЬНИХ ДИСЦИПЛІН»

Виконав: студент 4 курсу, групи 2ІСТ-21б спеціальності 126 – «Інформаційні системи та технології»
_____ Денис ЯБЛОНСЬКИЙ

Керівник: PhD, асистент каф. САІТ
_____Арсен ЛОСЕНКО
«05» 06 2025 р.

Рецензент: к.т.н., доц. каф. КН
_____ Олексій СЛАГІН
«09» 06 2025 р.

Допущено до захисту

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«06» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма – Прикладні інформаційні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"20" 05 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Яблонському Денису Анатолійович

1. Тема роботи: «Інтелектуальна технологія генерування тестів навчальних дисциплін»
керівник роботи: Лосенко А.В., PhD, асистент каф. САІТ
затверджені наказом ВНТУ від «20» 05 2025 року № 97
2. Термін подання студентом роботи 31.05.25 р.
3. Вихідні дані до роботи:
1) Дані про теми навчальних дисциплін для генерування навчальних тестів
2) Дані про формування інструкцій для роботи з великими мовними моделями
4. Зміст текстової частини:
1) Аналіз предметної області.
2) Огляд технологій та інструментів розробки.
3) Розроблення інтелектуальної технології для автоматизованого створення навчальних тестів.
5. Перелік ілюстративного матеріалу:
1) Структура компонентів інтелектуальної технології;
2) Діаграма модулів реалізації бізнес-логіки;
3) Use case діаграма інтелектуальної технології;
4) UML-діаграма послідовності механізму кешування у вебдодатку;
5) UML-діаграма послідовності механізму генерації тестів та після отримання його для проходження;

6) Архітектура інтелектуальної технології;
7) Блок-схема алгоритму модулю генерування навчального тесту.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн

7. Дата видачі завдання 28.03.25 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Прим
		початок	закінчення	
1	Аналіз предметної області	01.04	15.04	вик
2	Аналіз існуючих технологій генерування тестових завдань	15.04	30.04	вик
3	Вибір оптимальних інформаційних технологій для розроблення	01.05	10.05	вик
4	Розроблення інтелектуальної технології для автоматизованого створення навчальних тестів	10.05	20.05	вик
5	Оформлення матеріалів до захисту БКР	20.05	30.05	вик

Студент _____ Денис ЯБЛОНСЬ

Керівник роботи _____ Арсен ЛОСЕН

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 90 сторінок формату А4, на яких є 42 рисунки, список використаних джерел містить 27 найменувань.

Метою роботи є підвищити швидкість та якість генерування тестів для навчальних дисциплін шляхом створення інтелектуальної інформаційної технології на основі великої мовної моделі.

В роботі наведено розроблення інтелектуальної технології на основі React.js для клієнтської частини та Firebase для серверної інфраструктури. Реалізовано інтеграцію з великими мовними моделями для автоматичного створення тестових завдань. Розроблено архітектуру з впровадженням механізмів кешування для забезпечення офлайн-режиму роботи. Створена технологія суттєво скорочує час на підготовку навчальних тестів.

Ключові слова: інтелектуальна технологія, тести, тести з ШІ, штучний інтелект, готові тести, React.js, Firebase

ABSTRACT

The bachelor's qualification thesis consists of 90 pages in A4 format, containing 42 figures, and the list of references includes 27 items.

The aim of the work is to increase the speed and quality of test generation for educational disciplines by creating intelligent information technology based on a large language model.

The work presents the development of intelligent technology based on React.js for the client side and Firebase for server infrastructure. Integration with large language models for automatic creation of test tasks has been implemented. An architecture with caching mechanisms has been developed to ensure offline operation. The created technology significantly reduces the time required for preparing educational tests.

Keywords: intelligent technology, tests, AI tests, artificial intelligence, ready-made tests, React.js, Firebase.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні реалії освітнього процесу в Україні характеризуються зростанням самоосвіти, що викликане як постійними проблемами організації традиційного навчання, так і обмеженою кількістю високоякісних ресурсів. Онлайн-формат стає основним способом отримання знань, що надає більше можливостей для реалізації своїх перспектив, це дозволяє студентам активно впроваджувати цифрові інструменти у свою роботу.

Більшість рішень для формування тестових завдань вимагає стабільного інтернет-з’єднання та не забезпечує необхідної гнучкості для адаптації матеріалів до потреб користувачів. В результаті освітнім фахівцям доводиться витрачати значну кількість часу на створення тестових завдань власноруч, що є особливо складним завданням для навчальних закладів або компаніям та тим хто навчається віддалено з великою кількістю інформації.

Це підкреслює необхідність розробки нової технології, яка автоматизує процес створення тестів із використанням штучного інтелекту. Такий підхід дозволить працювати без необхідності постійного підключення до мережі та забезпечить інтуїтивну зрозумілість для користувачів, оптимізуючи роботу і сприяючи більш адаптивній перевірці знань.

Мета і задачі дослідження. Метою дослідження є підвищити швидкість та якість генерування тестів для навчальних дисциплін шляхом створення інтелектуальної інформаційної технології на основі великої мовної моделі.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- провести аналіз існуючих методів і технологій генерування тестових завдань для навчальних дисциплін;
- дослідити можливості великих мовних моделей для автоматичного створення якісних тестових питань різних типів;
- розробити архітектуру інтелектуальної технології генерування тестів з урахуванням специфіки навчальних дисциплін;

- реалізувати вебдодаток з інтеграцією AI-сервісу для генерування тестових завдань на основі навчальних матеріалів;
- провести перевірку розробленої технології на прикладі різних навчальних дисциплін.

Об'єктом дослідження є процеси генерування тестових завдань для оцінювання знань з навчальних дисциплін у системі вищої освіти.

Предметом дослідження є засоби розроблення інтелектуальних технологій генерування тестів з використанням великих мовних моделей, алгоритми адаптації тестових завдань до специфіки навчальних дисциплін та механізми забезпечення якості автоматично згенерованих тестів.

Публікації. За результатами даної роботи була зроблена доповідь на тему «Проектування освітньої платформи «THINK IT» на базі сучасних веб-технологій» на Міжнародній науково-практичній інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (2025) з публікацією тез.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування актуальності розробки інтелектуальної технології генерування тестів навчальних дисциплін

У сучасних умовах освіти, де швидкість та ефективність оцінювання набувають особливого значення, розробка інтелектуальної технології генерації тестів для навчальних дисциплін є надзвичайно актуальною [1,2].

Запропоноване рішення покладатиметься на принципи модульності та масштабованості, що дозволить адаптувати технологію до різних навчальних середовищ і оперативно відповісти на вимоги часу. Для створення клієнтської частини традиційно застосовуються frontend-фреймворки, що забезпечують зручність, продуктивність і гнучкість розробки.

Розроблювана інтелектуальна технологія передбачає клієнтську та серверну частину. Додатково планується інтеграція прогресивних вебтехнологій (PWA [3]) для забезпечення офлайн-режиму, що стане ключовим аспектом стабільної роботи за умов нестабільного доступу до інтернету.

Ключовою особливістю цієї технології є використання гнучкого механізму кешування, який аналізує патерни використання тестових завдань і автоматично завантажує найбільш актуальні підходи для її реалізації за допомогою промптів. Передбачений також механізм завантаження власних матеріалів. Такий підхід значно скорочує час реакції системи порівняно з традиційними методами генерації тестів і забезпечує безперебійну роботу незалежно від стабільності інтернет-з'єднання.

Інтелектуальна технологія підтримуватиме різні моделі розгортання — від хмарних сервісів до локальних серверних рішень для навчальних закладів, що гарантує гнучкість впровадження у будь-яких організаційних та технічних умовах. Адаптивний дизайн забезпечить ефективну роботу на різноманітних пристроях – від мобільних телефонів до настільних комп’ютерів.

Комплексне застосування таких підходів дозволить значно скоротити час створення якісних тестових завдань у порівнянні з ручними методами, забезпечуючи при цьому високий рівень якості, об'єктивності та різноманітності контенту [4]. Цей чинник є вирішальним для сучасної освіти, де інноваційні технології мають стати ключовим інструментом підтримки самоосвіти та оптимізації навчального процесу.

1.2 Аналіз існуючих технологій генерування тестових завдань

У сфері інтелектуальних технологій існує кілька аналогів, здатних скоротити процес створення тестових завдань, що значно спрощує та прискорює процес оцінювання знань студентів. Сучасні цифрові платформи забезпечують широкий спектр можливостей для створення, налаштування та проведення тестування, що робить їх незамінними інструментами як у сфері освіти, так і в корпоративному навчанні. Розглянемо три найбільш популярні платформи, аналізуючи їх функціональні можливості, особливості впровадження та експлуатації, а також потенціал у генерації тестів для різних навчальних дисциплін.

Kahoot! [5] – це широко відома інтерактивна платформа, яка активно використовується як навчальними закладами, так і корпоративним сектором, а також окремими викладачами для створення вікторин та опитувань у форматі гри. Основна концепція технології заснована на гейміфікації навчального процесу, що сприяє більш активному залученню учасників завдяки елементу змагання. Платформа дозволяє створювати різноманітні типи завдань: від вікторин з множинним вибором відповідей до опитувань, обговорень чи навіть пазлів.

Під час сесій у режимі реального часу користувачі можуть використовувати власні пристрої для взаємодії, а результати одразу ж демонструються на загальному екрані, що підвищує ефективність залучення учасників. Крім цього, платформа пропонує можливість проходження

домашніх завдань у асинхронному режимі, що розширює гнучкість навчального процесу. Система забезпечує базову аналітику результатів, надаючи можливість експортувати дані для подальшого аналізу. Яскравий та динамічний інтерфейс, доповнений музичним супроводом, створює мотивуючу атмосферу та робить процес тестування більш захопливим (рис 1.1).

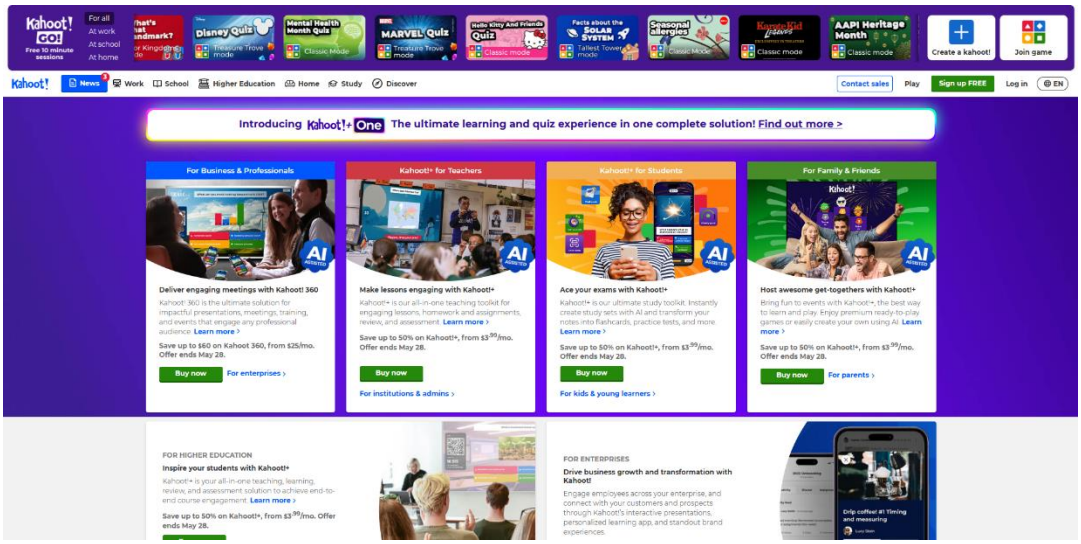

Рисунок 1.1 – Kahoot!

Попри свої переваги, Kahoot! має ряд обмежень. Платформа орієнтована на короткі інтерактивні вікторини, що робить її менш придатною для комплексного оцінювання знань. Крім того, технологія потребує стабільного інтернет-з'єднання, що може створити труднощі при використанні в умовах ненадійного доступу до мережі. Безкоштовна версія має обмеження щодо кількості учасників та доступних функцій, а ігровий формат не завжди відповідає потребам серйозного академічного тестування.

Classmarker [6] – це професійна онлайн-система тестування, призначена для навчальних закладів та корпоративного сектора. Платформа пропонує розширений набір форматів питань, серед яких множинний вибір, питання True/False, заповнення пропусків та есе з можливістю ручної перевірки. Особливу увагу технологія приділяє безпеці: система використовує

рандомізацію порядку питань, встановлює часові обмеження та блокує доступ до інших ресурсів під час тестування, що мінімізує ризик списування.

Classmarker забезпечує детальну аналітику результатів, надаючи глибокі звіти з можливістю експорту даних у різних форматах. Крім цього, платформа дозволяє створювати групи чи класи з окремими рівнями доступу та правами користувачів. За рахунок API інтеграція з іншими технологіями відбувається без труднощів, що значно розширює можливості використання (рис 1.2).

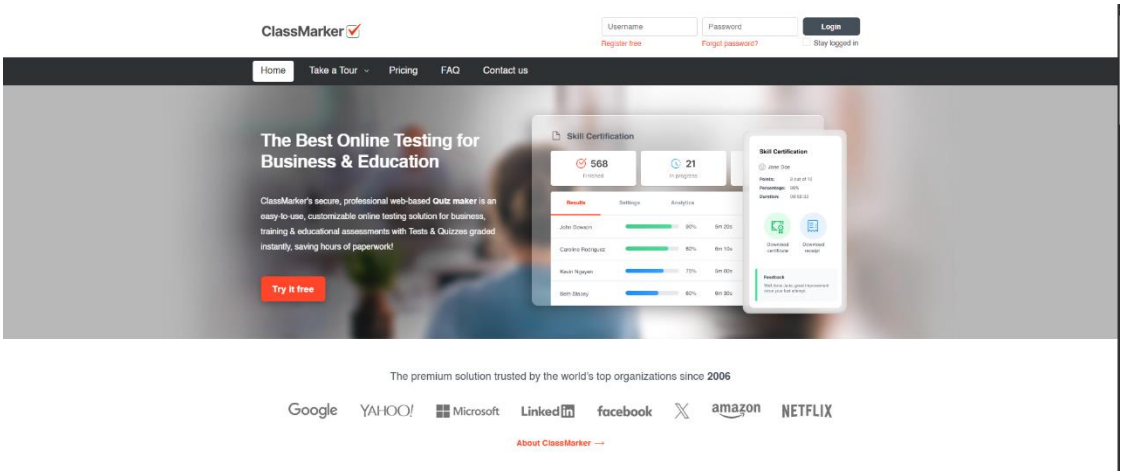

Рисунок 1.2 – Classmarker

Серед недоліків варто відзначити те, що повний функціонал доступний лише за платною підпискою, що може стати бар'єром для невеликих навчальних закладів та індивідуальних користувачів. Інтерфейс системи є досить складним, що потребує часу для його освоєння. Відсутність офлайн-доступу також може створити труднощі для користувачів у регіонах з нестабільним інтернет-з’єднанням. Крім того, безкоштовна версія має обмежені можливості щодо налаштування зовнішнього вигляду тестів.

Microsoft Forms [7] входить до екосистеми Microsoft 365 і пропонує простий у використанні інструмент для створення опитувань, тестів та збору зворотного зв’язку. Завдяки підтримці різних типів питань — від вибору з варіантів до шкал рейтингу, текстових відповідей і навіть завантаження файлів — сервіс дозволяє швидко створювати тестові завдання та анкетування.

Глибока інтеграція з іншими сервісами Microsoft, такими як Teams, SharePoint і Excel, сприяє ефективному використанню у навчальних та корпоративних середовищах. Простий та зрозумілий інтерфейс дозволяє швидко налаштовувати форми, додаючи теми та логотипи. Крім того, платформа автоматично оцінює відповіді та надає базову статистику з візуалізацією даних, що робить аналіз результатів більш зручним (рис 1.3).

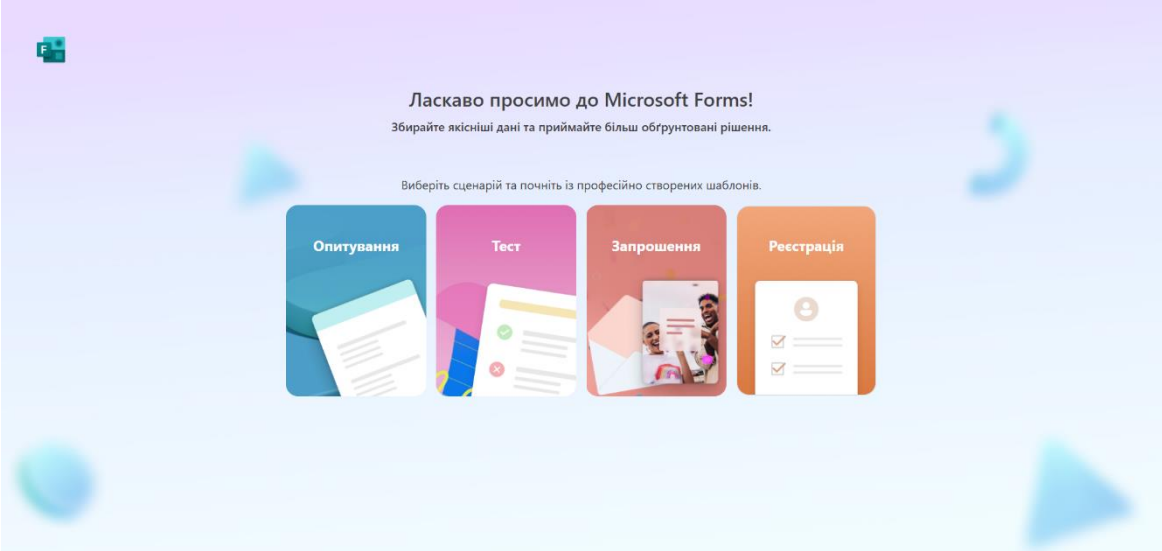

Рисунок 1.3 – Microsoft Forms

Попри переваги, Microsoft Forms має певні обмеження. Набір типів запитань є менш різноманітним у порівнянні зі спеціалізованими платформами тестування. Можливості персоналізації та налаштування обмежені базовими функціями. Крім того, сервіс потребує постійного інтернет-з’єднання та не підтримує офлайн-режим. Недостатня глибина аналізу результатів та неможливість довгострокового відстеження прогресу студентів також можуть обмежувати його застосування у професійних освітніх сценаріях.

1.3 Формування вимог до інтелектуальної технології генерування тестів навчальних дисциплін

Розробка ефективної освітньої платформи для тестування знань вимагає ретельного аналізу потреб користувачів та обмежень існуючих технологічних рішень. Після детального вивчення предметної області та дослідження аналогічних технологій, було встановлено, що більшість існуючих платформ мають суттєві недоліки: залежність від постійного інтернет-з'єднання, надмірна складність інтерфейсу, обмежені можливості розгортання та низька продуктивність на пристроях з обмеженими ресурсами.

Зважаючи на виявлені проблеми, розроблення інтелектуальної технології для генерування навчальних тестів спрямоване на створення універсального, продуктивного та доступного інструменту для тестування знань, що зможе функціонувати в різноманітних технічних умовах. Особливий акцент планується зробити на забезпеченні можливості роботи в офлайн-режимі при проходженні тестів, оптимізації швидкодії та створенні інтуїтивно зрозумілого інтерфейсу.

Передбачається впровадження інноваційних механізмів кешування даних та попереднього завантаження контенту, що дозволить суттєво підвищити швидкість роботи системи та забезпечити її функціонування в умовах нестабільного інтернет-з'єднання. Технологія має бути спроектована таким чином, щоб забезпечити гнучкість розгортання – від хмарного сервісу до локальних серверів навчальних закладів та окремих комп'ютерів користувачів.

Ґрунтуючись на проведеному аналізі існуючих освітніх інструментів в цілях тестування знань, для розробленої інтелектуальної технології була обрана назва "THINK IT", що перекладається як "Обміркуй це" і відображає концепцію даної інтелектуальної технології, що розробляється для вдосконалення тестування знань. Така технологія буде орієнтована на адаптивний підхід, інтеграцію аналітичних інструментів та персоналізовані

рекомендації. Її мета — не просто оцінка знань, а допомога користувачам у виявленні прогалин й удосконаленні розуміння матеріалу через інтелектуальний аналіз відповідей. Використання інтерактивних методів навчання забезпечить ефективне засвоєння інформації та сприятиме розвитку критичного мислення.

Вимоги до майбутньої технології "THINK IT" формуються з урахуванням потреб широкого кола користувачів – від учнів та студентів до викладачів та освітніх організацій чи компаній. Функціональні та нефункціональні вимоги мають бути спрямовані на забезпечення комплексного підходу до процесу тестування знань, підвищення його ефективності та створення персоналізованого досвіду навчання.

На основі проведеного аналізу сформульовано наступні групи вимог до майбутньої технології:

Функціональні вимоги:

a) Управління тестами:
    1) Створення, редагування та видалення тестів різних типів;
    2) Підтримка різних форматів тестових завдань (вибір однієї відповіді, відкриті питання);
    3) Категоризація тестів за предметами та типами;
    4) Можливість імпорту тестових матеріалів.
b) Проходження тестів:
    1) Інтерактивний інтерфейс для відповідей на питання;
    2) Автоматична перевірка відповідей та розрахунок результатів;
    3) Можливість перегляду правильних відповідей після завершення тесту;
    4) Підтримка повторного проходження тестів для закріплення знань.
c) Управління користувачами:
    1) Реєстрація та авторизація користувачів;
    2) Налаштування профілю та персоналізація;
    3) Управління доступом до тестів.

d) Аналіз результатів:
    1) Збереження історії проходження тестів;
    2) Детальна статистика успішності;
    3) Відстеження прогресу навчання у часі.
e) Пошук та фільтрація:
    1) Можливість пошуку тестів за назвою, описом, предметом;
    2) Сортування результатів пошуку за різними критеріями;
    3) Швидкий доступ до нещодавно використаних тестів.

Нефункціональні вимоги**:**

a) Продуктивність:
    1) Швидкий час відгуку системи (не більше 2 секунд для основних операцій);
    2) Оптимізоване використання ресурсів пристрою;
    3) Ефективне кешування даних для зменшення мережевого трафіку;
    4) Попереднє завантаження тестів для плавного переходу між ними.
b) Доступність:
    1) Підтримка офлайн-режиму роботи з синхронізацією при відновленні з'єднання;
    2) Адаптивний дизайн для оптимального відображення на різних пристроях;
    3) Підтримка різних браузерів та операційних систем;
    4) Мінімальні вимоги до апаратних ресурсів.
c) Надійність:
    1) Захист від втрати даних при нестабільному з'єднанні;
    2) Механізми відновлення після помилок;
    3) Регулярне резервне копіювання даних.
d) Масштабованість:
    1) Здатність технології працювати з великими обсягами тестів та користувачів;

2) Можливість розширення функціональності без суттєвої переробки архітектури;
3) Модульна структура для легкої інтеграції нових компонентів;
4) Гнучкість налаштування відповідно до розміру організації.

e) Безпека:
1) Захист персональних даних та результатів тестування;
2) Обмеження доступу до конфіденційної інформації;
3) Шифрування передачі даних.

f) Зручність використання:
1) Інтуїтивно зрозумілий інтерфейс без необхідності тривалого навчання;
2) Логічна організація функцій та елементів керування;
3) Мінімальна кількість дій для виконання часто використовуваних завдань.

Особлива увага при розробці інтелектуальної технології "THINK IT" буде приділена впровадженню механізмів кешування та попереднього завантаження тестів, що дозволить значно підвищити швидкодію та забезпечити можливість офлайн-роботи. Технологія зможе автоматично аналізувати способи використання тестів користувачем та завчасно завантажувати найбільш вірогідні для використання тести, що зменшить час очікування та покращить користувацький досвід.

Сформульовані вимоги стануть основою для проектування та розробки системи, забезпечуючи створення ефективного, надійного та зручного інструменту для тестування знань, що відповідатиме сучасним освітнім потребам та технологічним можливостям.

1.4 Висновки

Проведений аналіз демонструє, що впровадження інтелектуальних технологій для генерування тестів у навчальних дисциплінах – це ключовий

крок у цифровій трансформації освіти. Сучасні вимоги до оперативності, об'єктивності та персоналізації оцінок знань спричиняють потребу в інноваційних рішеннях, які забезпечують гнучкість, масштабованість і стабільність освітнього процесу, навіть у умовах нестабільного доступу до інтернету. Інтелектуальна технологія буде має клієнтську й серверну частини, з PWA для офлайн-режиму при нестабільному з’єднанні

Аналіз існуючих аналогів, зокрема Kahoot!, Classmarker і Microsoft Forms, показав, що хоча вони мають корисні функції, жодна з них не забезпечує оптимального балансу між адаптивністю, офлайн-доступом, зручністю інтерфейсу та гнучкістю розгортання. Це підтверджує необхідність створення нової системи, яка зможе подолати ці недоліки й відповідати потребам ширшої аудиторії.

Назва інтелектуальної технології генерування навчальних тестів "THINK IT" (в перекладі — "Обміркуй це") відображає концепцію інтелектуального підходу до тестування знань.

Враховуючи отримані висновки по аналізу, було сформульовано ключові вимоги до інтелектуальної технології генерування навчальних тестів "THINK IT". Вони охоплюють функціональні аспекти — створення тестів, управління користувачами, аналіз результатів — а також нефункціональні: продуктивність, надійність, масштабованість, безпеку та зручність використання. Особливий акцент зроблено на можливості роботи в офлайн-режимі та ефективному кешуванні, що гарантує стабільність системи навіть за нестабільного інтернет-з'єднання.

Таким чином, інтелектуальна технологія генерування навчальних тестів під назвою "THINK IT" — це сучасне рішення, яке буде поєднувати передові технології з реальними освітніми потребами. Його потенціал полягає у підвищенні якості знань, гнучкому адаптуванні до навчальних сценаріїв та активному залученні користувачів через інтелектуальний підхід до тестування.

2 ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Огляд та аналіз великих мовних моделей для генерування навчальних тестів

Великі мовні моделі (Large Language Models, LLM) [8] є ключовою технологією штучного інтелекту, яка активно використовується для автоматизації освітніх процесів. Їх здатність генерувати текстовий контент на основі заданих інструкцій та контексту відкриває нові можливості для персоналізації навчання, створення адаптивних тестів та аналізу знань учнів.

Серед розглянутих альтернатив для реалізації функцій генерації тестів були: Google Gemini [9], Anthropic Claude [10], ChatGPT [11] (рис 2.1), а також локальні моделі, такі як LLaMA [12] і Mistral [13] (рис 2.2).

Рисунок 2.1 – Логотипи розглянутих сервісів з великими мовними моделями

Google Gemini — мультимодальна велика мовна модель від Google DeepMind, інтегрована з екосистемою Google. Вона пропонує потужні можливості для обробки природної мови та сумісність із Google Search. Її переваги включають мультимодальність і можливість використання в освітніх платформах для аналізу текстів і створення навчальних матеріалів. Недоліки

включають залежність від екосистеми Google, обмеження доступу до розширених функцій для безкоштовних користувачів і точність відповідей.

Anthropic Claude — модель, яка демонструє високу якість генерації контенту та розширені можливості для роботи з довгими текстами. В освітніх додатках модель може бути використана для створення складних текстових завдань, але має складності з розумінням контексту при роботі з рідкісними мовами або специфічними діалектами.

ChatGPT — генеративний чат-бот від OpenAI [14], який вирізняється адаптивністю, здатністю підтримувати діалог і широким спектром запитів. У освітніх додатках ChatGPT може використовуватися для створення тестових завдань, генерації пояснень до відповідей та адаптації контенту до рівня знань учнів. Недоліки включають відсутність емоційного інтелекту та креативного мислення.

Локальні моделі (LLaMA, Mistral) — відкриті рішення, які можуть бути розгорнуті на власних серверах. Вони забезпечують повний контроль над даними та процесом генерації, що є важливим для конфіденційності освітніх матеріалів. Однак ці моделі вимагають значних обчислювальних ресурсів та технічних знань для впровадження.

Рисунок 2.2 – Логотипи сервісів для роботи з локальними LLM

Серед розглянутих варіантів було обрано сервіс ChatGPT від OpenAI, у зв’язку з його високою якістю генерації тексту українською мовою, стабільністю API, доступністю функціоналу без значних фінансових витрат, а також гнучкими можливостями налаштування параметрів генерації для освітніх потреб.

Для реалізації функціональності генерації навчальних тестів в інтелектуальній технології "THINK IT" ключовим компонентом стане інтеграція з ChatGPT. Це забезпечить високоякісну генерацію тестових завдань різних типів і складності відповідно до освітніх стандартів. ChatGPT дозволяє працювати на основі матеріалу від користувача, використовуючи технології обробки природної мови для створення структурованого контенту з урахуванням контексту.

Особливо важливою є підтримка української мови, що забезпечує створення якісних освітніх матеріалів для вітчизняної аудиторії. Інтеграція з ChatGPT дозволить значно прискорити процес створення тестових матеріалів, забезпечити їх різноманітність, варіативність і адаптивність до конкретних освітніх потреб. Користувачі зможуть отримувати персоналізовані тести, що відповідають їх рівню знань і специфіці навчальної програми.

2.2 Огляд технологій для розроблення клієнтської частини інтелектуальної технології

Для розроблення клієнтської частини технології генерування тестів навчальних дисциплін було обрано сучасний технологічний стек, що забезпечує високу адаптивність до різних предметних областей, продуктивність та зручність розробки освітніх інтерфейсів.

Під час огляду інструментів для розробки було розглянуто низку популярних середовищ, серед яких IntelliJ IDEA, PyCharm, Eclipse та Visual Studio Code. Ці середовища забезпечують широкий спектр функцій для

створення, тестування та організації програмного коду, що дозволяє адаптувати вибір до специфічних вимог кожного проекту.

Для розробки було обрано Visual Studio Code [15] — сучасний текстовий редактор, який вирізняється високою гнучкістю та широкими можливостями налаштування. Завдяки підтримці численних мов програмування, інтеграції з системами контролю версій та великій кількості доступних розширень, Visual Studio Code легко адаптується до потреб будь-якого проекту. Його простота у використанні, а також потужні функції, такі як автодоповнення коду та інтегрований дебагер, сприяють підвищенню продуктивності навіть під час роботи над складними завданнями.

Популярні фреймворки для розробки вебдодатків, такі як Vue, Angular, Svelte та React, були розглянуті під час аналізу технологій. Кожен із них пропонує унікальні можливості для створення сучасних інтерфейсів, що дозволяє обрати оптимальне рішення для конкретних потреб проекту.

Як фреймворк для реалізації клієнтської частини додатку було обрано React.js [16] (рис 2.3) — один із провідних інструментів мови JavaScript для створення користувацьких інтерфейсів. React базується на компонентній архітектурі, що дозволяє створювати багаторазово використовувані елементи інтерфейсу та забезпечує ефективне управління станом додатку. Крім того, React було обрано завдяки його здатності оптимізувати процес оновлення інтерфейсу за допомогою віртуального DOM (об'єктної моделі документа) — внутрішнього представлення сторінки в пам’яті, з яким React працює перед внесенням змін у реальний DOM.

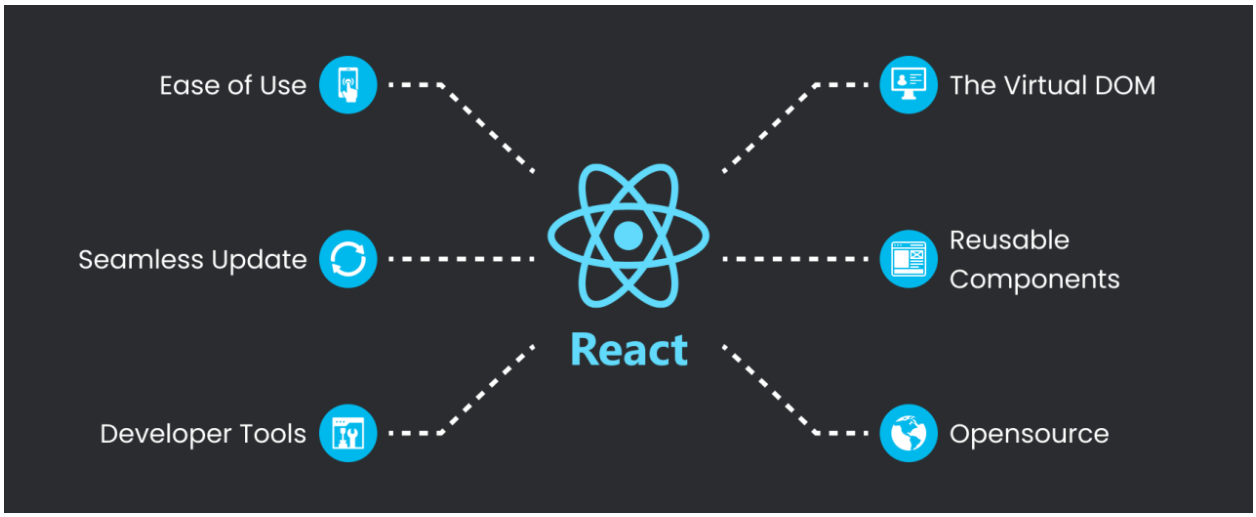

Рисунок 2.3 – React та його можливості

У свою чергу, TypeScript [17] є надмножиною JavaScript із підтримкою статичної типізації (рис 2.4). Що напряму дозволяє виявляти помилки на етапі розробки, підвищує читабельність коду та спрощує його підтримку [18]. Для додатку зі складною структурою даних тестів TypeScript є критично важливим, оскільки забезпечує точне визначення типів і покращує загальну якість програмного коду.

Рисунок 2.4 – TypeScript та його інструменти

Для оформлення клієнської частини було обрано Tailwind CSS [19] — utility-first CSS фреймворк, який надає велику кількість невеликих класів-утиліт, кожен із яких відповідає за окрему стилістичну властивість. Замість написання складних стилів або створення власних класів, просто додає потрібні утиліти до HTML-елементів. Це дозволяє швидко створювати сучасні, адаптивні інтерфейси. На відміну від традиційних CSS-фреймворків, Tailwind пропонує низькорівневі утиліти, які дають змогу будувати індивідуальні дизайни без написання додаткового CSS-коду (рис 2.5). Такий підхід забезпечує консистентність стилю та пришвидшує розробку.

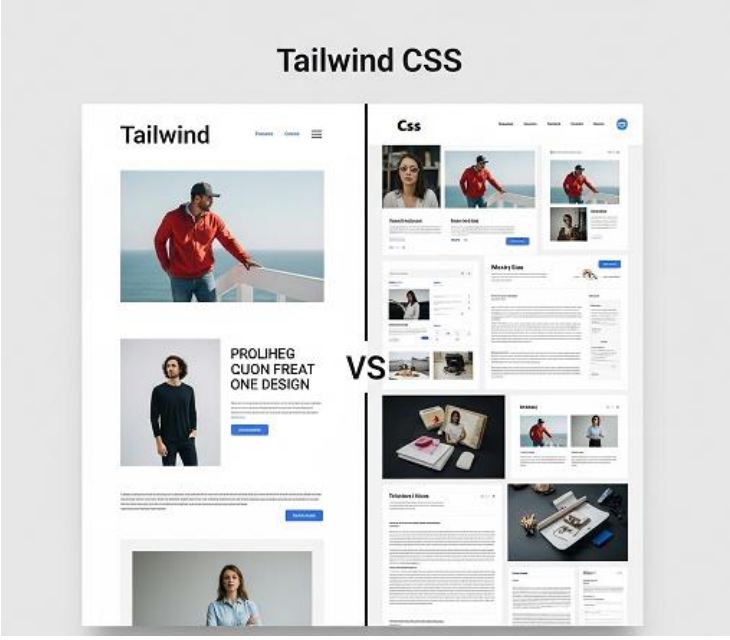

Рисунок 2.5 – Порівняння Tailwind CSS та простого CSS

Для розгортання клієнтської частини було використано Vite [20] — сучасний інструмент збирання проєктів, який забезпечує надзвичайно швидку ініціалізацію та гаряче оновлення модулів (HMR) (рис 2.6). Vite використовує переваги нативних ES-модулів у браузері на етапі розробки, що дозволяє уникнути повної перекомпіляції коду при кожній зміні. Це значно скорочує час очікування та підвищує продуктивність роботи. Завдяки підтримці сучасних фреймворків, таких як React, і простоті налаштування, Vite є оптимальним вибором для створення швидких і масштабованих фронтенд-додатків.

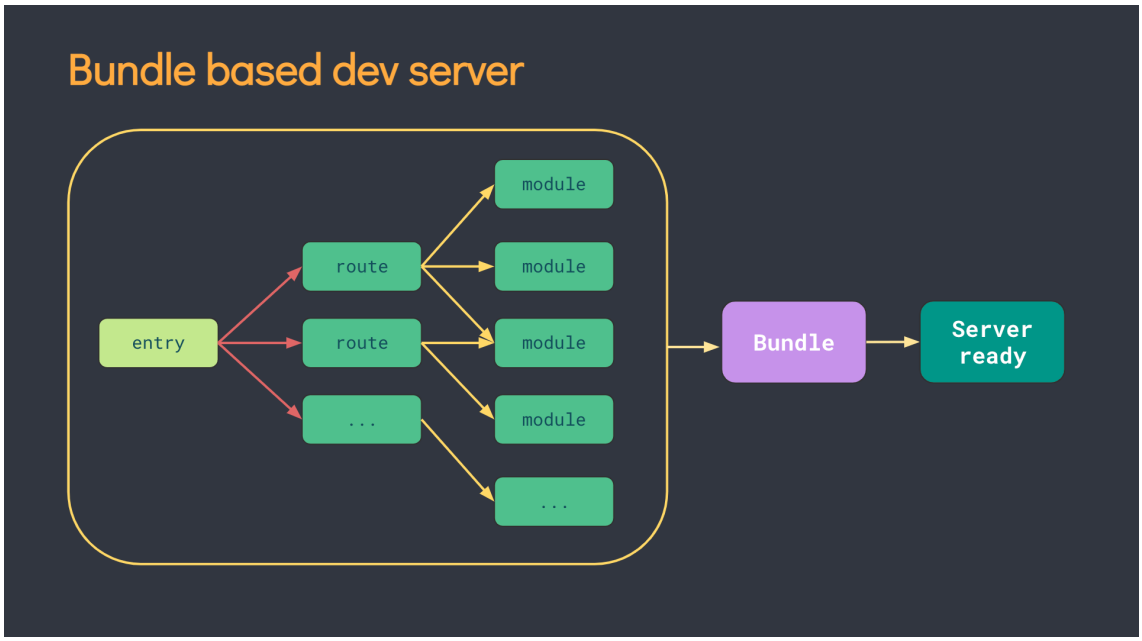

Рисунок 2.6 – Схема роботи Vite

Для підвищення зручності використання та доступності додатку було обранно підтримку технології Progressive Web App (PWA) — підходу, що дозволяє вебдодатку працювати майже як нативний мобільний застосунок. Завдяки використанню Service Workers, додаток може функціонувати в офлайн-режимі, надсилати push-сповіщення та бути встановленим на пристрій користувача безпосередньо з браузера (рис 2.7). Це особливо важливо для освітніх рішень, де стабільне інтернет-з'єднання не завжди гарантоване. Такий підхід забезпечує кращу взаємодію з користувачем і робить додаток більш надійним у різних умовах.

Рисунок 2.7 – Приклади використання PWA

Компонентна архітектура додатку дозволяє структурвати програмні продукти у логічно розділені модулі, що спрощує розробку, тестування та підтримку. Каталог pages відповідає за основні маршрути та представлення сторінок, components містить багаторазові елементи інтерфейсу, lib — бізнес-логіку та взаємодію з API. Крім того, contexts використовуються для централізованого управління глобальним станом, а utils — для зберігання допоміжних функцій, що підвищують повторне використання коду[21].

2.3 Огляд технологій для розроблення серверної частини інтелектуальної технології

Firebase[22] (рис 2.8) — це хмарна платформа від Google, яка забезпечує комплексне Backend-as-a-Service (BaaS) рішення для розробки веб та мобільних додатків. Вона пропонує готову серверну інфраструктуру, що дозволяє розробникам зосередитися на створенні функціональності додатку, не витрачаючи ресурси на налаштування та підтримку серверів.

Рисунок 2.8 – Логотип Firebase

Ця платформа включає сервіси для управління даними, автентифікації користувачів, хмарного зберігання, аналітики та реального часу синхронізації (рис 2.9). Завдяки автоматичному масштабуванню Firebase забезпечує стабільну роботу додатків навіть за високого навантаження. Для освітніх рішень це означає можливість миттєвого оновлення результатів тестування, синхронізацію даних між користувачами та підтримку багатокористувацького режиму.

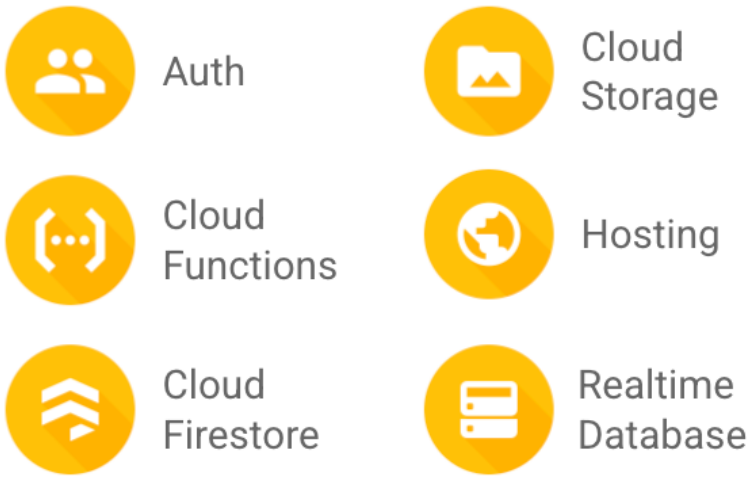

Рисунок 2.9 – Перелік сервісів Firebase

Firebase Hosting забезпечує швидкий і безпечний хостинг для статичних вебсайтів, використовуючи глобальну CDN мережу для оптимальної доставки контенту. Цей сервіс інтегрований з іншими продуктами Firebase і підтримує автоматичне розгортання через CI/CD процеси, що значно спрощує управління додатком.

Firebase Functions дозволяє виконувати серверний код у відповідь на події, такі як HTTP-запити або зміни в базі даних, без необхідності підтримки власних серверів. Functions автоматично масштабуються, що робить їх ідеальним рішенням для обробки даних, інтеграції з зовнішніми API а особливо з GPT-3.5.

Firebase Authentication — це сервіс, який забезпечує просту та безпечну аутентифікацію користувачів через email та пароль. Він дозволяє зберігати

облікові записи користувачів у захищеному середовищі та автоматично генерує JWT токени для авторизації. Це спрощує інтеграцію з іншими сервісами та забезпечує високий рівень безпеки додатків, зосереджуючи увагу на зручності для користувачів.

Firebase Firestore — це сучасна NoSQL документна база даних, яка забезпечує синхронізацію даних у реальному часі між різними пристроями (рис 2.10). Завдяки автоматичному масштабуванню відповідно до навантаження, Firestore гарантує стабільну роботу навіть за високої кількості одночасних запитів. Можливості для запитів та індексації дозволяють ефективно працювати з великими обсягами даних, забезпечуючи швидкий доступ до необхідної інформації.

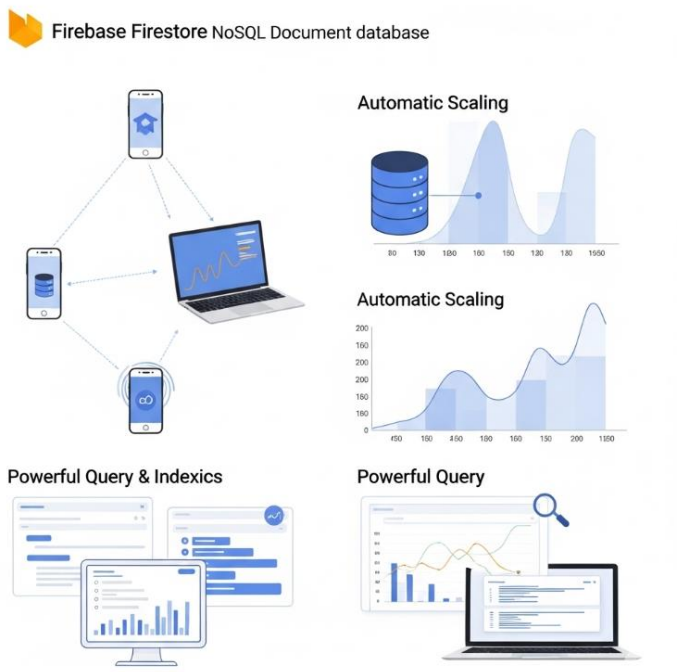

Рисунок 2.10 – Приклад синхронізації даних у реальному часі

Архітектура системи базується на принципах безсерверних обчислень (serverless) [23], які забезпечують автоматичне масштабування відповідно до навантаження, оптимізацію витрат завдяки відсутності простоювання ресурсів, а також значне спрощення процесу підтримки інфраструктури. Такий підхід дозволяє зосередитися на розробці функціональності додатку,

мінімізуючи технічні складнощі та витрати на адміністрування серверів (рис 2.11).

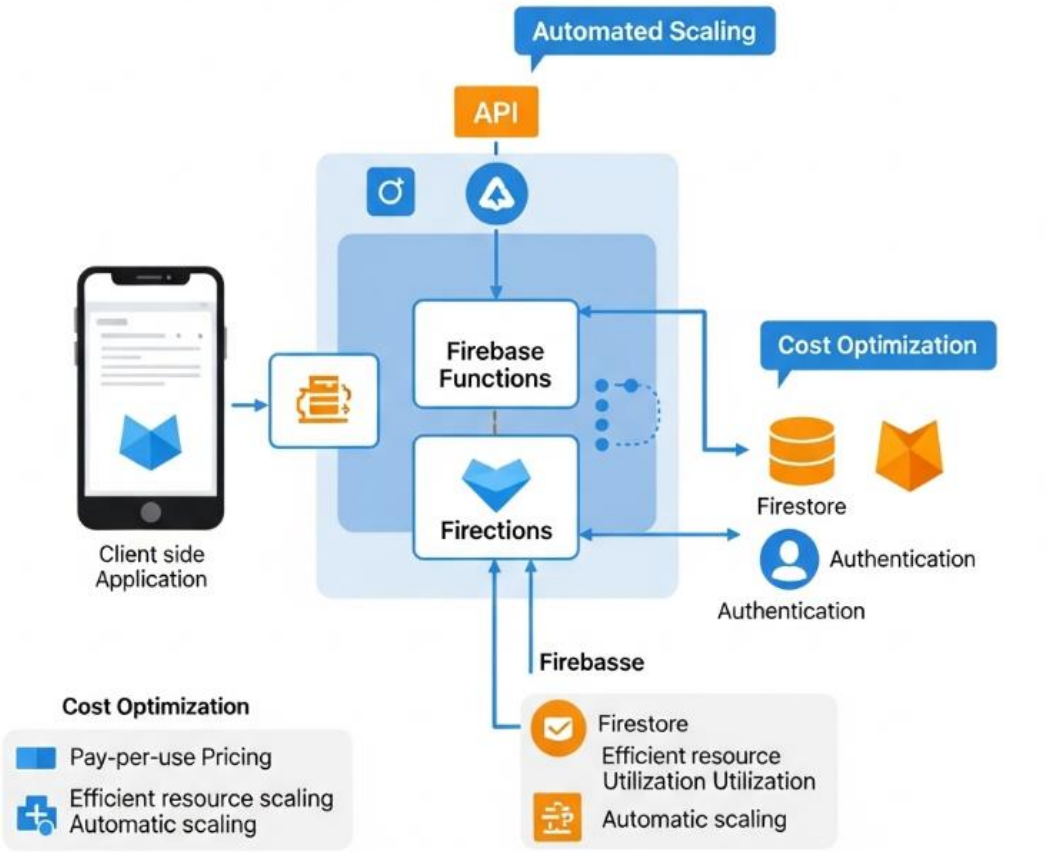

Рисунок 2.11 – Проста безсерверна архітектура з Firebase Functions

Для інтеграції з OpenAI API була реалізована безпечна система передачі запитів через Firebase Functions. Цей підхід дозволяє приховати API ключі від клієнтського коду, забезпечуючи додатковий рівень безпеки. Крім того, використання Firebase Functions дає можливість контролювати швидкість запитів, запобігаючи перевантаженню системи та забезпечуючи стабільну роботу додатку.

2.4 Висновки

У даному розділі було проведено детальний огляд великих мовних моделей для генерації навчальних тестів, а також аналіз сучасних технологій і інструментів для розробки вебдодатків. В результаті дослідження було обрано найбільш доцільні рішення для впровадження в інтелектуальну технологію генерування навчальних тестів “THINK IT”.

Серед розглянутих мовних моделей, таких як Google Gemini, Anthropic Claude, ChatGPT та локальні моделі (LLaMA, Mistral), перевага була надана ChatGPT версії 3.5 від OpenAI через його гарну підтримку української мови, що забезпечує створення якісних освітніх матеріалів для вітчизняної аудиторії. Google Gemini, хоча й пропонує мультимодальність та інтеграцію з екосистемою Google, але має обмеження у доступі до розширених функцій для безкоштовних користувачів та залежність від екосистеми Google, що може створювати труднощі для інтеграції з іншими платформами. Anthropic Claude демонструє високу якість генерації контенту, але має складності з розумінням контексту при роботі з рідкісними мовами. Локальні моделі, такі як LLaMA і Mistral, забезпечують повний контроль над даними, але вимагають значних обчислювальних ресурсів та технічних знань для впровадження, що ускладнює їх використання в освітніх проектах.

Окрім цього, було проведено огляд технологій для реалізації клієнтської та серверної частини вебдодатку. Для клієнтської частини обрано React.js, TypeScript, Tailwind CSS та Visual Studio Code, які забезпечують продуктивність, адаптивність та зручність розробки. React.js дозволяє створювати багаторазово використовувані компоненти, TypeScript забезпечує статичну типізацію для зменшення помилок, а Tailwind CSS прискорює створення сучасного дизайну. Visual Studio Code, як середовище розробки, забезпечує гнучкість, інтеграцію з системами контролю версій та потужні інструменти для дебагінгу, що значно спрощує процес розробки. Серверна частина побудована на платформі Firebase, яка включає сервіси для управління

даними, аутентифікації користувачів, хостингу та виконання серверного коду через Firebase Functions. Інтеграція Firebase Functions з OpenAI API дозволяє безпечно передавати запити, приховувати ключі доступу та контролювати швидкість запитів, що забезпечує стабільну роботу системи навіть за високого навантаження.

Таким чином, обрані технології та інструменти забезпечують створення ефективної, масштабованої та безпечної технології для генерації навчальних тестів, яка відповідає сучасним освітнім стандартам та потребам користувачів.

3 РОЗРОБЛЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ТЕХНОЛОГІЇ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ НАВЧАЛЬНИХ ТЕСТІВ

3.1 Розроблення набору інструкцій для великої мовної моделі для генерування навчальних тестів

При роботі з великими мовними моделями важливу роль відіграє правильне формування інструкцій або промптів (“prompts”) для забезпечення точності, релевантності та структурованості відповідей. Основною ціллю формування докладних промптів є адаптувація моделі до специфіки задачі, враховуючи контекст, стиль та формат результату. У випадку генерації навчальних тестів це дозволяє створювати питання, які відповідають освітнім стандартам, враховують дисциплінарні особливості та рівень складності. Завдяки правильно сформульованим промптам можна досягти високої якості відповідей, що є критично важливим для освітніх проектів.

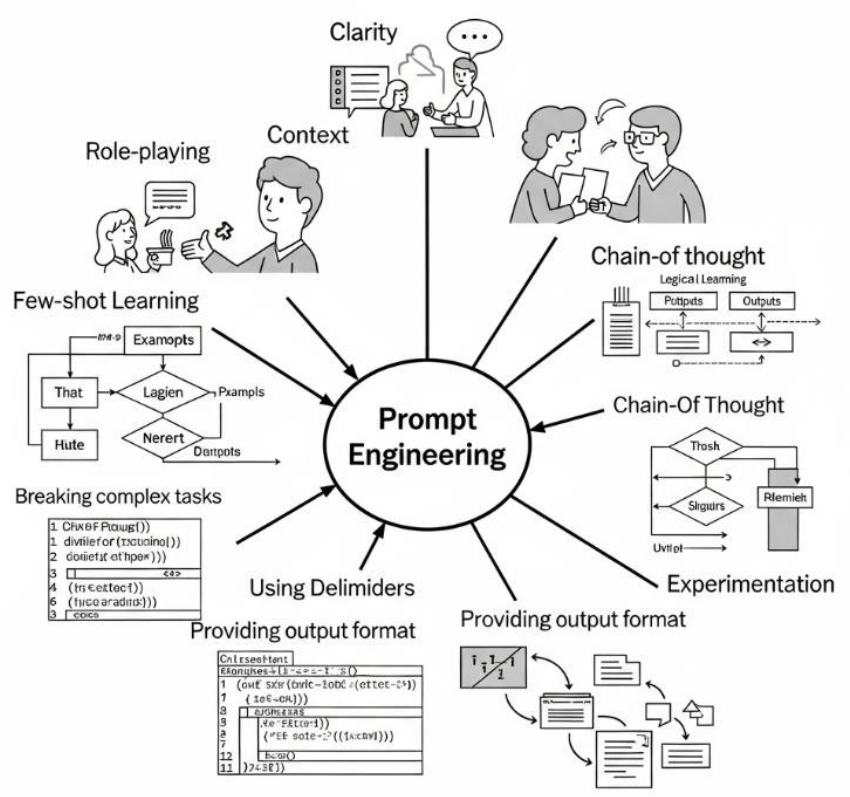

Рисунок 3.1 – Принципи Промпт-Інженерінгу

Prompt engineering [24] — це набір підходів для формування ефективних запитів для великих мовних моделей, який дозволяє отримувати точні та релевантні відповіді (рис 3.1). Він включає формулювання запитів, які враховують контекст задачі, стиль відповіді та очікуваний формат результату. Цей підхід дозволяє моделі адаптуватися до специфіки задачі, забезпечуючи високу якість та структурованість відповідей.

Важливість точності формування запитів полягає в забезпеченні ефективної інтеграції великої мовної моделі у технологію генерування тестів навчальних дисциплін. Для цього необхідно розробити спеціалізовану збірку промптів та інструкцій, які гарантують якісну генерацію тестових завдань. Ключовим аспектом успішної взаємодії з LLM є створення запитів, що включають контекстні інструкції про специфіку навчальної дисципліни, обмеження та чіткі вимоги до формату відповіді.

Адаптація промптів до задачі проходить завдяки ефективно сформульованій задачі, яка є ключовим елементом для отримання якісних відповідей. Такий підхід дозволяє моделі враховувати всі важливі аспекти задачі. Наприклад, для гуманітарних дисциплін необхідно враховувати контекст текстів, а для технічних — забезпечити точність використання термінології. Завдяки адаптації промптів до специфічних потреб задачі можна досягти результатів, які повністю відповідають вимогам користувачів та освітнім стандартам.

Сучасна промпт-інженерія передбачає використання кількох перевірених стратегій для ефективної взаємодії з великими мовними моделями. Однією з базових є zero-shot prompting — підхід, за якого модель отримує чітко сформульоване завдання без демонстраційних прикладів, орієнтуючись виключно на інструкції. Стратегія few-shot prompting доповнює цей підхід, надаючи моделі приклади очікуваного результату, що допомагає точніше відтворити бажаний формат і стиль відповіді. Chain-of-thought prompting, у свою чергу, стимулює модель до поетапного пояснення логіки міркувань, що особливо важливо для складних освітніх завдань.

В інтелектуальній технології генерування тестів навчальних дисциплін реалізовано гібридний підхід, який поєднує елементи всіх зазначених стратегій. Основна структура промптів побудована на zero-shot принципах із докладними інструкціями щодо формату та вимог до тестових завдань (рис 3.2).

```
const baseInstructions = `
На основі наданого тексту створи різноманітні запитання для тестування знань.

Формат відповіді:
Питання: [текст питання]
Варіанти:
A) [варіант відповіді]
B) [варіант відповіді]
C) [варіант відповіді]
D) [варіант відповіді]
Правильна відповідь: [повний текст правильної відповіді]

АБО для відкритих питань:
Питання: [текст питання з пропуском]
Правильна відповідь: [точна відповідь з тексту]
`;
```

Рисунок 3.2 – Zero-Shot елементи в промпті

Водночас, застосовуються елементи few-shot (рис 3.3), створюючи чіткі рамки для генерації відповідей.

```
const generalRequirements = `
Вимоги до генерації питань:
1. БАЗОВІ ПРИНЦИПИ:
   - Використовуй ТІЛЬКИ інформацію з наданого тексту
   - Кожне питання має перевіряти конкретне знання з тексту
   - Уникай неоднозначних формулювань
   - Правильна відповідь має бути очевидною при знанні матеріалу

2. СТРУКТУРА ПИТАНЬ:
   - Для питань з варіантами: рівно 4 варіанти (A, B, C, D)
   - Для відкритих питань: конкретна однозначна відповідь
   - Правильна відповідь ОБОВ'ЯЗКОВО має точно співпадати з одним з варіантів A-D

3. ТИПИ ПИТАНЬ ДЛЯ ВАРІАНТІВ:
   - Заповнення пропусків у реченнях з тексту
   - Вибір правильного визначення або пояснення
   - Встановлення причинно-наслідкових зв'язків
   - Визначення хронології подій
   - Питання на розуміння контексту

4. РОЗПОДІЛ ПО ТИПАХ:
   - 50% питань з відкритою відповіддю (заповнення пропусків)
   - 50% питань з варіантами відповідей (множинний вибір)
```

Рисунок 3.3 – Few-shot елементи в промпті

Для дисциплін підвищеної складності впроваджено елементи chain-of-thought (рис 3.4), які надають покрокові інструкції аналізу матеріалу та формування питань різного рівня.

```
8. АДАПТАЦІЯ ДО КОНТЕНТУ:
   - Для історичних текстів: фокус на датах, подіях, особах, причинах і наслідках
   - Для технічних текстів: термінологія, процеси, взаємозв'язки
   - Для літературних текстів: сюжет, персонажі, художні засоби
   - Для наукових текстів: факти, теорії, експерименти

9. РІВЕНЬ СКЛАДНОСТІ:
   - Питання мають відповідати рівню складності тексту
   - Баланс між простими фактологічними та складнішими аналітичними питаннями
   - Уникай занадто очевидних або занадто складних питань
```

Рисунок 3.4 – Chain-of-thought елементи в промпті

Ключовою особливістю розробленої інтелектуальної технології є стратегія Enhanced Contextual Prompting (рис 3.5).

```
function generatePromptForQuestions(
  text: string,
  count: number,
  testType: TestType = TestType.SELF_LEARNING,
  subject: string = "general_knowledge"
): string {
  const multipleChoiceCount = Math.ceil(count / 2);
  const openQuestionsCount = Math.floor(count / 2);

  return `
${baseInstructions}

${generalRequirements}

${testTypeInstructions[testType] || ""}

${subjectInstructions[subject] || ""}

КІЛЬКІСТЬ ПИТАНЬ:
- Загальна кількість: ${count}
- Питань з варіантами відповідей: ${multipleChoiceCount}
- Відкритих питань: ${openQuestionsCount}

Текст для аналізу: ${text}
  `.trim();
}
```

Рисунок 3.5 – Стратегія Enhanced Contextual Prompting

Вона забезпечує динамічну адаптацію промптів залежно від типу тесту (шкільний, університетський, професійний) та предметної області (наприклад, історія, математика, програмування). Такий підхід дозволяє генерувати педагогічно обґрунтовані та релевантні тестові завдання без необхідності створення окремих моделей для кожної дисципліни, забезпечуючи гнучкість, масштабованість і високу якість результатів при оптимальних обчислювальних витратах.

System prompt — це спеціальна інструкція, яка задається великій мовній моделі на початку взаємодії та визначає її роль, стиль спілкування й основні принципи роботи під час виконання завдання (рис 3.6). Основна мета system prompt полягає у формуванні базового контексту: наприклад, можна вказати, що модель повинна діяти як експерт певної галузі, дотримуватися академічного стилю чи відповідати виключно українською мовою.

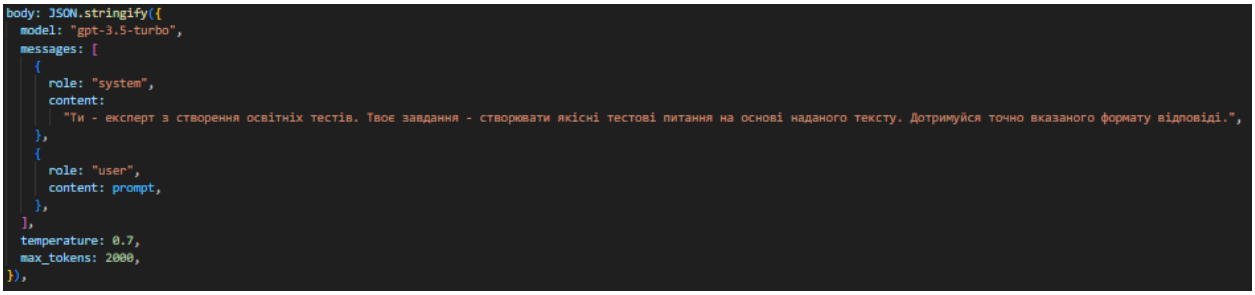

Рисунок 3.6 – Задання ролі LLM

В інтелектуальній технології генерування тестів system prompt використовується для того, щоб модель одразу орієнтувалася на створення педагогічно коректних, структурованих і релевантних тестових завдань. У даній реалізації system prompt задає моделі роль експерта зі створення освітніх тестів, акцентує на дотриманні чіткого формату відповідей та врахуванні освітніх стандартів. Це дозволяє забезпечити послідовність, якість і відповідність результатів незалежно від подальших користувацьких інструкцій.

Формування промпта для інтелектуальної технології генерування тестів здійснювалось поетапно з урахуванням сучасних підходів prompt engineering.

Основу складає zero-shot підхід: модель отримує чітко структуроване завдання з деталізованими інструкціями щодо формату, типів питань, розподілу складності та мовних вимог. Для підвищення якості впроваджено багаторівневу few-shot , а для складних дисциплін — елементи chain-of-thought із покроковими інструкціями.

Ключова інновація — Enhanced Contextual Prompting: промпт динамічно адаптується до типу тесту та предметної області, що дозволяє генерувати релевантні та педагогічно обґрунтовані завдання без створення окремих моделей. Окремо використовується system prompt, який задає моделі роль експерта, визначає стиль і акцентує на дотриманні освітніх стандартів, забезпечуючи якість і послідовність результатів.

Під час впровадження великих мовних моделей у інтелектуальну технологію особливо важливо формувати структуровану відповідь. Чітко визначений формат (наприклад, через типізовані інтерфейси) дозволяє легко інтегрувати результати генерації у функціонал додатку — для відображення, перевірки чи подальшої обробки.

Структурованість відповіді гарантує однозначність кожного елемента (питання, варіанти, правильна відповідь), що спрощує автоматизацію перевірки та взаємодію між фронтендом і бекендом. Такий підхід підвищує надійність системи й забезпечує якість освітнього контенту у вебсередовищі.

Отже, застосування сучасних підходів prompt engineering дало змогу сформувати оптимальний набір інструкцій у вигляді системного промпту, що повністю відповідає завданням інтелектуальної технології. Основу цього підходу складає zero-shot prompting, який доповнюється структурованими вимогами, а також елементами few-shot (чіткі приклади очікуваного результату) та chain-of-thought (покрокові інструкції для складних завдань). Додатково використовується стратегія contextual prompting, що забезпечує адаптацію промпта до типу тесту й предметної області. Завдяки такій

комбінації вдалося досягти стабільної генерації педагогічно обґрунтованих тестових завдань високої якості, які легко інтегруються у вебдодаток і відповідають сучасним освітнім стандартам.

3.2 Розроблення клієнтської частини інтелектуальної технології

Для початку розробки інтелектуальної технології було обрано сучасне та зручне середовище, яке забезпечує ефективність і комфорт роботи. Основним інструментом для написання коду став Visual Studio Code — текстовий редактор із широким набором розширень для фронтенд-розробки. Для організації швидкого старту проєкту, зручного керування залежностями та оптимізації процесу розробки було налаштовано Vite. В якості основного стеку технологій, згідно з вибором у огляді технологій для клієнтської частини. Використали React.js, Vite, TypeScript і Tailwind CSS. Такий підхід дозволив створити гнучке, масштабоване та легко підтримуване середовище для реалізації всіх функціональних вимог.

Для комфортної роботи у Visual Studio Code було налаштовано середовище під створення клієнтської частини інтелектуальної технології. Для цього застосовано Vite як сучасний фреймворк, який забезпечує миттєве оновлення змін під час розробки з використанням React та TypeScript. За допомогою React реалізовано компонентну архітектуру (рис 3.7), що дозволяє створювати багаторазово використовувані елементи інтерфейсу, ефективно управляти станом додатку та оптимізувати процес оновлення інтерфейсу. TypeScript використано для статичної типізації, що дало змогу визначати типи даних, підвищити надійність коду та спростити його підтримку. Tailwind CSS значно спростив процес стилізації інтерфейсу, дозволивши швидко створювати адаптивний і сучасний дизайн без необхідності писати велику кількість кастомних CSS-правил. Така конфігурація середовища забезпечила високу продуктивність розробки, зручність масштабування та підтримки проекту.

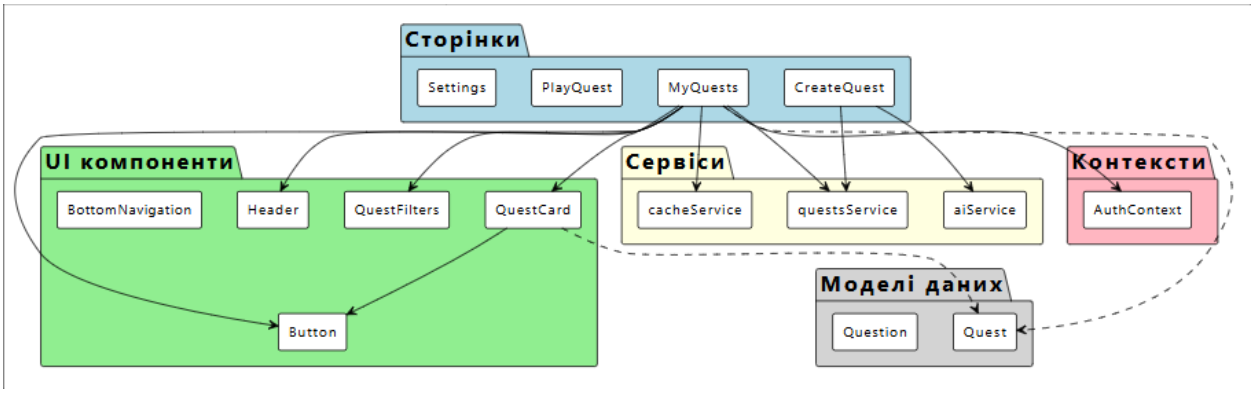

Рисунок 3.7 – Структура компонентів

За допомогою компонентів React був побудований інтерфейс користувача . Кожен елемент інтерфейсу це окремий компонент, що відповідає за свою частину функціоналу: від відображення тестових завдань до керування навігацією чи завантаженням файлів. Така архітектура дозволяє легко масштабувати додаток, повторно використовувати компоненти та підтримувати чистоту коду.

Основу інтерфейсу складають компоненти:

– AITestGenerator — головний компонент для створення та генерації тестів;

– QuestCard — відображення окремого тестового завдання;

– Header — верхня панель із заголовком та навігаційними елементами;

– BottomNavigation — нижня панель навігації для перемикання між основними розділами;

– QuestionForm — форма для створення або редагування тестових питань.

Tailwind CSS був використаний для забезпечення швидкої, гнучкої та сучасної стилізації інтерфейсу інтелектуальної технології. Цей фреймворк значно прискорив розробку та підтримку зовнішнього вигляду технології. Наприклад, реалізація підтримки темної та світлої теми дозволила інтерфейсу легко адаптуватися до налаштувань користувача, що підвищує зручність і сучасність технології. Завдяки Tailwind CSS також вдалося легко впровадити

анімації для інтерактивних елементів, наприклад, плавне відображення сповіщень чи підсвічування активних кнопок. Окрім цього, використання вбудованих утиліт для роботи з кольоровими палітрами та тінями дозволило швидко створити візуально привабливий і цілісний дизайн, який відповідає сучасним вимогам до користувацького досвіду.

Бізнес-логіка (рис 3.8) інтелектуальної технології побудована на взаємодії з хмарними сервісами та зовнішніми API, що забезпечує надійність і масштабованість системи. Для зберігання даних, аутентифікації користувачів та організації доступу до тестових завдань використовується платформа Firebase. Взаємодія з Firebase реалізована через спеціалізовані сервіси, які відповідають за збереження результатів проходження тестів, управління профілями користувачів та синхронізацію даних у реальному часі. Це дозволяє надати безпечний доступ до інформації, швидке оновлення даних та зручну роботу з різних пристроїв.

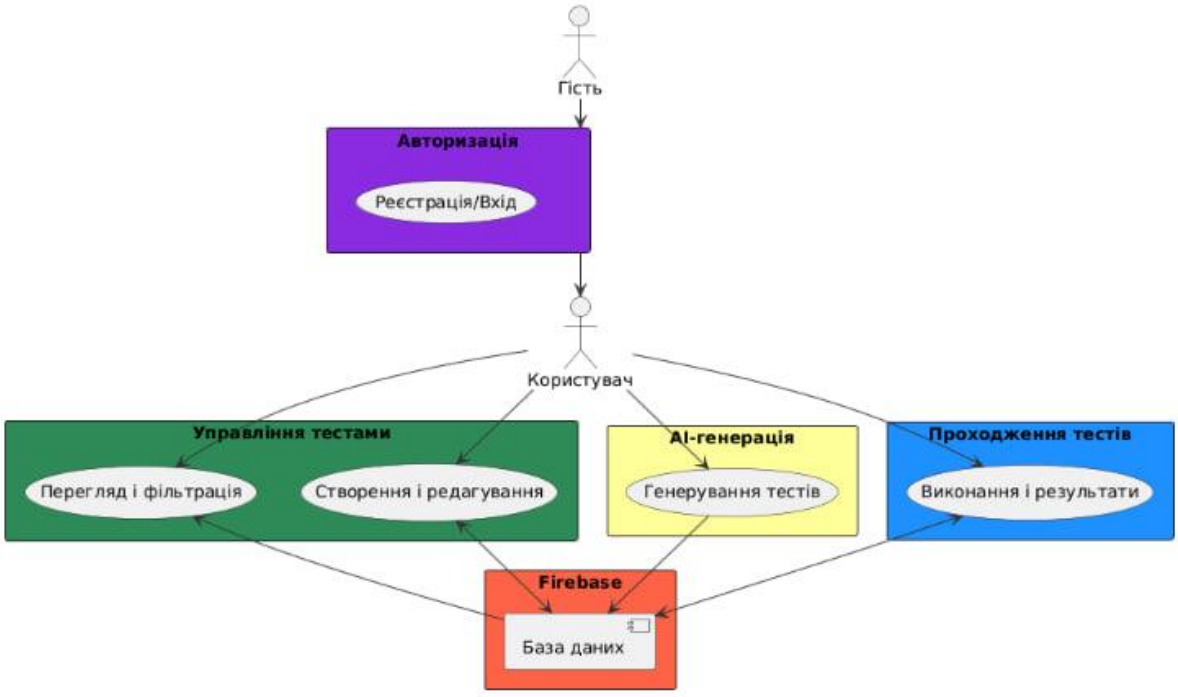

Рисунок 3.8 – Діаграма модулів, які реалізовують бізнес-логіку

Окрему роль у бізнес-логіці відіграє інтеграція з великою мовною моделлю (LLM) через OpenAI API. Результати роботи LLM використовуються

для автоматичної генерації тестових завдань на основі навчальних матеріалів. Для цього реалізовано окремий сервіс, який формує запит до моделі, обробляє отриману відповідь, структурує її відповідно до вимог системи та зберігає у базі даних. Такий підхід дозволяє автоматизувати процес створення якісних тестів, адаптованих до різних дисциплін і рівнів складності.

Завдяки чіткому розподілу відповідальності між компонентами бізнес-логіки, технологія залишається гнучкою та легко масштабованою. Всі основні процеси — від генерації тестів до збереження результатів і управління користувачами — реалізовані у вигляді окремих модулів, що спрощує підтримку, тестування та подальший розвиток інтелектуальної технології.

Серед тих, які використовувалися для побудови інтерфейсу, можна перечислити усі компоненти :

- AITestGenerator — головний компонент для створення та генерації тестів;
- QuestCard — відображення окремого тестового завдання;
- Header — верхня панель із заголовком та навігаційними елементами;
- BottomNavigation — нижня панель навігації для перемикання між основними розділами;
- FileUploader — завантаження файлів користувачем;
- QuestionForm — форма для створення або редагування тестових питань;
- QuestFilters — фільтрація тестів за різними параметрами;
- QuestHistoryList — відображення історії проходження тестів;
- ReferenceManager — керування довідковою інформацією;
- SubjectManager — керування предметами;
- TestTypeManager — керування типами тестів;
- AuthRequired — захист сторінок, які потребують авторизації.

Такий підхід дозволяє створити гнучкий, сучасний і зручний для користувача інтерфейс, який легко підтримується та розширюється.

За бізнес-логіку у проєкті відповідали такі основні компоненти та модулі:

- aiService.ts — відповідає за генерацію тестових завдань із використанням великої мовної моделі (LLM) та обробку результатів генерації;
- firebase.ts — забезпечує взаємодію з Firebase для зберігання даних, аутентифікації користувачів та синхронізації інформації;
- questsService.ts — реалізує логіку керування тестовими завданнями, включаючи створення, редагування, видалення та отримання списків тестів;
- AuthContext — контекст для управління станом аутентифікації користувача у додатку;
- SettingsContext — контекст для зберігання та зміни налаштувань користувача.

Ці компоненти забезпечують основні бізнес-процеси: генерацію, збереження, обробку та управління тестами, а також роботу з користувачами та їхніми налаштуваннями.

3.3 Розроблення серверної частини інтелектуальної технології

Серверна частина інтелектуальної технології побудована на основі хмарної платформи Firebase, що дозволяє забезпечити надійне зберігання даних. Для організації бази даних використанно Firebase Firestore яке зберігає інформацію про тести, користувачів, результати проходження та інші сутності у структурованому вигляді. У Firestore дані синхронізуються у реальному часі, що надає актуальність інформації для всіх користувачів незалежно від пристрою.

Для розгортання та забезпечення доступу до вебдодатку було використано Firebase Hosting. Цей сервіс дозволяє швидко публікувати вебресурси та гарантує їхню постійну доступність для користувачів. Firebase Hosting забезпечує надійне та захищене з’єднання через автоматичне використання протоколу HTTPS, що підвищує безпеку передачі даних. Окрім

цього, завдяки використанню глобальної мережі доставки контенту (CDN), сторінки завантажуються швидко незалежно від місцезнаходження користувача, що позитивно впливає на зручність і якість роботи з додатком.

Аутентифікація користувачів у інтелектуальній технології реалізована за допомогою сервісу Firebase Authentication. Для доступу до функціоналу розробленно вхід за логіном та паролем, що дозволяє ідентифікувати кожного користувача та забезпечити захист персональних даних (рис 3.9). Реалізовано можливість реєстрації нового облікового запису, авторизації та управління профілем. Завдяки інтеграції з Firebase Authentication обмежується доступ до тестів для неавторизованих користувачів, а також забезпечується персоналізація інтерфейсу та збереження індивідуальних налаштувань.

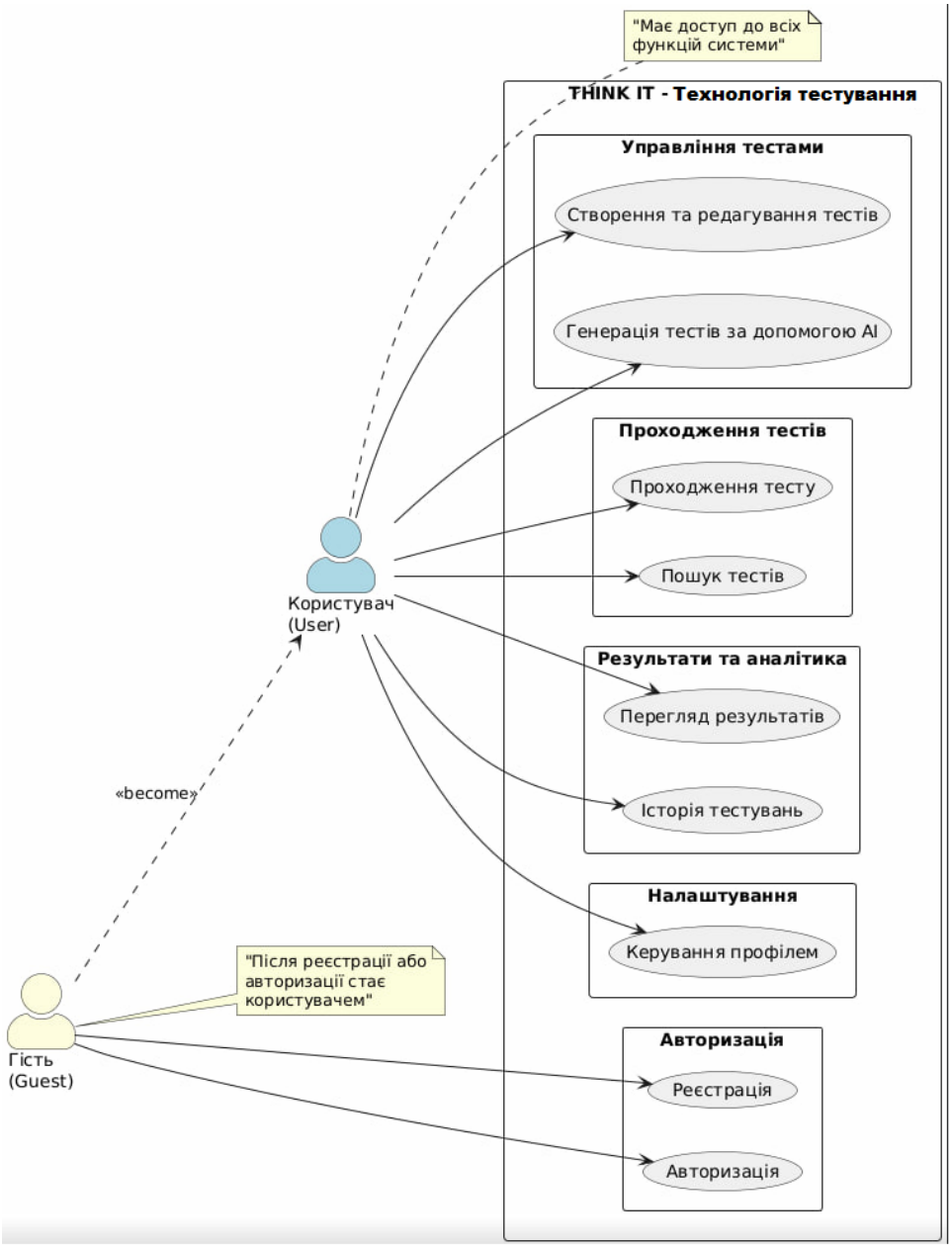

Рисунок 3.9 – Use case діаграма інтелектуальної технології

Реалізацію механізмів кешування у вебдодатку виконує сервіс Firebase Firestore. Він дозволяє зберігати основні дані (тести, результати, профілі користувачів) у структурованому вигляді та забезпечує їхню синхронізацію у реальному часі між клієнтом і сервером (рис 3.10).

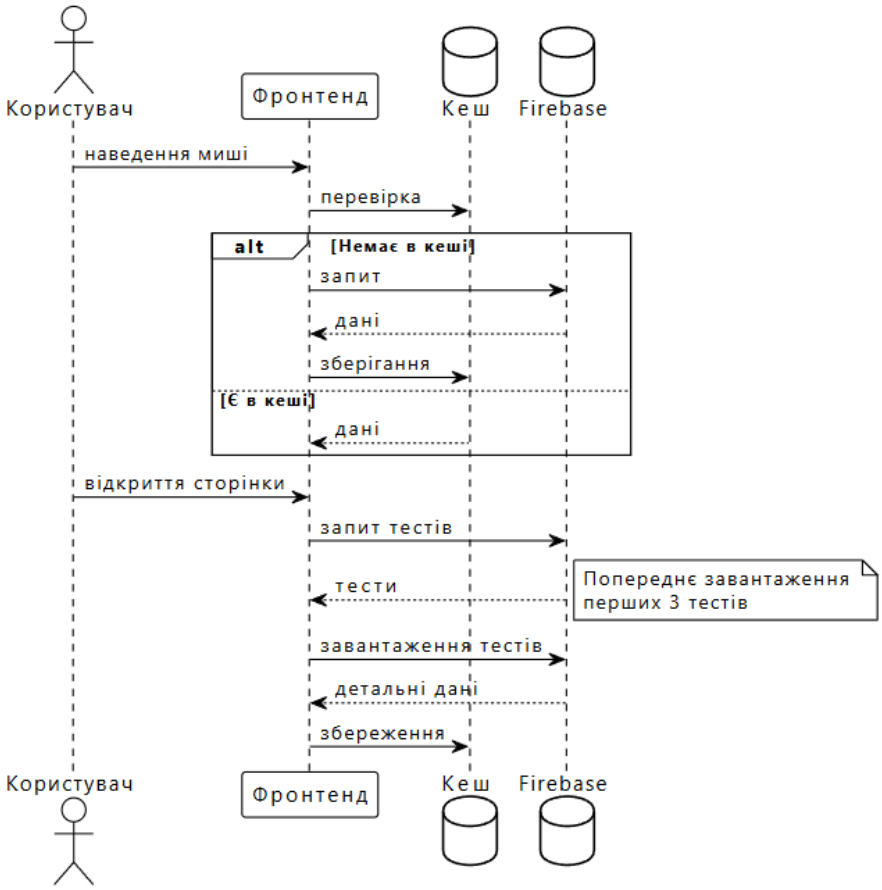

Рисунок 3.10 – UML-діаграма послідовності механізму кешування у вебдодатку

Окрім цього, Firestore підтримує локальне кешування даних на пристрої користувача: навіть при відсутності інтернет-з’єднання додаток продовжує працювати з локальною копією даних, а після відновлення мережі всі зміни автоматично синхронізуються з хмарною базою. Такий підхід гарантує швидкий доступ до інформації, підвищує стабільність роботи додатку та забезпечує комфортну взаємодію незалежно від якості з’єднання.

Однією з ключових складових серверної частини є організація взаємодії з великою мовною моделлю для автоматизованої генерації тестових завдань. Взаємодія з великою мовною моделлю (LLM) реалізована за допомогою Firebase Functions (рис 3.11).

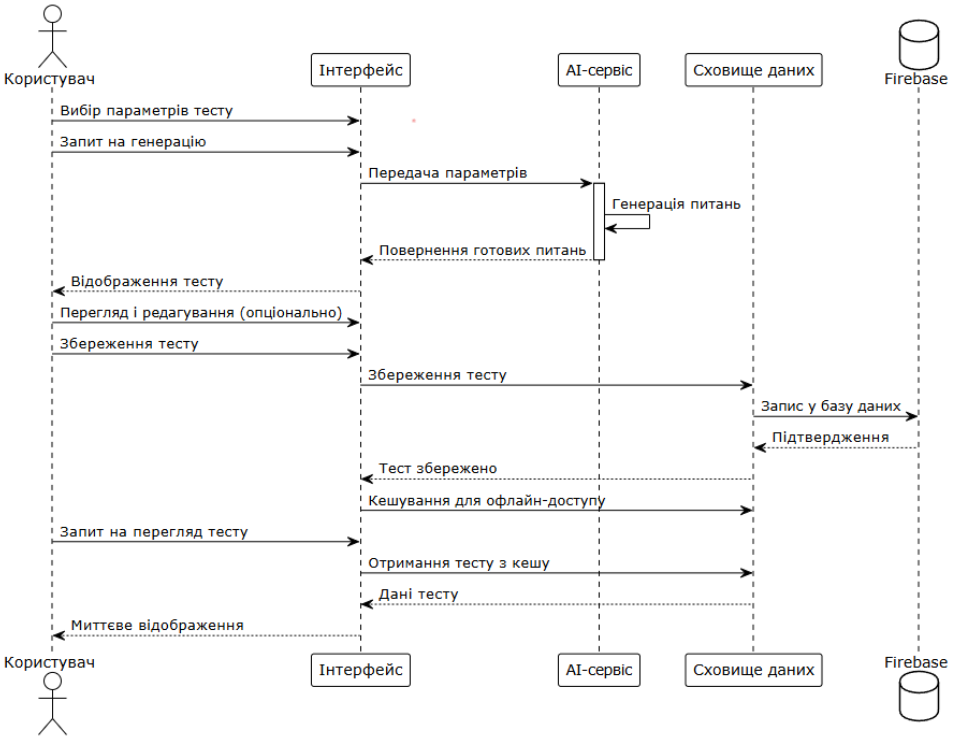

Рисунок 3.11 – UML-діаграма послідовності механізму генерації тестів та після отримання його для проходження

Серверні функції отримують запити від клієнтської частини, формують відповідні промпти для LLM, надсилають їх до OpenAI API, а після отримання результату обробляють відповідь і зберігають її у Firestore. Це дає можливість централізовано керувати процесом генерації тестів, гарантує безпечну передачу даних і приховує API-ключі від клієнтської частини, що підвищує загальний рівень безпеки технології.

Використання сервісів Firebase дозволило повністю реалізувати як функціональні, так і нефункціональні вимоги інтелектуальної технології. Усі необхідні можливості були впроваджені наступним чином:

– Управління тестами (створення, редагування, видалення), підтримка різних форматів завдань, категоризація за предметами та типами, імпорт матеріалів — реалізовано за допомогою Firebase Firestore.
– Інтерактивне проходження тестів, автоматична перевірка відповідей, перегляд правильних відповідей, повторне проходження, збереження

історії, детальна статистика та аналіз результатів — забезпечено через Firebase Firestore.

– Реєстрація, авторизація користувачів, налаштування профілю, персоналізація, управління доступом до тестів — реалізовано за допомогою Firebase Authentication.
– Пошук, фільтрація, сортування тестів, швидкий доступ до нещодавно використаних — впроваджено через Firebase Firestore.
– Нефункціональні вимоги:
– Висока продуктивність (швидкий час відгуку, ефективне кешування, попереднє завантаження тестів) — досягнуто завдяки Firebase Firestore з підтримкою локального кешування.
– Доступність (офлайн-режим, адаптивний дизайн, підтримка різних пристроїв і браузерів, мінімальні вимоги до апаратних ресурсів) — забезпечено через Firebase Firestore (офлайн-синхронізація) та Firebase Hosting.
– Надійність (захист від втрати даних, механізми відновлення після помилок, регулярне резервне копіювання) — реалізовано за допомогою Firebase Firestore.
– Масштабованість (робота з великими обсягами тестів і користувачів, модульна структура, гнучкість налаштування) — досягнуто завдяки архітектурі Firebase.
– Безпека (захист персональних даних, шифрування, обмеження доступу до конфіденційної інформації) — забезпечено через Firebase Authentication, правила доступу Firebase Firestore та використання HTTPS у Firebase Hosting.
- Зручність використання (інтуїтивний інтерфейс, логічна організація функцій, мінімальна кількість дій для виконання основних завдань) — досягнуто завдяки комплексному використанню сервісів Firebase у поєднанні з сучасними підходами до розробки.

Завдяки такій інтеграції всі вимоги до серверної частини були виконані в повному обсязі, що дозволило створити надійну, безпечну та сучасну інтелектуальну технологію для автоматизованого створення навчальних тестів.

Для взаємодії з Firebase у проєкті створено низку спеціалізованих модулів[25]. Основний модуль firebase.ts відповідає за ініціалізацію та конфігурацію сервісів Firebase, зокрема Firestore, Authentication, Hosting і Functions. Через нього здійснюється підключення до бази даних, налаштування безпеки та доступ до серверних функцій.

Модуль questsService.ts реалізує бізнес-логіку для роботи з тестами: створення, редагування, видалення, отримання списків і збереження результатів. Для управління користувачами використовується контекст AuthContext.ts, який взаємодіє з Firebase Authentication для реєстрації, входу та зберігання інформації про користувача.

Серверні функції у Firebase Functions відповідають за обробку запитів до LLM, формування промптів, надсилання їх до OpenAI API та збереження результатів у Firestore. Така структура спрощує підтримку, тестування та розвиток інтелектуальної технології.

3.4 Проєктування IT-інфраструктури та архітектури технології

IT-інфраструктура — це комплекс апаратних і програмних ресурсів, мережевих сервісів та інструментів, які забезпечують стабільну роботу, зберігання, обробку й передачу даних у межах інтелектуальної технології. У даному програмному продукті ця інфраструктура виступає надійною основою для функціонування всіх сервісів і модулів, дозволяє організувати ефективну взаємодію між клієнтською та серверною частинами, а також гарантує масштабованість, безпеку й доступність додатку для користувачів на різних пристроях[26]. Саме завдяки продуманій IT-інфраструктурі система легко розгортається, підтримується та може розвиватися відповідно до сучасних вимог цифрових технологій.

Архітектура технології — це продумана схема організації всіх основних компонентів системи, їхніх взаємозв’язків і принципів взаємодії. У даній розробці архітектура визначає, як саме структуровані та взаємодіють між собою клієнтський інтерфейс, серверна логіка, база даних, модулі аутентифікації, бізнес-логіка та зовнішні сервіси, зокрема LLM. Такий підхід дозволяє забезпечити ефективність, надійність, безпеку й гнучкість технології, а також значно спрощує її підтримку та подальший розвиток (рис 3.12). Завдяки продуманій архітектурі система легко розширюється новими функціями й гарантує стабільну роботу для кінцевого користувача.

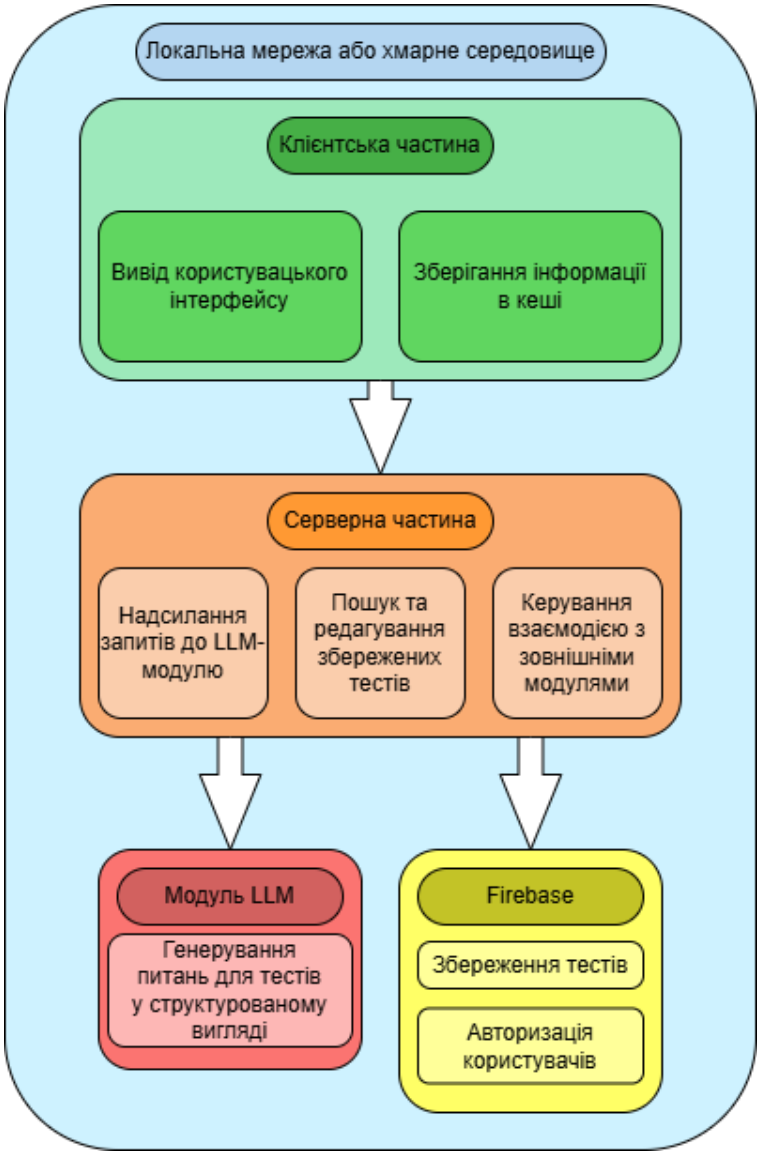

Рисунок 3.12 – Архітектура технології

Розгортання інтелектуальної технології можливе як на власному апаратному забезпеченні організації, так і у хмарному середовищі. Локально технологія запускається через встановлення всіх необхідних залежностей за допомогою менеджера пакетів (наприклад, npm або bun) та запуску розробницького сервера Vite, що дозволяє розробникам тестувати й удосконалювати функціонал на власному комп'ютері. Для хмарного розгортання використовується сервіс Firebase Hosting, який забезпечує швидке та безпечне публікування вебдодатку, а також автоматичне підключення до хмарної бази даних, аутентифікації та серверних функцій. Це дозволяє впроваджувати технолгогію у різних умовах, а саме у школах, університетах, компаніях чи інших установах залежно від потреб і технічних можливостей. Гнучкість розгортання забезпечує адаптацію технології до специфіки організації, незалежно від її масштабу чи інфраструктури.

- Архітектура технології складається з кількох основних модулів, кожен із яких виконує окрему роль:
- Клієнтський модуль (frontend) — реалізований на React.js з використанням Vite, TypeScript і Tailwind CSS. Відповідає за відображення інтерфейсу, взаємодію з користувачем, відправку запитів до серверної частини та обробку відповідей.
- Модуль аутентифікації — побудований на Firebase Authentication, забезпечує реєстрацію, вхід, вихід користувачів, а також захист персональних даних і управління доступом до функцій додатку.
- Модуль зберігання та обробки даних — реалізований на Firebase Firestore, відповідає за зберігання тестів, результатів, профілів користувачів, історії проходження та іншої інформації, а також за синхронізацію даних у реальному часі.
- Серверний модуль (backend) — реалізований через Firebase Functions, відповідає за виконання бізнес-логіки, взаємодію з великою мовною

моделлю (LLM), формування та обробку промптів, а також за збереження результатів генерації тестів.

– Модуль розгортання та доставки контенту — реалізований на Firebase Hosting, забезпечує публікацію, доступність і захищене з’єднання для всіх користувачів.

Така модульна структура дозволяє легко масштабувати технологію, розширювати функціонал і забезпечувати стабільну роботу як у локальному, так і в хмарному середовищі. Поєднання модулів, описаних вище, дають змогу взаємодіяти з інтелектуальною технологією та виконувати генерування навчальних тестів, алгоритм дій при створенні таких тестів наведений на блок-схемі нижче (рис 3.13).

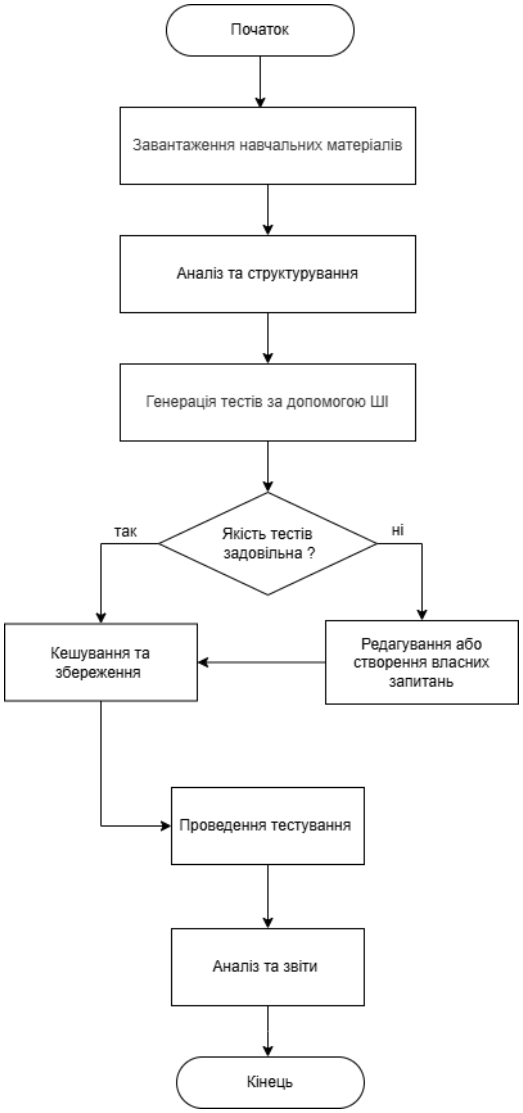

Рисунок 3.13 – Блок-схема алгоритму дій для генерування навчального тесту

Основні типи архітектури програмних технології включають MVC, MVP та MVVM. Підходи, які дозволяють розділити логіку додатку, інтерфейс користувача та обробку даних, що спрощує підтримку, тестування й масштабування програмних продуктів.

Архітектурні патерни MVC, MVP та MVVM [27] забезпечують структуровану організацію програмного коду, розділяючи відповідальність між окремими компонентами технології.

MVC (Model-View-Controller) — це підхід, за якого додаток поділяється на три незалежні частини: модель (відповідає за зберігання та обробку даних), представлення (відображає інформацію користувачу) та контролер (реалізує логіку взаємодії, обробляє дії користувача та координує обмін між моделлю і представленням). Така структура дозволяє чітко відокремити бізнес-логіку від інтерфейсу, що спрощує підтримку, тестування та подальший розвиток технології.

MVP (Model-View-Presenter) — це варіація MVC, у якій основна взаємодія відбувається між View (інтерфейсом) і Presenter (презентером). View відповідає лише за візуалізацію, а вся логіка обробки подій і взаємодії з моделлю зосереджена у Presenter. View не взаємодіє з моделлю напряму, а отримує дані виключно через Presenter. Такий підхід підвищує тестованість логіки та часто використовується у desktop- і mobile-додатках.

MVVM (Model-View-ViewModel) — це архітектурний патерн, що передбачає наявність проміжного шару ViewModel між моделлю та представленням. ViewModel містить логіку відображення, трансформує дані з моделі для View і реагує на дії користувача. Зв’язок між View і ViewModel зазвичай реалізується через двостороннє data binding, що забезпечує автоматичне оновлення інтерфейсу при зміні даних. MVVM активно застосовується у сучасних фреймворках для розробки складних UI, оскільки спрощує підтримку, тестування та повторне використання компонентів.

У THINK IT використовується архітектурна модель MVVM (Model-View-ViewModel).

Цей підхід забезпечує чітке розділення відповідальності між інтерфейсом користувача (View), логікою відображення та обробки даних (ViewModel) і самими даними (Model). Завдяки цьому технологія стає більш структурованою, гнучкою та простою у підтримці.

- Переваги MVVM:
- Полегшує масштабування та розвиток проекту, оскільки компоненти ізольовані та легко змінюються;
- Забезпечує автоматичне оновлення інтерфейсу при зміні даних через двосторонній зв’язок;
- Сприяє повторному використанню логіки та UI-компонентів;
- Підвищує тестованість бізнес-логіки незалежно від інтерфейсу;
- Особливо ефективна для сучасних вебдодатків, де важлива швидкість і гнучкість оновлення UI.

У сучасних інтелектуальних технологіях масштабування є важливою характеристикою, що дозволяє адаптувати технологію до зростання навантаження, кількості користувачів або обсягу даних. Масштабування може відбуватись у двох напрямках: вертикально шляхом збільшення потужності окремих компонентів, та горизонтально через додавання нових екземплярів сервісів для паралельної обробки запитів. Обидва підходи забезпечують гнучкість, стабільність і готовність системи до подальшого розвитку.

У реалізованій архітектурі додатку передбачені можливості як вертикального, так і горизонтального масштабування.

Вертикальне масштабування забезпечується завдяки використанню хмарних сервісів, зокрема Firebase, які дозволяють оперативно збільшувати обчислювальні ресурси для обробки складніших завдань або більших обсягів даних без зміни структури системи.

Горизонтальне масштабування реалізується через модульну побудову додатку: окремі сервіси, такі як генерація тестів на основі ШІ, робота з базою даних чи обробка користувацьких запитів, можуть бути розгорнуті у вигляді незалежних екземплярів.

Це дає змогу ефективно розподіляти навантаження між різними вузлами, забезпечуючи стабільну роботу системи навіть при значному зростанні кількості користувачів чи запитів.

Сучасні технології можуть будуватись за різними архітектурними підходами, серед яких найбільш поширеними є мікросервісна та монолітна архітектури. Мікросервіси передбачають поділ системи на окремі незалежні сервіси, кожен з яких виконує чітко визначену функцію та може бути різного типу: для обробки даних, автентифікації, генерації звітів чи інтеграції зі сторонніми API. Кожен мікросервіс розгортається, масштабується та оновлюється окремо, а взаємодія між ними відбувається через стандартизовані інтерфейси (наприклад, REST API або message broker). Моноліт, навпаки, — це єдина цілісна система, де всі компоненти тісно пов’язані між собою та розгортаються разом. Такий підхід простіший на початкових етапах, але з часом може ускладнювати масштабування та підтримку.

У даній технології реалізовано модульну монолітну архітектуру: основні функції (інтерфейс, бізнес-логіка, робота з даними, інтеграція з AI та Firebase) знаходяться в єдиній кодовій базі та розгортаються разом. Такий підхід спрощує розробку, тестування та підтримку на початкових етапах, а також дозволяє швидко вносити зміни у функціонал. Водночас структура системи передбачає можливість подальшого виділення окремих сервісів у самостійні модулі, що відкриває шлях до поступового переходу до мікросервісної архітектури у разі зростання проєкту чи збільшення навантаження.

Розгортання клієнтської частини на CDN (Content Delivery Network) у проєкті THINK IT реалізовано за допомогою Firebase Hosting, що забезпечує швидку та надійну доставку статичного контенту (HTML, CSS, JavaScript, зображення) користувачам через глобальну мережу серверів. Цей підхід

значно підвищує швидкодію додатку, оскільки вміст завантажується з найближчого до користувача сервера, мінімізуючи затримки. Firebase Hosting також автоматично забезпечує SSL-сертифікати для безпечного з'єднання та інтегрується з іншими сервісами Firebase, що спрощує розробку та підтримку. Завдяки такому рішенню вебдодаток THINK IT отримує високу доступність, автоматичне масштабування під навантаженням та оптимальну продуктивність для користувачів з різних регіонів.

3.5 Тестування функціональності та продуктивності вебдодатку

Тестування та продуктивність інтелектуальної технології THINK IT продемонстрували стабільне функціонування та відповідність заявленим вимогам. Процес перевірки включав тестування всіх ключових компонентів: головної сторінки, сторінки налаштувань з елементами реєстрації та персоналізації, а також сторінки створення тестів із можливістю вибору рівня складності, визначення назви та опису тесту. Особлива увага приділялася функціональній перевірці генерації тестових питань, зокрема вибору типу тесту, предмету та кількості питань. Система продемонструвала надійну роботу при завантаженні користувацьких текстів, їх редагуванні перед генерацією, а також відповідність згенерованого контенту заданим параметрам промпту, з підтримкою різної типізації питань (як з вибором відповіді, так і відкритого типу).

Додатково було протестована функціональність проходження тестів, яка показала відсутність критичних помилок та коректне відображення результатів (оцінки, відсоток правильних відповідей та кількість вірних відповідей) з аналітикою відповідей, збереженням часу проходження та фіксацією найкращого результату з усіх спроб. При перевірці редагування питань у тестах зберігали нову актуальну відповідь та виконували свої функції.

Перевірка підтвердила можливість повторного проходження тестів та ефективність системи фільтрації за всіма доступними параметрами , що значно покращує навігацію серед створених тестових матеріалів.

На головній сторінці показано основні можливості інтелектуальної технології. Сама сторінка без помилок та візуальних дефектів (рис 3.14).

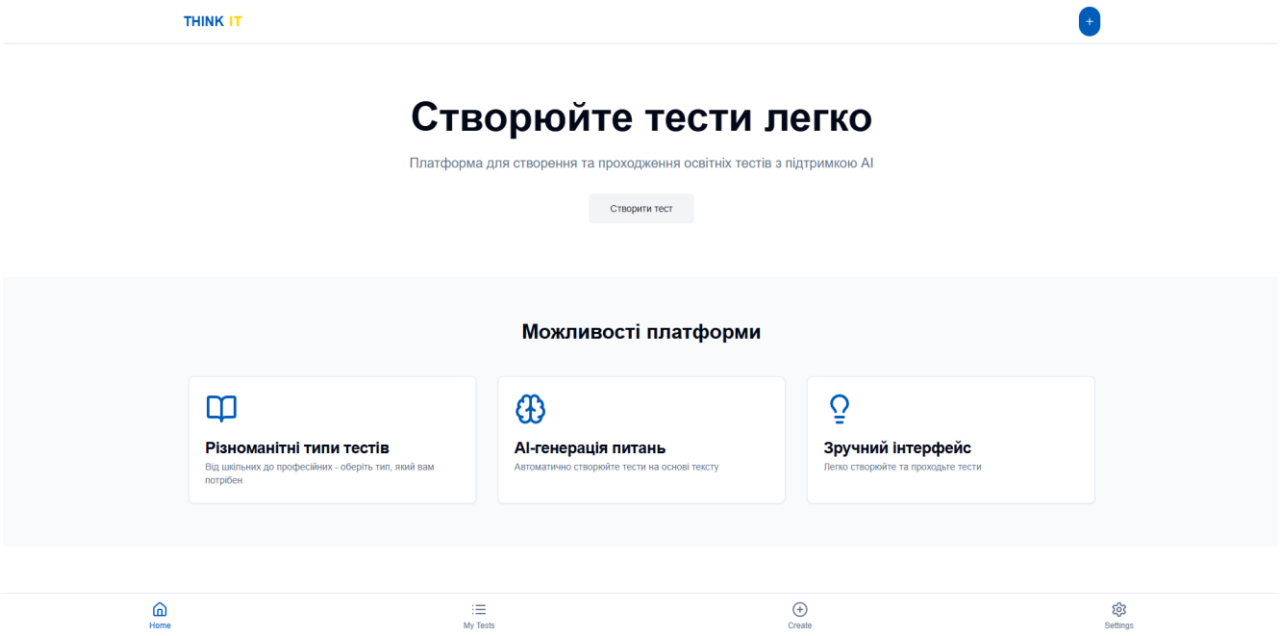

Рисунок 3.14 – Головна сторінка додатку

На рисунку показаний елемент навігації по наявних сторінках, при взаємодії з кожним окремим значком змінює наявну сторінку на представлені у панелі навігації (рис 3.15).

Рисунок 3.15 – Елемент навігації

Даний елемент дозволяє пройти перевірку на наявність вже існуючого профілю чи зареєструвати за допомогою логіну та паролю. На рисунку проведена перевірка полів для введення даних аби після перевірити можливість входу чи реєстрації (рис 3.16).

Рисунок 3.16 – Елемент входу в профіль

На рисунку наведено можливості персоналізації зміна теми додатку для зменшення навантаження на очі, розміру тексту для збільшення комфорту у різних вікових категорій, налаштування аудіо девайсів і регуляція гучності (рис 3.17).

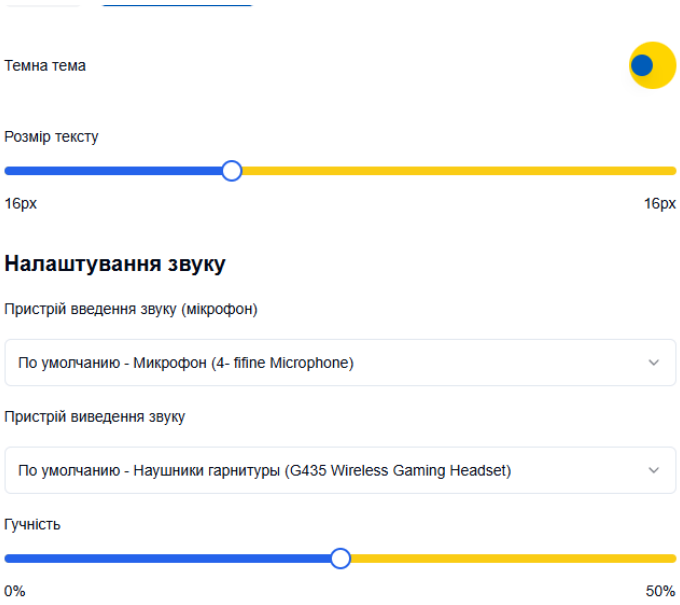

Рисунок 3.17 – Персоналізація платформи

Здійсненна зміна теми, розміру тексту, входу та компонентів вибору пристроїв введення та виведення звуку для достовірної перевірки на роботу цих компонентів (рис 3.18).

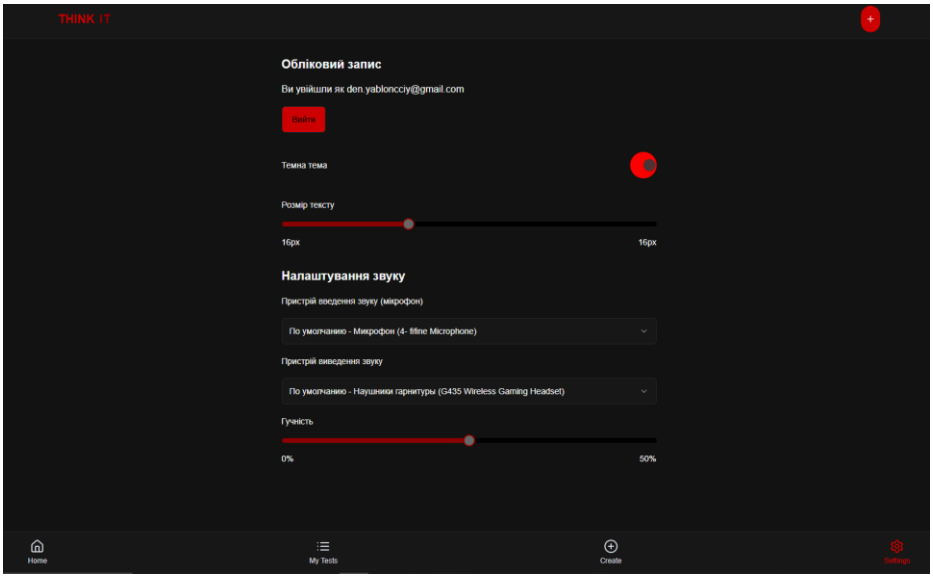

Рисунок 3.18 – Сторінка налаштувань після всіх змін

Протестовані поля для вводу теми та опису тесту. Вибору тесту та предметом для нього для подальшого пошуку його на сторінці з тестами за допомогою фільтрації (рис 3.19).

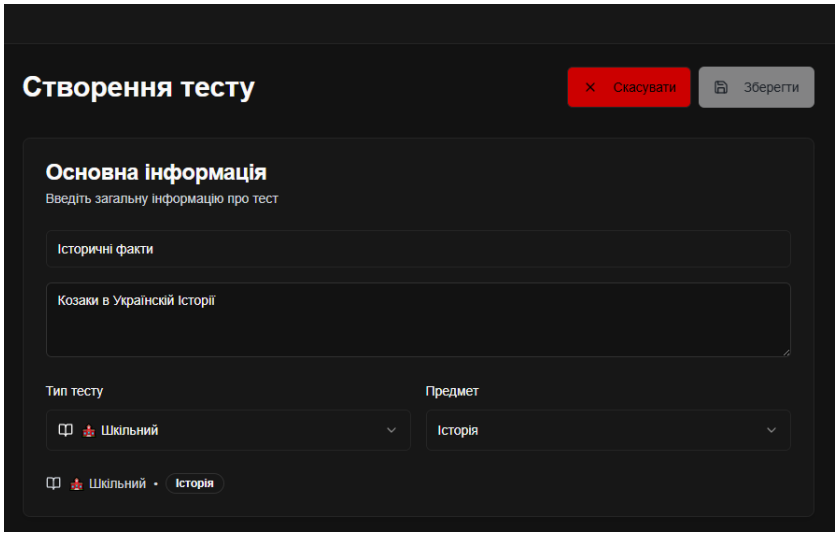

Рисунок 3.19 – Компонент на сторінці для створення тестів

Показані на рисунку можливості для створення питань а саме вибір типу створення вручну чи генерацією , після вибору генерації вибір для завантаження тексту у показаному форматі (.txt .docx) чи вставлення або написання тексту особисто. В нижній панелі наведено вибір між фіксованою кількістю питань типом тесту та предметом для більш кращої генерації (рис 3.20).

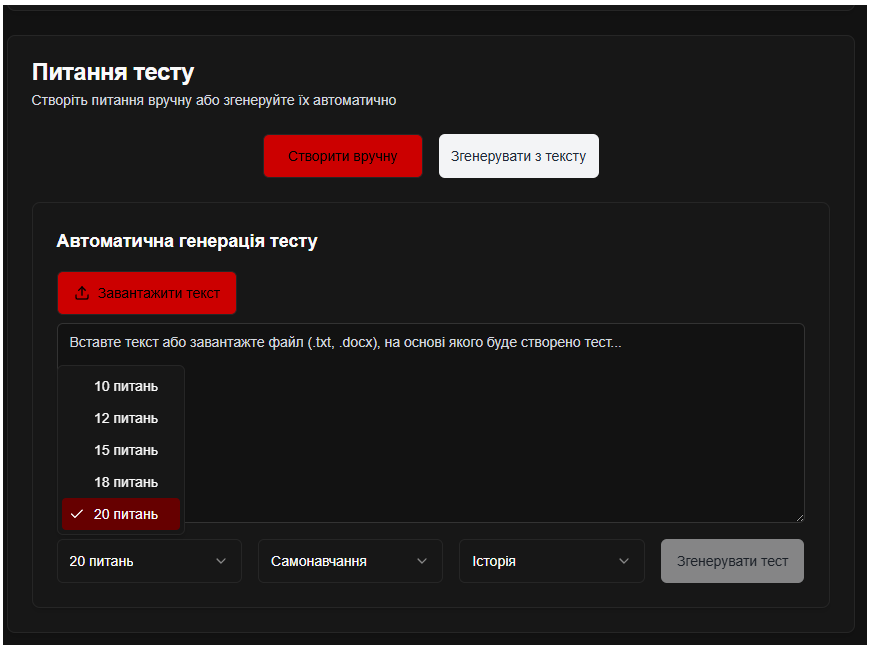

Рисунок 3.20 – Компонент для створення питань

Для перевірки роботи вводу чи завантаження тексту та редагування його перед генеруванням текст був вставлений обома способами (рис 3.21).

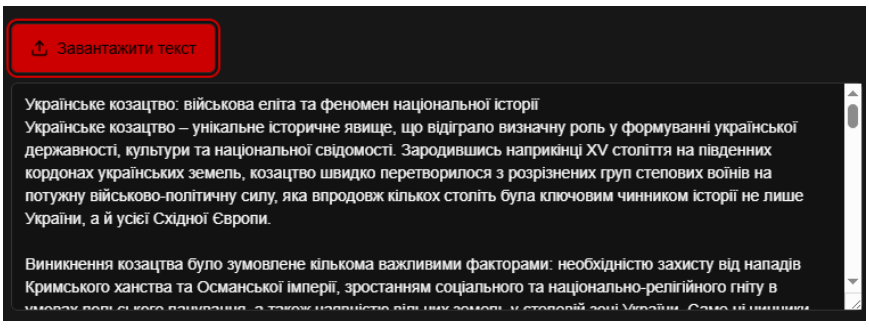

Рисунок 3.21 – Компонент з текстом

Після генерації тесту є можливість відредагувати запитання або змінити тип запитань на вибір з відкритою відповіддю та з варіантами. Також його це можна використати для змінення відповіді у запитанні на більш коректну (рис 3.22).

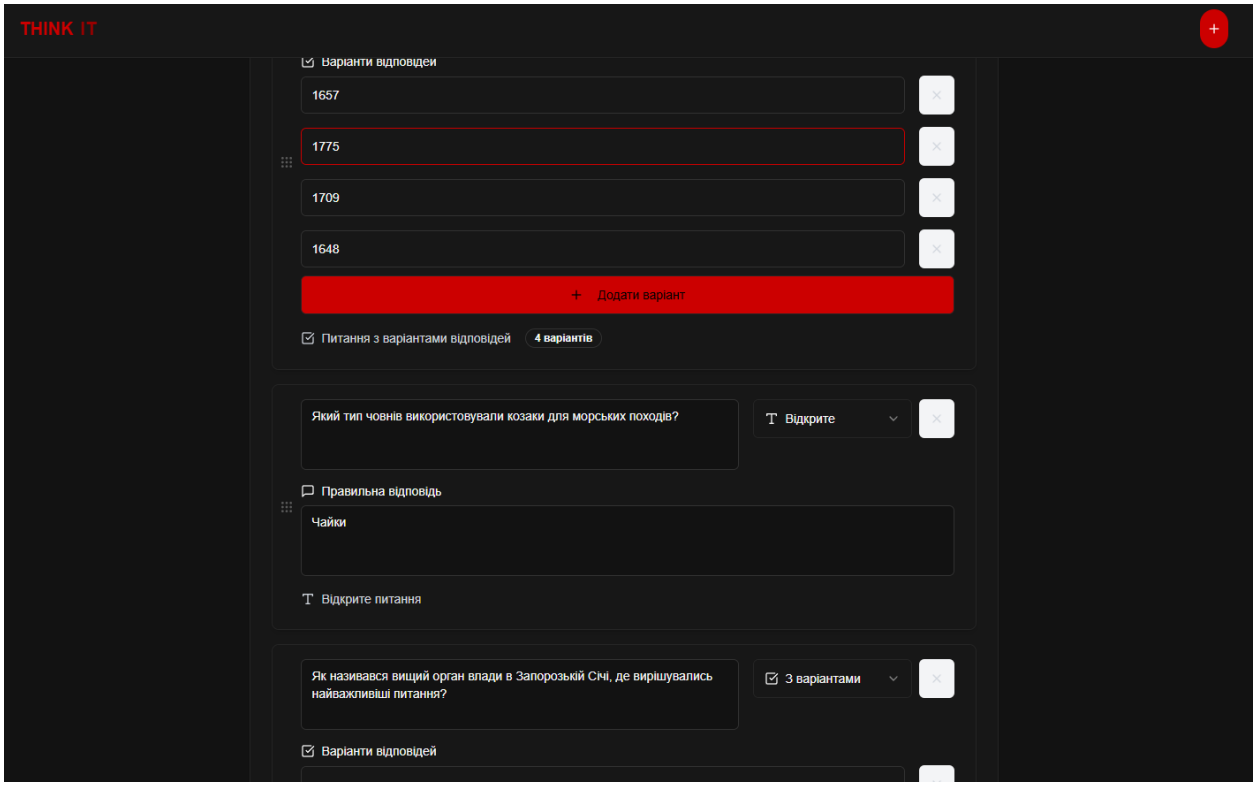

Рисунок 3.22 – Згенерований тест

Після генерації та можливого редагування тест зберегти або скасувати створення якщо потрібно створити тест за іншим текстом, з іншою кількістю питань чи іншим типом за предметом (рис 3.23).

Рисунок 3.23 – Компонент збереження

Після тестування компонента збереження , створений користувачем тест з’являється на сторінці з тестами. На ньому ми можемо переглянути раніше вказану інформацію таку як назву, опис, в правому кутку тип, предмет, кількість питань, дату створення. Також можливість редагування тесту до та після проходження, видалення тесту та проходження (рис 3.24).

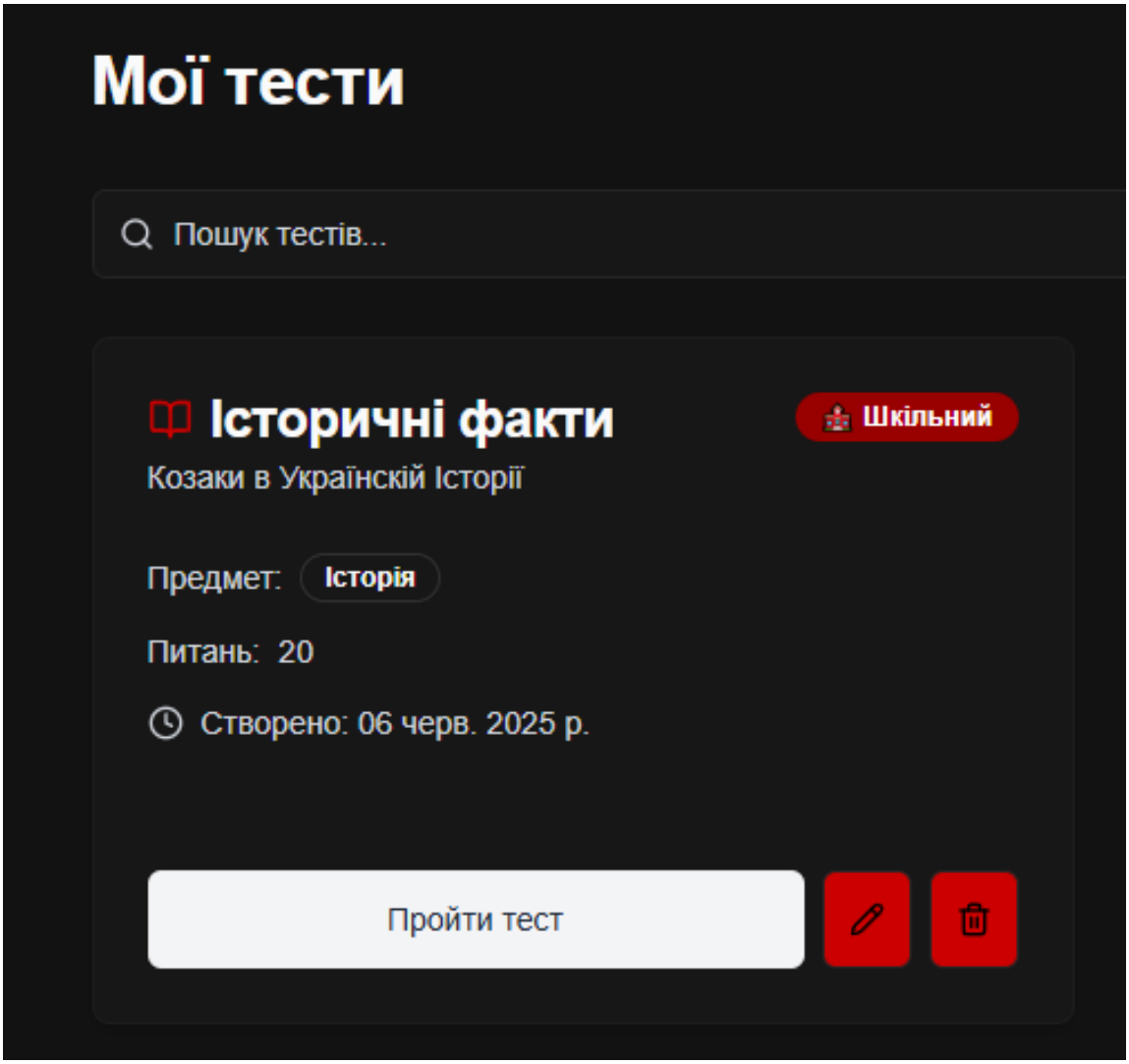

Рисунок 3.24 – Створений тест

Для виявлення візуальних недоліків чи проблем з компонентами у тесті було проведено функціональне тестування. За результатами якого нічого з раніше з попередньо перерахованого не відбулось. Усі 20 питань були пройдені і протестовані на виявлення можливих недоліків таких як: неробочі елементи керування запитаннями питаннями (назад та далі) , поля для вводу відповіді та елементів вибору відповіді з переліку (рис 3.25).

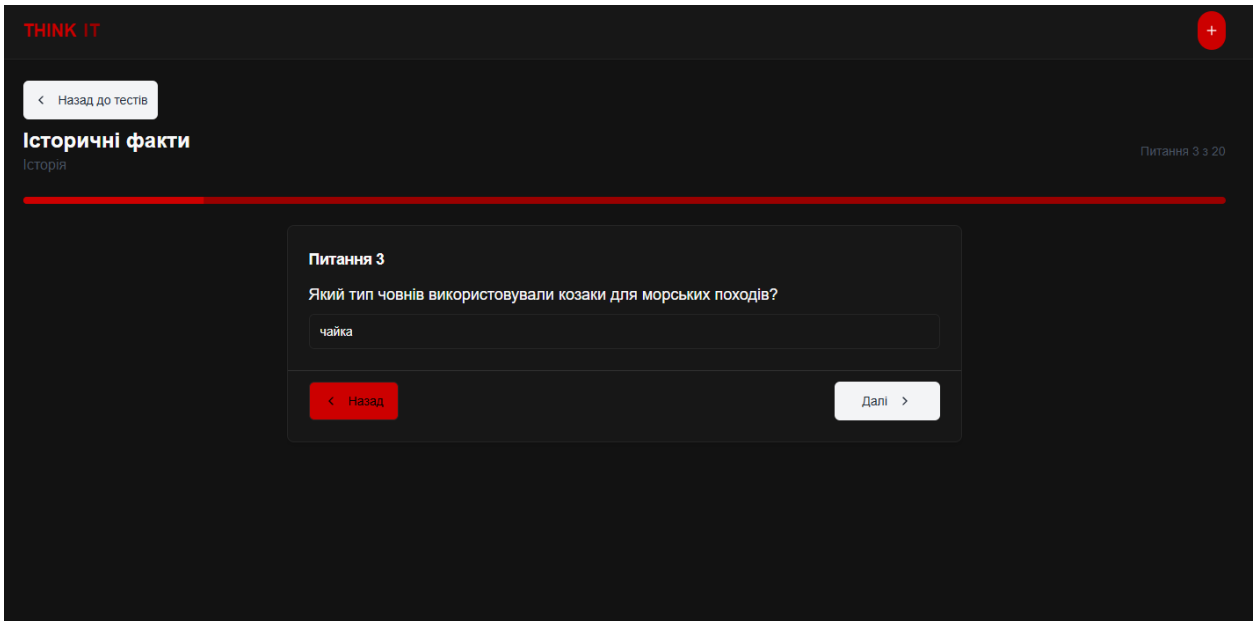

Рисунок 3.25 – Проходження тесту

Після проходження тесту отримуємо результати та перелік запитань з відповідями та аналізом які з них виявились правильними а на які відповідь була негативною (рис 3.26).

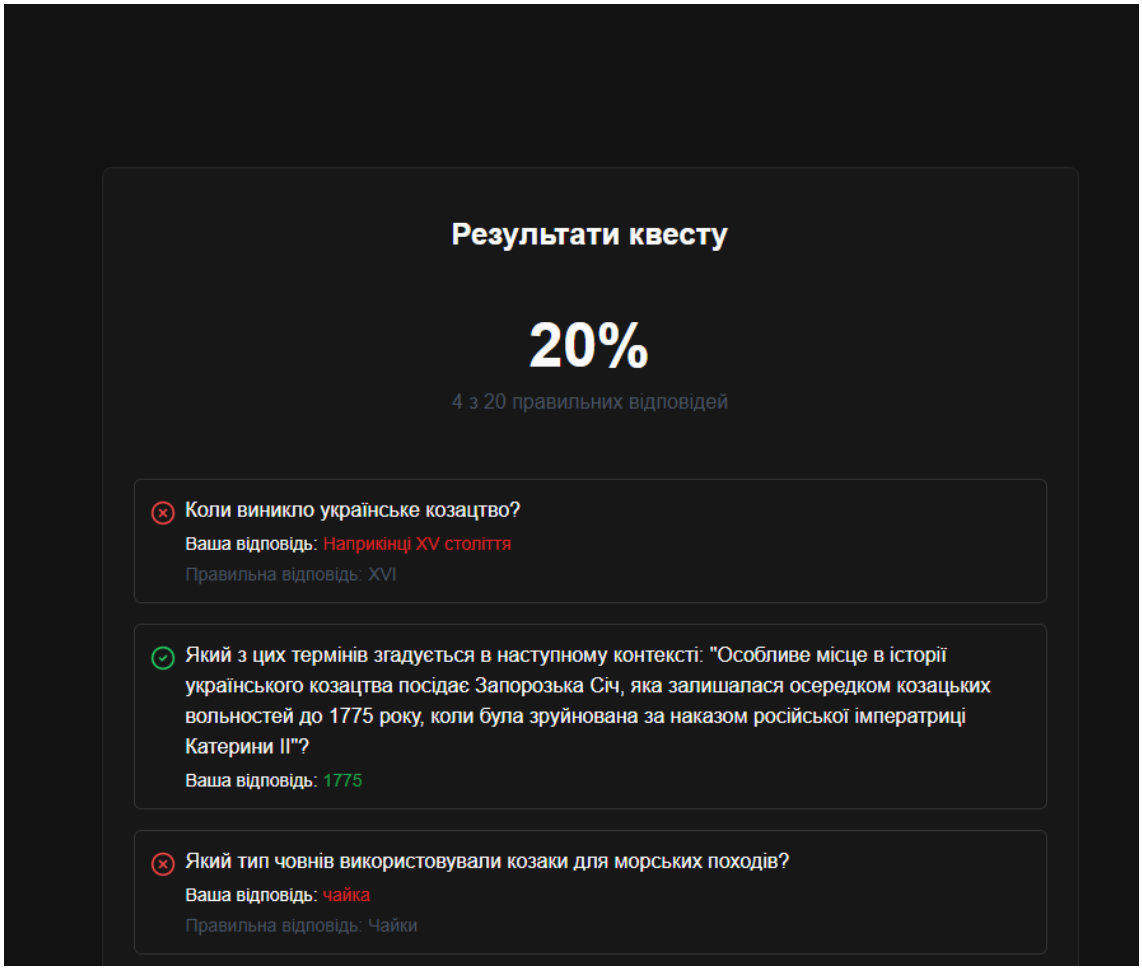

Рисунок 3.26 – Результати проходження

Після проходження користувач має можливість знову пройти тест якщо його не влаштувала оцінка або повернутись до інших тестів (рис 3.27).

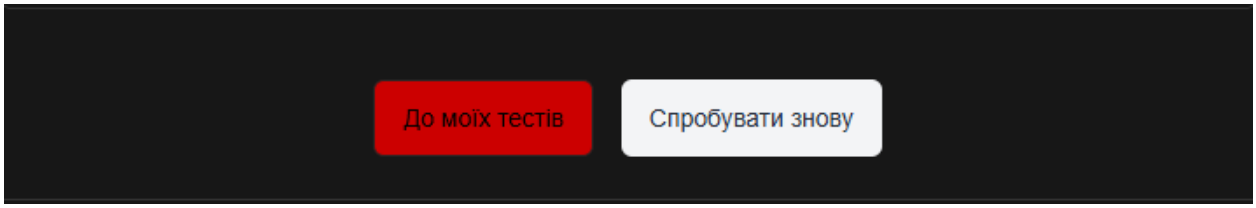

Рисунок 3.27 – Компонент управління після проходження

Якщо користувач пройшов тест ще раз чи йому потрібна статистика проходжень по цьому тесті він може побачити коли був пройдений тест та скільки було обрано правильних відповідей з відсотками (рис 3.28).

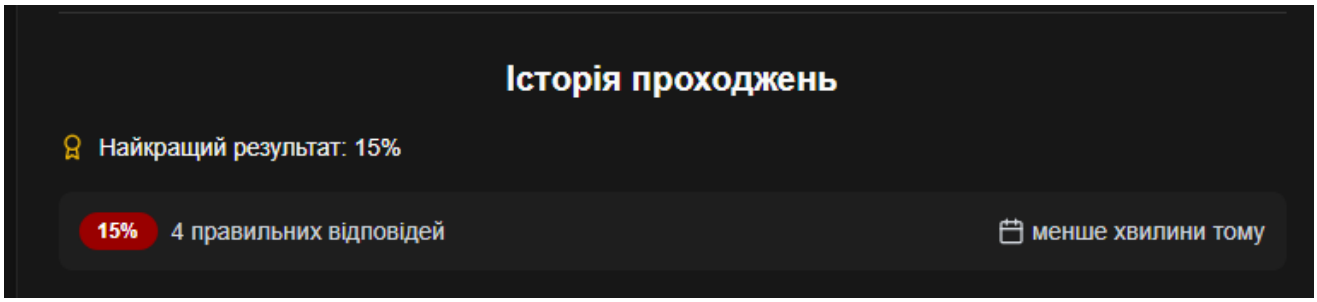

Рисунок 3.28 – Історія проходжень

Для перевірки механізму фільтрації було обрано фільтри по яким такого тесту ще не створено, за допомогою якого було протестовано компонент який повідомляє про невідповідність параметрів та надає змогу створити тест (рис 3.29).

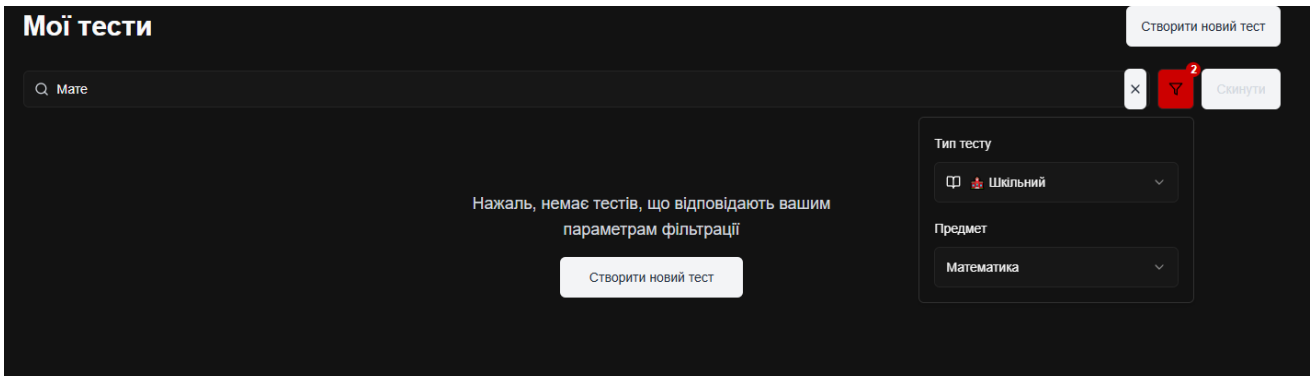

Рисунок 3.29 – Механізм фільтрації

На рисунку було обрано відповідні фільтри та параметри тесту для його пошуку. Цим ми повністю перевірили роботу фільтрації (рис 3.30).

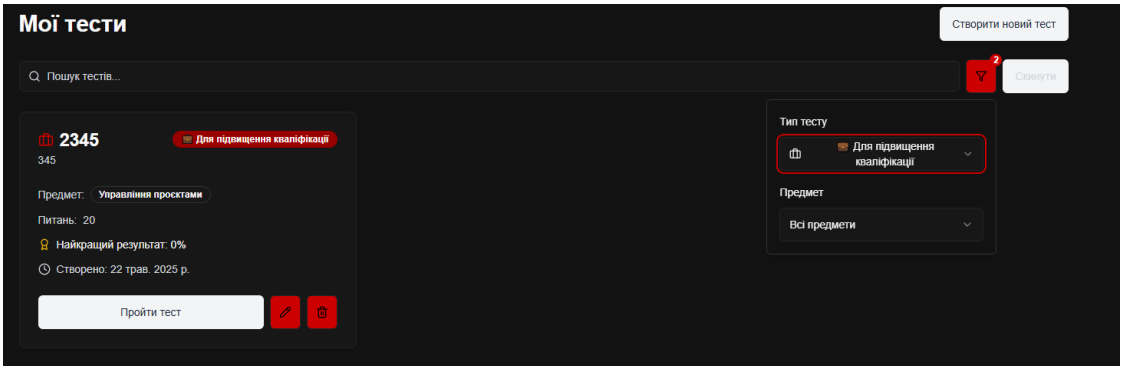

Рисунок 3.30 – Тестування фільтрації

Для перевірки роботи редагування тесту були протестовані компоненти в яких описана інформація про тест, тип та предмет за якими він зберігається, та питання (рис 3.31).

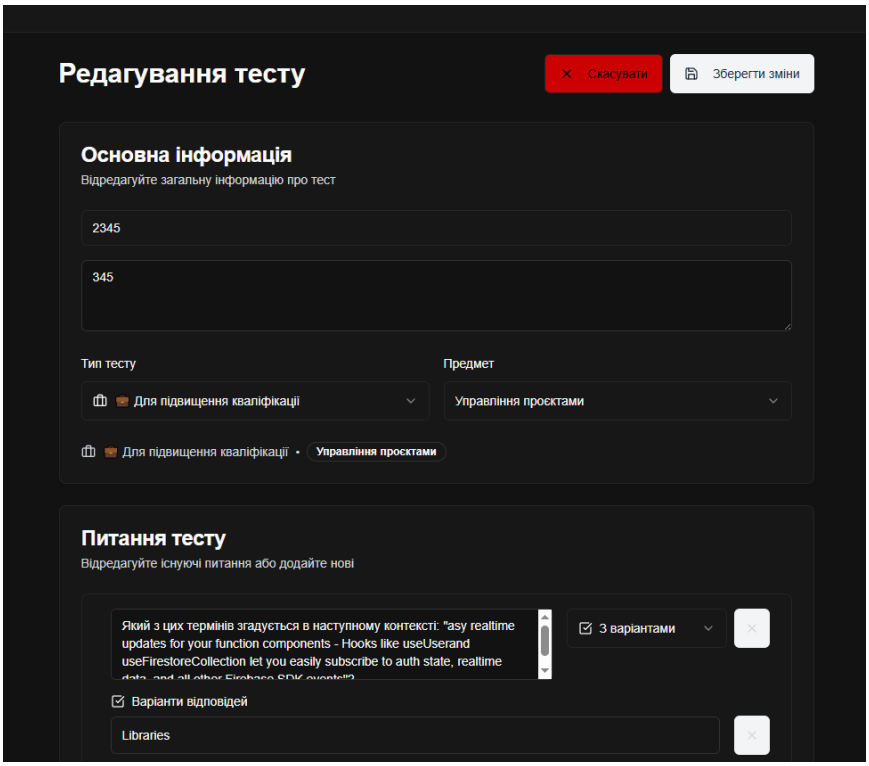

Рисунок 3.31 – Редагування тесту

3.6 Висновки

Для розроблення інтелектуальної технології генерування навчальних тестів були застосовані сучасні мови програмування, фреймворки та сервіси, що забезпечили ефективне функціонування системи. TypeScript та React стали основою клієнтської частини, а Firebase запропонував комплексне рішення для серверної інфраструктури. Використання Vite, Tailwind CSS та інших інструментів дозволило створити швидкий, адаптивний та зручний у користуванні вебдодаток.

У процесі розробки були застосовані поширені прийоми формування системних інструкцій для великої мовної моделі, що забезпечило отримання структурованих наборів тестових питань та відповідей. Стратегії zero-shot, few-shot, chain-of-thought та Enhanced Contextual Prompting дозволили точно адаптувати промпти до специфіки навчальних дисциплін. Розроблений системний промпт успішно інтегрований у вебдодаток, забезпечуючи якісну генерацію різнотипних тестових завдань.

Реалізація клієнтської частини на основі компонентної архітектури програмного коду дозволила створити функціональний та гнучкий користувацький інтерфейс. React-компоненти, такі як AITestGenerator, QuestCard, Header та інші, забезпечили зручну взаємодію користувачів із системою. Використання TypeScript підвищило типобезпеку та підтримуваність коду, а Tailwind CSS спростив стилізацію та адаптивність інтерфейсу для різних пристроїв.

Обрана екосистема serverless сервісів Google Firebase значно пришвидшила розробку інтелектуальної технології та покращила процес створення серверної частини. Firebase Firestore забезпечив надійне зберігання даних та синхронізацію у реальному часі, Firebase Authentication гарантував безпечну авторизацію, а Firebase Hosting спростив розгортання додатку через глобальну CDN мережу. Такий підхід дозволив делегувати складні серверні аспекти без втрати якості та функціональності інтелектуальної технології.

Розроблена інтелектуальна технологія може бути розгорнута як локально, так і в межах хмарного середовища, демонструючи гнучкість впровадження для різних організаційних потреб. Монолітна модульна архітектура з MVVM паттерном забезпечує легкість підтримки та розширення функціоналу, а використання Firebase створює можливості для швидкого масштабування (як вертикального, так і горизонтального) відповідно до зростання користувацького навантаження.

Комплексне тестування інтелектуальної технології THINK IT підтвердило стабільність і надійність створеного рішення. Процес верифікації охопив усі ключові компоненти системи: інтерфейсні елементи (головна сторінка, налаштування, створення тестів), функціональність генерації завдань (вибір типу, предмету, складності), обробку користувацьких текстів та підтримку різних типів питань. Технологія продемонструвала високу продуктивність при проходженні тестів, включаючи коректне відображення результатів, аналітики відповідей та збереження історії тестування. Особливої уваги заслуговує відсутність критичних помилок під час використання та ефективна робота системи фільтрації контенту, що суттєво покращує користувацький досвід. Результати тестування показали повну відповідність технології заявленим функціональним та нефункціональним вимогам, підтверджуючи її готовність до практичного використання в освітньому процесі.

ВИСНОВКИ

В результаті виконання бакалаврської кваліфікаційної роботи було проведено аналіз предметної області, представлено суть проблеми та доведена актуальність питання технології генерування тестів навчальних дисциплін.

Ця розробка повністю відповідає визначеній меті дослідження — Підвищити швидкість та якість генерування тестів для навчальних дисциплін шляхом створення інтелектуальної інформаційної технології на основі великої мовної моделі.

Аналіз предметної області продемонстрував актуальність і проблематику у сфері самонавчання і генерації освітніх тестів. Розглянуто основні можливості, переваги і недоліки існуючих аналогів. Було виявлено, що існуючі рішення (Kahoot!, Classmarker, Microsoft Forms) мають суттєві недоліки: залежність від постійного інтернет-з'єднання, обмежені можливості персоналізації, відсутність офлайн-режиму та складність інтерфейсу.

Розроблена інтелектуальна технологія "THINK IT" успішно вирішує ці проблеми, пропонуючи гнучке, масштабоване та доступне рішення для створення навчальних тестів.

Технологія ефективно поєднує можливості великих мовних моделей із сучасними вебтехнологіями, забезпечуючи якісний інструмент для оцінювання знань студентів та підвищення ефективності освітнього процесу.

Перевагою розробленої технології є використання великої мовної моделі GPT-3.5 від OpenAI, обраної завдяки високій якості генерації тексту українською мовою, стабільності API та гнучким можливостям налаштування. Для ефективної роботи з LLM був створений спеціалізований набір інструкцій (промптів) із використанням передових стратегій промпт-інженерії: zero-shot, few-shot, chain-of-thought та Enhanced Contextual Prompting. Це забезпечило генерацію релевантних та педагогічно обґрунтованих тестових завдань, адаптованих до різних дисциплін без необхідності створення окремих моделей.

Клієнтська частина технології реалізована на основі сучасного технологічного стеку (React.js, TypeScript, Tailwind CSS, Vite), що дозволило створити гнучкий, масштабований та зручний для користувача інтерфейс. Компонентна архітектура забезпечує модульність, спрощує підтримку та подальший розвиток технології. Серверна частина побудована на платформі Firebase, що забезпечує надійне зберігання даних, аутентифікацію користувачів, розгортання та функціонування вебдодатку в різних середовищах.

Перевагою технології "THINK IT" є можливість роботи в офлайн-режимі завдяки механізмам кешування та попереднього завантаження контенту в Firebase Firestore. Безпечна інтеграція з OpenAI API реалізована через Firebase Functions. Архітектурна модель MVVM (Model-View-ViewModel) забезпечує чіткий розподіл відповідальності між компонентами системи, підвищує тестованість та повторне використання коду.

Комплексне тестування технології підтвердило її стабільність, надійність та відповідність заявленим вимогам. Функціональні можливості, включаючи генерацію тестових завдань, проходження тестів, аналіз результатів та управління користувачами, реалізовані в повному обсязі. Нефункціональні вимоги (продуктивність, доступність, надійність, масштабованість, безпека, зручність використання) також успішно виконані.

Таким чином, усі поставлені задачі дослідження були виконані в повному обсязі, що підтверджує досягнення визначеної мети роботи. Інтелектуальна технологія "THINK IT" є інноваційним рішенням, що поєднує передові технології штучного інтелекту із сучасними підходами до веброзробки. Її практична значущість проявляється у значному скороченні часу на підготовку тестових матеріалів, підвищенні їх якості та різноманітності, а також у забезпеченні гнучких можливостей використання в різних освітніх сценаріях.

За результатами даної роботи була зроблена доповідь на тему «проектування освітньої платформи «THINK IT» на базі сучасних веб-

технологій» на Міжнародній науково-практичній інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (2025) з публікацією тез.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Яблонський Д.А., Лосенко А.В. проєктування освітньої платформи «THINK IT» на базі сучасних веб-технологій. Міжнародна науково-практична інтернет-конференція студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця,2025).URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25200(дата звернення: 25.05.2025).

2. Мосолова М. Д. Технології використання штучного інтелекту для адаптивного навчання у вищий освіті. 2024. URL:http://ds.knu.edu.ua/jspui/bitstream/123456789/6195/1/%D0%9A%D0%A0%D0%91%20%D0%9C%D0%BE%D1%81%D0%BE%D0%BB%D0%BE%D0%B2%D0%B0%20%D0%9C.%D0%94.%20%D0%9A%D0%86-20.pdf (дата звернення: 25.05.2025).

3. Progressive Web Apps Documentation . URL: https://web.dev/progressive-web-apps/ (дата звернення: 25.05.2025).

4. Медведєва М. О. Застосування технологій штучного інтелекту для автоматизації оцінювання знань у адаптивних навчальних системах // матеріали десятої міжнародної конференції з адаптивних технологій управління навчанням ATL-2024. Одеса, 2024. С. 57. URL: https://dspace.udpu.edu.ua/bitstream/123456789/17268/1/збірник_Медведєва.pdf (дата звернення: 25.05.2025).

5. Kahoot! – Інтерактивна освітня платформа. URL: https://kahoot.com/ (дата звернення: 25.05.2025).

6. Classmarker – Онлайн-система тестування. URL: https://www.classmarker.com/ (дата звернення: 25.05.2025).

7. Microsoft Forms – Сервіс для створення опитувань та тестів. URL: https://forms.microsoft.com/ (дата звернення: 25.05.2025).

8. Brownlee, J. Prompt Engineering for Effective Interaction with ChatGPT. Machine Learning Mastery, 2022. URL: https://machinelearningmastery.com/prompt-engineering-for-effective-interaction-with-chatgpt/ (дата звернення: 25.05.2025).

9. Google Gemini Documentation. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 25.05.2025).

10. Anthropic Claude Documentation. URL: https://docs.anthropic.com/en/home (дата звернення: 25.05.2025).

11. OpenAI ChatGPT Documentation. URL: https://platform.openai.com/docs/ (дата звернення: 25.05.2025).

12. Touvron, H., et al. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971, 2023. URL: https://arxiv.org/abs/2302.13971 (дата звернення: 25.05.2025).

13. Mistral AI Models Documentation. URL: https://docs.mistral.ai (дата звернення: 25.05.2025).

14. OpenAI Text Generation Guide. URL: https://platform.openai.com/docs/guides/text?api-mode=responses (дата звернення: 25.05.2025).

15 Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs (дата звернення: 25.05.2025).

16. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення: 25.05.2025).

17. TypeScript Documentation . URL: https://www.typescriptlang.org/docs/ (дата звернення: 25.05.2025).

18. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall. 2008. Aug 11; 464 p. URL: https://ptgmedia.pearsoncmg.com/images/9780132928472/samplepages/0132928477.pdf (дата звернення: 25.05.2025).

19. Tailwind CSS Framework Docs. URL: https://tailwindcss.com/docs (дата звернення: 25.05.2025).

20. Vite Build Tool Docs. URL: https://vitejs.dev/guide/ (дата звернення: 25.05.2025).

21. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2022. Nov 15; 672 p. URL: https://dl.ebooksworld.ir/motoman/Patterns%20of%20Enterprise%20Application%20Architecture.pdf (дата звернення: 25.05.2025).

22. Firebase Documentation. URL: https://firebase.google.com/docs (дата звернення: 25.05.2025).

23. Serverless Computing: Economic and Architectural Impact. IEEE Cloud Computing, 2022. URL: https://ieeexplore.ieee.org/document/8451770 (дата звернення: 25.05.2025).

24. OpenAI GPT-4 Prompt Engineering Guide. URL: https://cookbook.openai.com/examples/gpt4-1_prompting_guide (дата звернення: 25.05.2025).

25. Kelleher J. D. Deep Learning. MIT Press Essential Knowledge Series. MIT Press. 2023; 296 p. URL: https://mitpress.mit.edu/9780262537551/deep-learning/

26. Brown S, Christudas B. Microservices for the Enterprise: Designing, Developing, and Deploying. Apress. 2023. Dec 20; 428 p. doi: https://doi.org/10.1007/978-1-4842-3858-5

27. Microsoft Docs. MVVM pattern. 2024. URL: https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (дата звернення: 25.05.2025).

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
“____”_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу

ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ ГЕНЕРУВАННЯ ТЕСТІВ НАВЧАЛЬНИХ ДИСЦИПЛІН

08-34.БКР.027.00.000 ТЗ

Керівник: PhD., асистент
_____ Арсен ЛОСЕНКО
«__» _____ 2025 р.
Розробив студент гр. 2ІСТ-21б
_____Денис ЯБЛОНСЬКИЙ
«__» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____ 2025р., та індивідуальне завдання на БКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.

2. Джерела розробки:

1) Kelleher J. D. Deep Learning. MIT Press Essential Knowledge Series. MIT Press. 2023; 296 p. URL: https://mitpress.mit.edu/9780262537551/deep-learning/;

2) Touvron, H., et al. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971, 2023. doi: https://doi.org/10.48550/arXiv.2302.13971

3. Мета і призначення роботи.

Метою дослідження є підвищити швидкість та якість генерування тестів для навчальних дисциплін шляхом створення інтелектуальної інформаційної технології на основі великої мовної моделі.

4. Вихідні дані для проведення робіт:

Дані про теми навчальних дисциплін для генерування навчальних тестів та про формування інструкцій для роботи з великими мовними моделями

5. Методи дослідження:

Дослідження існуючих інтелектуальних технологій, розробка UML-діаграм, створення компонентів вебінтерфейсу, клієнт-серверна взаємодія через HTTP-запити та платформу Firebase, тестування застосунку.

6. Етапи роботи і терміни їх виконання:

a) Аналіз предметної області _____ – _____
b) Аналіз існуючих інтелектуальних технологій _____ – _____
c) Вибір оптимальних інформаційних технологій _____ – _____
d) Розроблення інтелектуальної технології генерування навчальних тестів _____ – _____
e) Оформлення матеріалів до захисту БКР _____ – _____

7. Очікувані результати та порядок реалізації

Отримання інтелектуальної технології для генерування навчальних тестів.

8. Вимоги до розробленої документації

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист «___» _____ 2025 р.
Початок розробки «___» _____ 2025 р.
Граничні терміни виконання БКР «___» _____ 2025 р.

Розробив студент групи 2ІСТ-21б _____ Денис ЯБЛОНСЬКИЙ

Додаток Б

(обов’язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Інтелектуальна технологія генерування тестів навчальних дисциплін»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1.12%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- □ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- □ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____ (підпис)

Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ _____ (підпис)

Особа, відповідальна за перевірку _____ (підпис) Сергій ЖУКОВ

З висновком експертної комісії ознайомлений(-на)

Керівник _____ (підпис) Арсен ЛОСЕНКО, PhD, ас. каф. САІТ

Здобувач _____ Денис ЯБЛОНСЬКИЙ

Додаток В

(довідниковий)

Фрагмент лістингу програми

Системний промпт:

```
// Enum для типів тестів
enum TestType {
  SCHOOL = 'SCHOOL',
  UNIVERSITY = 'UNIVERSITY',
  SELF_LEARNING = 'SELF_LEARNING',
  PROFESSIONAL = 'PROFESSIONAL',
  TRAINING = 'TRAINING'
}

// Базові інструкції для формату відповіді
const baseInstructions = `
На основі наданого тексту створи різноманітні запитання для тестування знань.

Формат відповіді:
Питання: [текст питання]
Варіанти:
A) [варіант відповіді]
B) [варіант відповіді]
C) [варіант відповіді]
D) [варіант відповіді]
Правильна відповідь: [повний текст правильної відповіді]

АБО для відкритих питань:
Питання: [текст питання з пропуском]
Правильна відповідь: [точна відповідь з тексту]
`;

// Загальні вимоги до генерації питань
const generalRequirements = `
Вимоги до генерації питань:
1. БАЗОВІ ПРИНЦИПИ:
   - Використовуй ТІЛЬКИ інформацію з наданого тексту
   - Кожне питання має перевіряти конкретне знання з тексту
   - Уникай неоднозначних формулювань
   - Правильна відповідь має бути очевидною при знанні матеріалу

2. СТРУКТУРА ПИТАНЬ:
   - Для питань з варіантами: рівно 4 варіанти (A, B, C, D)
   - Для відкритих питань: конкретна однозначна відповідь
   - Правильна відповідь ОБОВ'ЯЗКОВО має точно співпадати з одним з варіантів A-D

3. ТИПИ ПИТАНЬ ДЛЯ ВАРІАНТІВ:
   - Заповнення пропусків у реченнях з тексту
   - Вибір правильного визначення або пояснення
   - Встановлення причинно-наслідкових зв'язків
   - Визначення хронології подій
   - Питання на розуміння контексту

4. РОЗПОДІЛ ПО ТИПАХ:
   - 50% питань з відкритою відповіддю (заповнення пропусків)
   - 50% питань з варіантами відповідей (множинний вибір)

5. ВИМОГИ ДО ВАРІАНТІВ ВІДПОВІДЕЙ:
```

```
   - Правильна відповідь: точна інформація з тексту
   - Неправильні варіанти мають бути:
     * Схожими за форматом та довжиною
     * Логічно пов'язаними з темою
     * Правдоподібними, але точно неправильними
     * Граматично правильними

6. ВИМОГИ ДО ПРОПУСКІВ:
   - Вибирай ключові фрази, терміни, дати для пропуску
   - Правильна відповідь має точно підходити за контекстом
   - Пропуск має бути в найважливішому місці речення

7. КОНТРОЛЬ ЯКОСТІ:
   - Всі варіанти мають бути унікальними
   - НЕ використовуй варіанти "всі відповіді правильні" або "жодна відповідь не правильна"
   - Уникай дублювання питань на однакову тему
   - Перевіряй логічність та зрозумілість кожного питання

8. АДАПТАЦІЯ ДО КОНТЕНТУ:
   - Для історичних текстів: фокус на датах, подіях, особах, причинах і наслідках
   - Для технічних текстів: термінологія, процеси, взаємозв'язки
   - Для літературних текстів: сюжет, персонажі, художні засоби
   - Для наукових текстів: факти, теорії, експерименти

9. РІВЕНЬ СКЛАДНОСТІ:
   - Питання мають відповідати рівню складності тексту
   - Баланс між простими фактологічними та складнішими аналітичними питаннями
   - Уникай занадто очевидних або занадто складних питань

10. МОВНІ ВИМОГИ:
    - Використовуй українську мову
    - Дотримуйся правил граматики та орфографії
    - Використовуй термінологію, що відповідає контексту тексту
`;

// Інструкції для різних типів тестів
const testTypeInstructions = {
  [TestType.SCHOOL]: `
Додаткові вимоги для ШКІЛЬНИХ тестів:
- Використовуй простий та зрозумілий виклад
- Фокусуйся на базових фактах та основних поняттях
- Уникай занадто складних аналітичних питань
- Питання мають відповідати шкільній програмі
- Використовуй знайому учням термінологію
`,

  [TestType.UNIVERSITY]: `
Додаткові вимоги для УНІВЕРСИТЕТСЬКИХ тестів:
- Включай питання вищого рівня складності
- Фокусуйся на аналізі, синтезі та критичному мисленні
- Використовуй професійну термінологію
- Включай питання на встановлення зв'язків між концепціями
- Перевіряй глибоке розуміння матеріалу
`,

  [TestType.SELF_LEARNING]: `
Додаткові вимоги для САМОНАВЧАННЯ:
- Створюй питання, що допомагають закріпити знання
- Включай пояснювальні елементи в питання
- Фокусуйся на практичному застосуванні знань
- Створюй питання для самоперевірки розуміння
- Використовуй різноманітні формати питань
```

```
[TestType.PROFESSIONAL]: `
Додаткові вимоги для ПРОФЕСІЙНИХ тестів:
- Фокусуйся на практичних навичках та знаннях
- Включай ситуаційні питання
- Використовуй професійну термінологію галузі
- Створюй питання на застосування знань у роботі
- Перевіряй знання стандартів та процедур
`,

  [TestType.TRAINING]: `
Додаткові вимоги для ТРЕНІНГОВИХ тестів:
- Створюй питання для перевірки засвоєння навчального матеріалу
- Фокусуйся на ключових навичках та компетенціях
- Включай питання на практичне застосування
- Використовуй формат, що підходить для тренінгу
- Створюй питання для контролю прогресу навчання
`
};

// Інструкції для різних предметів
const subjectInstructions = {
  math: `
Специфічні вимоги для МАТЕМАТИКИ:
- Фокусуйся на формулах, теоремах, властивостях
- Включай числові приклади та обчислення
- Створюй питання на застосування математичних правил
- Використовуй точну математичну термінологію
`,
  history: `
Специфічні вимоги для ІСТОРІЇ:
- Фокусуйся на датах, подіях, історичних особах
- Включай питання про причини та наслідки подій
- Використовуй хронологічні зв'язки
- Створюй питання про історичний контекст
`,

  philosophy: `
Специфічні вимоги для ФІЛОСОФІЇ:
- Фокусуйся на філософських концепціях та ідеях
- Включай питання про філософів та їхні теорії
- Створюй питання на розуміння абстрактних понять
- Використовуй філософську термінологію
`,

  economics: `
Специфічні вимоги для ЕКОНОМІКИ:
- Фокусуйся на економічних термінах та процесах
- Включай питання про економічні закони та принципи
- Створюй питання про ринкові механізми
- Використовуй економічну термінологію
`,

  programming: `
Специфічні вимоги для ПРОГРАМУВАННЯ:
- Фокусуйся на синтаксисі, алгоритмах, структурах даних
- Включай питання про принципи програмування
- Створюй питання про технології та інструменти
- Використовуй технічну термінологію IT
`,

  languages: `
Специфічні вимоги для МОВНИХ дисциплін:
```

```
- Фокусуйся на граматиці, лексиці, синтаксисі
- Включай питання про мовні правила та структури
- Створюй питання на розуміння тексту
- Використовуй відповідну лінгвістичну термінологію
`,

  project_management: `
Специфічні вимоги для УПРАВЛІННЯ ПРОЕКТАМИ:
- Фокусуйся на методологіях та процесах
- Включай питання про інструменти управління
- Створюй ситуаційні питання
- Використовуй термінологію проектного менеджменту
`,

  cybersecurity: `
Специфічні вимоги для КІБЕРБЕЗПЕКИ:
- Фокусуйся на загрозах, захисті, протоколах
- Включай питання про безпекові практики
- Створюй питання про технічні рішення
- Використовуй термінологію інформаційної безпеки
`,

  quick_review: `
Специфічні вимоги для ШВИДКОГО ОГЛЯДУ:
- Створюй короткі та лаконічні питання
- Фокусуйся на ключових фактах
- Уникай складних аналітичних питань
- Питання мають бути швидкими для відповіді
`,

  general_knowledge: `
Специфічні вимоги для ЗАГАЛЬНИХ ЗНАНЬ:
- Створюй різноманітні питання по темі
- Фокусуйся на найважливіших аспектах
- Включай як фактологічні, так і концептуальні питання
- Використовуй загальнозрозумілу термінологію
`
};

// Функція для генерації промпта
function generatePromptForQuestions(
  text: string,
  count: number,
  testType: TestType = TestType.GENERAL_KNOWLEDGE,
  subject: string = 'general_knowledge'
): string {
  const multipleChoiceCount = Math.ceil(count / 2);
  const openQuestionsCount = Math.floor(count / 2);

  return `
${baseInstructions}

${generalRequirements}

${testTypeInstructions[testType]  ''}

${subjectInstructions[subject]  ''}

КІЛЬКІСТЬ ПИТАНЬ:
- Загальна кількість: ${count}
- Питань з варіантами відповідей: ${multipleChoiceCount}
- Відкритих питань: ${openQuestionsCount}
```

```
Текст для аналізу: ${text}
  `.trim();
}
```

Сторінка створення тесту:

```
import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { Textarea } from "@/components/ui/textarea";
import { Header } from "@/components/Header";
import { QuestionForm } from "@/components/QuestionForm";
import { AITestGenerator } from "@/components/AITestGenerator";
import { useAuth } from "@/contexts/AuthContext";
import { TestType, TestTypeLabels, Question, DefaultSubjects } from "@/types";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { questsService } from "@/lib/questsService";
import { useToast } from "@/components/ui/use-toast";
import {
  BookOpen,
  GraduationCap,
  Lightbulb,
  Briefcase,
  Brain,
  Plus,
  Save,
  X,
} from "lucide-react";
import { Badge } from "@/components/ui/badge";
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";

const typeIcons = {
  school: BookOpen,
  university: GraduationCap,
  self_learning: Lightbulb,
  professional: Briefcase,
  training: Brain,
};

export default function CreateQuest() {
  const navigate = useNavigate();
  const { currentUser } = useAuth();
  const { toast } = useToast();

  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [questions, setQuestions] = useState<Question[]>([]);
  const [isAIMode, setIsAIMode] = useState(false);
  const [isSaving, setIsSaving] = useState(false);
```

```
const [selectedType, setSelectedType] = useState<TestType>(() => {
  const state = window.history.state?.usr;
  return state?.type || TestType.SCHOOL;
});
const [selectedSubject, setSelectedSubject] = useState("");

// Автоматично вибираємо перший доступний предмет для обраного типу тесту
useEffect(() => {
  const subjects = DefaultSubjects.filter((subject) =>
    subject.testTypes.includes(selectedType)
  );
  if (subjects.length > 0 && !selectedSubject) {
    setSelectedSubject(subjects[0].id);
  }
}, [selectedType]);

const availableSubjects = DefaultSubjects.filter((subject) =>
  subject.testTypes.includes(selectedType)
);

const handleQuestionsGenerated = (generatedQuestions: Question[]) => {
  setQuestions(generatedQuestions);
  setIsAIMode(false);
  toast({
    title: "Питання згенеровано",
    description: `Створено ${generatedQuestions.length} питань. Ви можете їх відредагувати.`,
  });
};

const handleSubmit = async () => {
  if (!currentUser) {
    toast({
      variant: "destructive",
      title: "Помилка",
      description: "Будь ласка, увійдіть в систему для створення тесту",
    });
    return;
  }

  setIsSaving(true);
  try {
    const questData = {
      title,
      description,
      type: selectedType,
      subject: selectedSubject,
      questions,
      authorId: currentUser.uid,
      createdAt: Date.now(),
      updatedAt: Date.now(),
    };

    await questsService.createQuest(questData);

    toast({
      title: "Успіх!",
      description: "Тест успішно створено",
    });

    navigate("/");
  } catch (error) {
    console.error("Помилка при створенні тесту:", error);
```

```
    toast({
      variant: "destructive",
      title: "Помилка",
      description: "Не вдалося створити тест. Спробуйте ще раз.",
    });
  } finally {
    setIsSaving(false);
  }
};

const TypeIcon = typeIcons[selectedType];
const subject = availableSubjects.find((s) => s.id === selectedSubject);

return (
  <div className="min-h-screen flex flex-col bg-background text-foreground">
    <Header />

    <main className="flex-1 container mx-auto max-w-4xl px-4 py-6 sm:p-6">
      <div className="flex flex-col sm:flex-row sm:items-center justify-between gap-4 mb-8">
        <h1 className="text-2xl sm:text-3xl font-bold">Створення тесту</h1>
        <div className="flex items-center gap-2 w-full sm:w-auto">
          <Button
            variant="outline"
            onClick={() => navigate("/")}
            className="flex-1 sm:flex-initial"
          >
            <X className="h-4 w-4 mr-2" />
            Скасувати
          </Button>
          <Button
            onClick={handleSubmit}
            disabled={
              !title ||
              !selectedType ||
              !selectedSubject ||
              questions.length === 0 ||
              isSaving
            }
            className="flex-1 sm:flex-initial"
          >
            <Save className="h-4 w-4 mr-2" />
            {isSaving ? "Збереження..." : "Зберегти"}
          </Button>
        </div>
      </div>

      <div className="space-y-4 sm:space-y-6">
        <Card>
          <CardHeader>
            <CardTitle>Основна інформація</CardTitle>
            <CardDescription>
              Введіть загальну інформацію про тест
            </CardDescription>
          </CardHeader>
          <CardContent className="space-y-4 sm:space-y-6">
            <div className="grid gap-4">
              <Input
                placeholder="Назва тесту"
                value={title}
                onChange={(e) => setTitle(e.target.value)}
              />
              <Textarea
```

```
    placeholder="Опис тесту"
    value={description}
    onChange={(e) => setDescription(e.target.value)}
   />
  </div>

  <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
   <div className="space-y-2">
    <label className="text-sm font-medium">Тип тесту</label>
    <Select
     value={selectedType}
     onValueChange={(value: string) =>
      setSelectedType(value as TestType)
     }
    >
     <SelectTrigger>
      <SelectValue>
       {selectedType && (
        <div className="flex items-center gap-2">
         {TypeIcon && <TypeIcon className="h-4 w-4" />}
         <span>{TestTypeLabels[selectedType]}</span>
        </div>
       )}
      </SelectValue>
     </SelectTrigger>
     <SelectContent>
      {Object.values(TestType).map((type) => {
       const Icon = typeIcons[type];
       return (
        <SelectItem key={type} value={type}>
         <div className="flex items-center gap-2">
          {Icon && <Icon className="h-4 w-4" />}
          <span>{TestTypeLabels[type]}</span>
         </div>
        </SelectItem>
       );
      })}
     </SelectContent>
    </Select>
   </div>

   <div className="space-y-2">
    <label className="text-sm font-medium">Предмет</label>
    <Select
     value={selectedSubject}
     onValueChange={setSelectedSubject}
     disabled={availableSubjects.length === 0}
    >
     <SelectTrigger>
      <SelectValue placeholder="Оберіть предмет" />
     </SelectTrigger>
     <SelectContent>
      {availableSubjects.map((subject) => (
       <SelectItem key={subject.id} value={subject.id}>
        {subject.name}
       </SelectItem>
      ))}
     </SelectContent>
    </Select>
   </div>
  </div>
```

```
    {selectedType && selectedSubject && (
     <div className="flex items-center gap-2 text-sm text-muted-foreground">
      <TypeIcon className="h-4 w-4" />
      <span>{TestTypeLabels[selectedType]}</span>
      <span>•</span>
      <Badge variant="outline">{subject?.name}</Badge>
     </div>
    )}
   </CardContent>
  </Card>

  <Card>
   <CardHeader>
    <CardTitle>Питання тесту</CardTitle>
    <CardDescription>
     Створіть питання вручну або згенеруйте їх автоматично
    </CardDescription>
   </CardHeader>
   <CardContent className="space-y-6">
    <div className="flex flex-col sm:flex-row justify-center gap-4">
     <Button
      variant={isAIMode ? "outline" : "default"}
      onClick={() => setIsAIMode(false)}
      className="w-full sm:w-40"
     >
      Створити вручну
     </Button>
     <Button
      variant={isAIMode ? "default" : "outline"}
      onClick={() => setIsAIMode(true)}
      className="w-full sm:w-40"
     >
      Згенерувати з тексту
     </Button>
    </div>

    {isAIMode ? (
     <AITestGenerator
      onQuestionsGenerated={handleQuestionsGenerated}
     />
    ) : (
     <div className="space-y-4">
      {questions.map((question, index) => (
       <QuestionForm
        key={question.id}
        question={question}
        onChange={(updatedQuestion) => {
         const newQuestions = [...questions];
         newQuestions[index] = updatedQuestion;
         setQuestions(newQuestions);
        }}
        onDelete={() => {
         setQuestions(
          questions.filter((q) => q.id !== question.id)
         );
        }}
       />
      ))}

      <Button
       variant="outline"
       onClick={() => {
```

```
                  setQuestions([
                    ...questions,
                    {
                      id: Date.now().toString(),
                      text: "",
                      type: "multiple",
                      correctAnswer: "",
                      options: ["", "", ""],
                    },
                  ]);
                }}
                className="w-full"
              >
                <Plus className="h-4 w-4 mr-2" />
                Додати питання
              </Button>
            </div>
          )}
        </CardContent>
      </Card>
    </div>
  </main>
 </div>
 );
}
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ ГЕНЕРУВАННЯ ТЕСТІВ НАВЧАЛЬНИХ ДИСЦИПЛІН

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

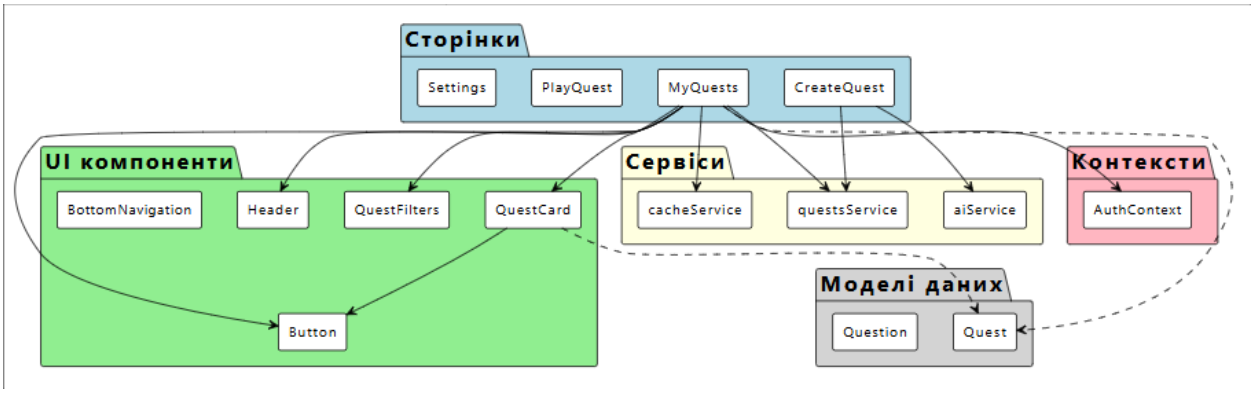

Рисунок Г.1 – Структура компонентів інтелектуальної технології

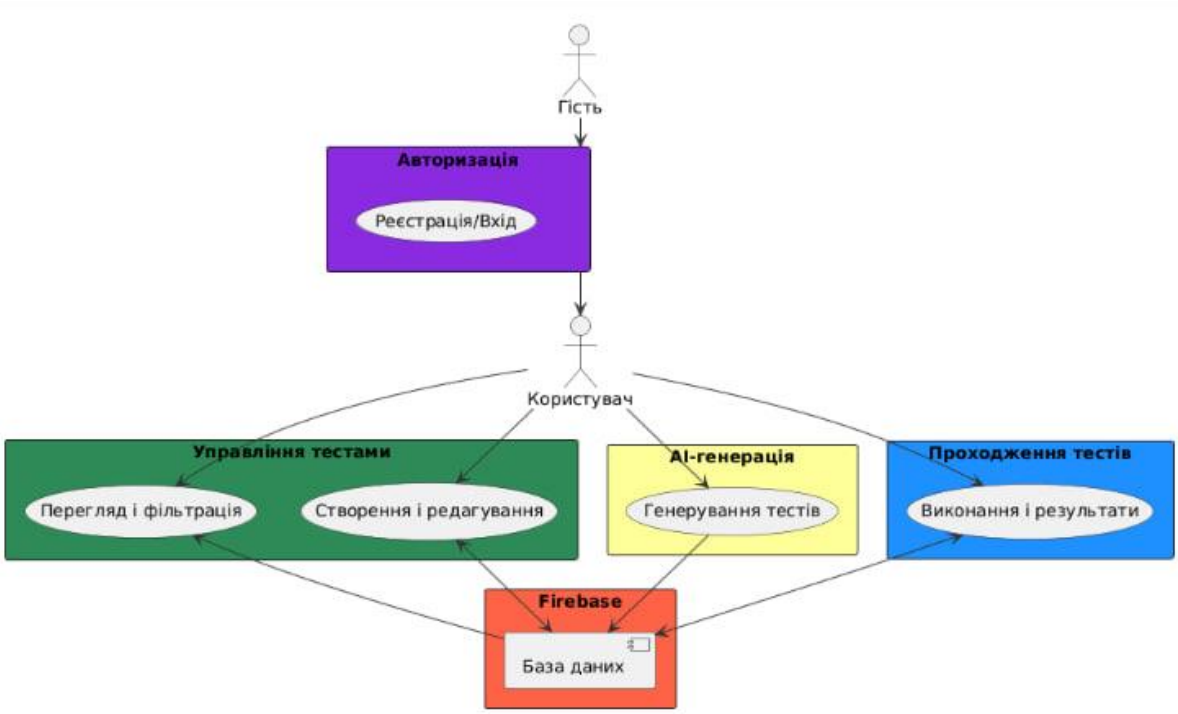

Рисунок Г.2 – Діаграма модулів реалізації бізнес-логіки

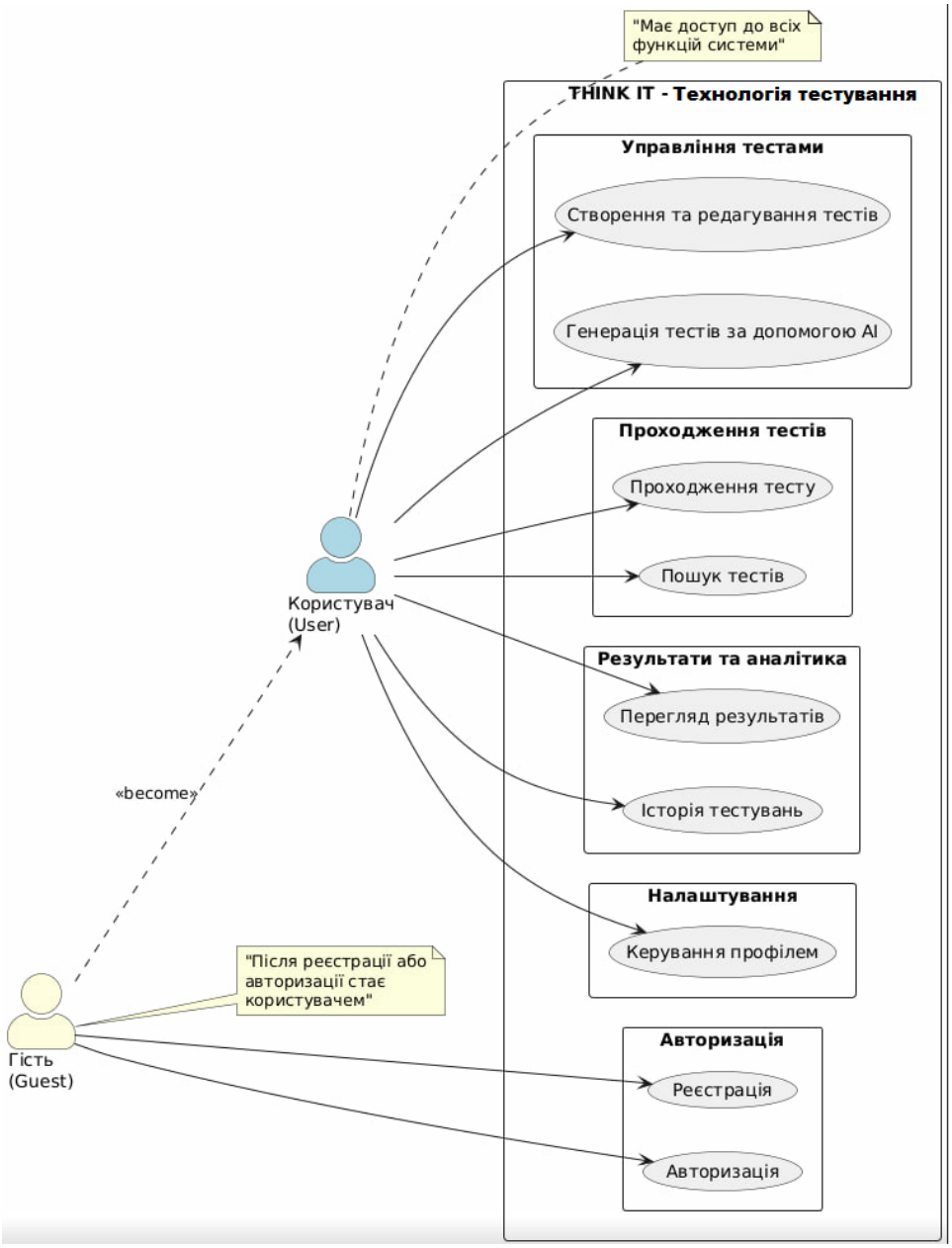

Рисунок Г.3 – Use case діаграма інтелектуальної технології

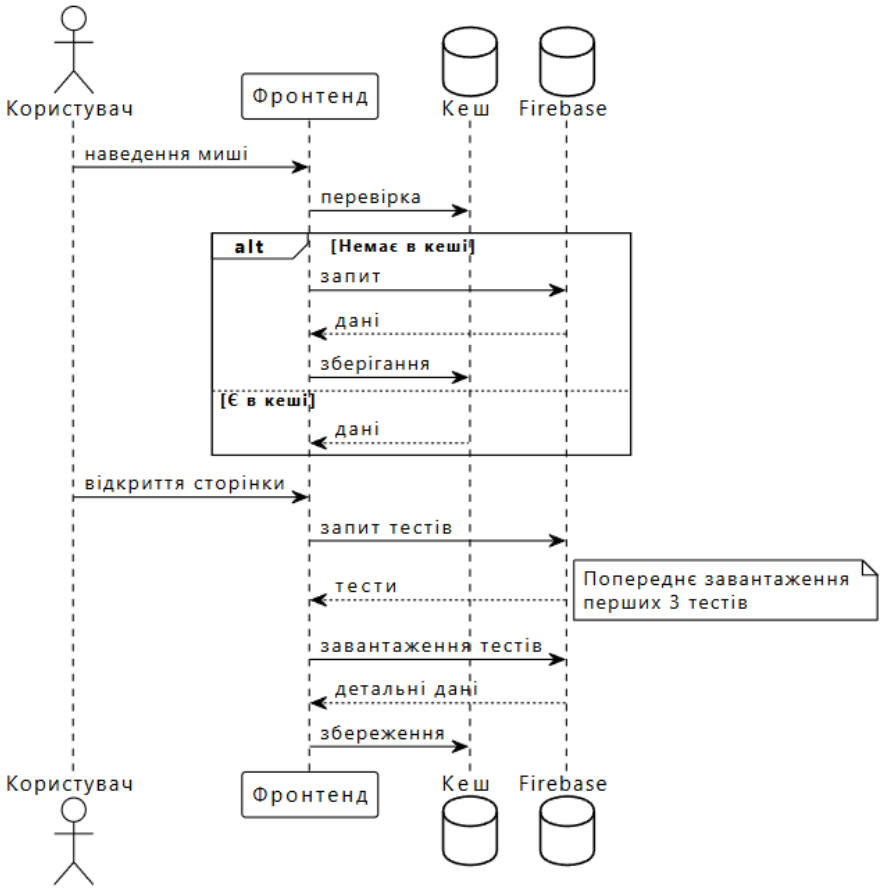

Рисунок Г.4 – UML-діаграма послідовності механізму кешування у вебдодатку

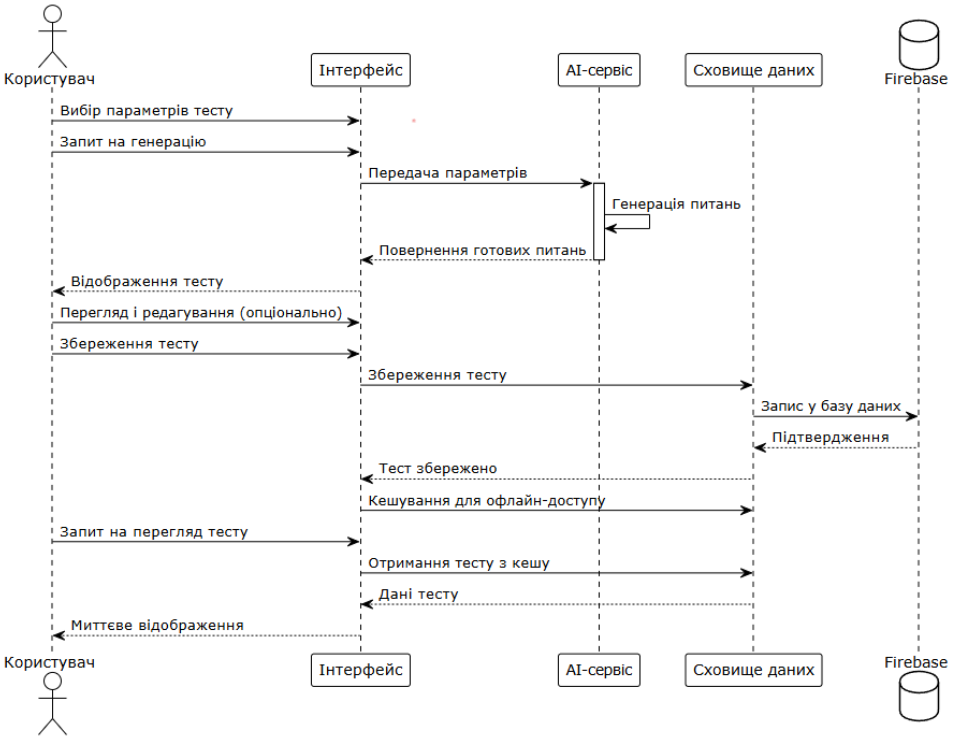

Рисунок Г.5 – UML-діаграма послідовності механізму генерації тестів та після отримання його для проходження

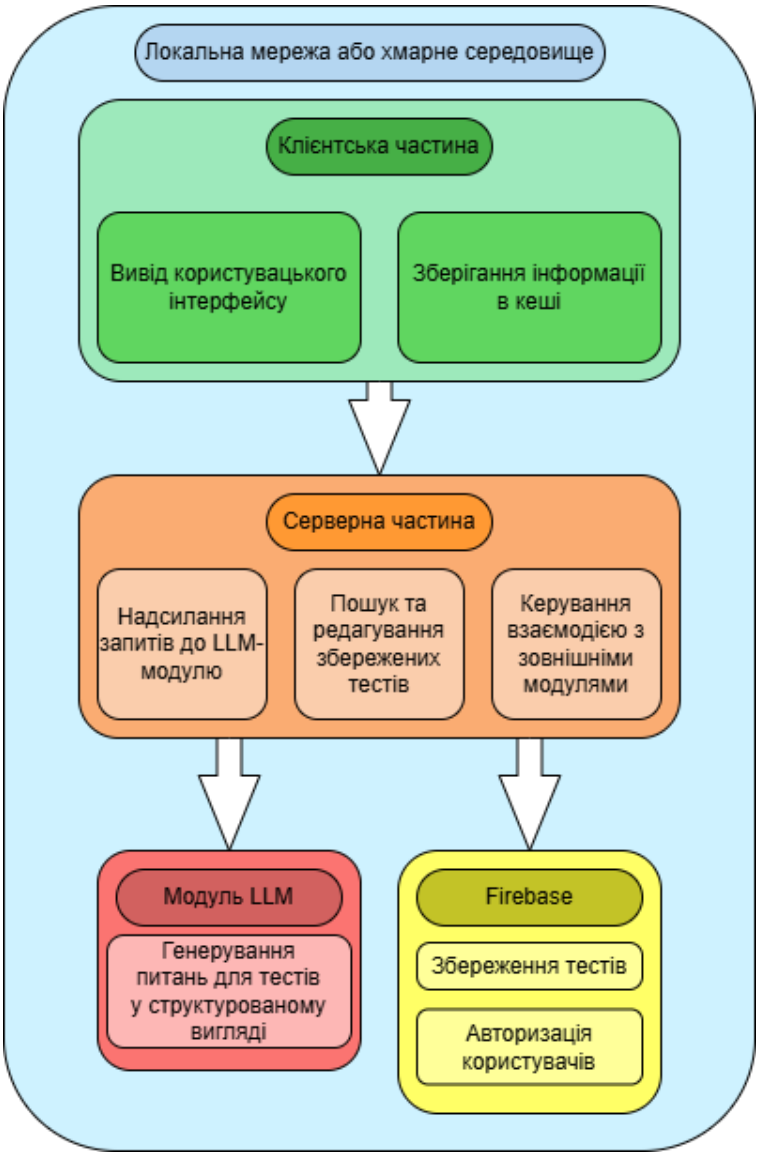

Рисунок Г.6 – Архітектура інтелектуальної технології

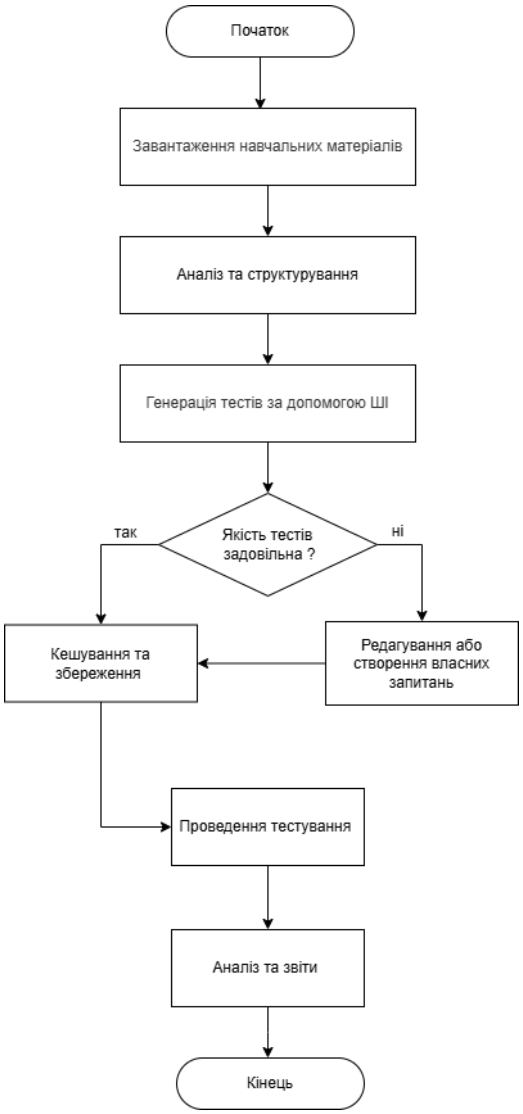

Рисунок Г.7 – Блок-схема алгоритму модулю генерування навчального тесту