Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

Система генерації тестових завдань та контрольних питань на основі силабусів з використанням ШІ

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121«Інженерія
програмного забезпечення»
Борис Артем Олександрович
Науковий керівник:
к.пед.н., доц. Петренко С. В.

Рівне – 2026

ЗМІСТ

ВСТУП

Автоматизація рутинних процесів у закладах вищої освіти набуває дедалі більшого поширення в контексті цифровізації освітнього середовища. Одним із напрямів такої автоматизації є формування контрольних матеріалів – тестових завдань та контрольних питань – на основі навчальних програм дисциплін. Силабус як структурований документ, що містить теми, цілі навчання, компетентності та рекомендовану літературу, є природною основою для побудови системи автоматизованої генерації завдань. Водночас ручне створення тестових матеріалів вимагає значних витрат часу від викладача, що обумовлює доцільність розробки інструменту, який делегує цю функцію засобам штучного інтелекту [1, с. 42].

Розвиток великих мовних моделей відкрив принципово нові можливості для обробки неструктурованого тексту та генерації змістовних відповідей на його основі. Такі моделі здатні аналізувати навчальний текст і формувати питання різних типів – з вибором відповіді та питання з короткою відповіддю – зберігаючи при цьому змістовий зв'язок із вхідним матеріалом [2]. Застосування подібних моделей у задачах автоматичної генерації питань (Automatic Question Generation, AQG) є предметом досліджень починаючи з початку 2020-х років, і результати свідчать про достатній рівень якості згенерованих завдань для використання в освітньому процесі [3].

Актуальність дослідження визначається відсутністю доступних спеціалізованих веб-застосунків, які б забезпечували повний цикл роботи з силабусом: від завантаження документа до генерації, редагування та публічного розповсюдження тестових матеріалів серед студентів. Існуючі системи управління навчанням здебільшого не інтегрують функцію автоматизованої генерації завдань безпосередньо з вмісту силабусу, а спеціалізовані інструменти на основі штучного інтелекту не забезпечують збереження результатів та організацію доступу для студентів [4].

Мета дослідження полягає у розробці веб-застосунку для автоматизованої генерації тестових завдань та контрольних питань на основі силабусів дисциплін з

використанням великих мовних моделей, що забезпечує зберігання згенерованих матеріалів, їх редагування та проходження студентами в онлайн-режимі.

Завдання дослідження:

– Проаналізувати предметну галузь та існуючі рішення у сфері автоматизованої генерації тестових завдань.

– Сформулювати функціональні та нефункціональні вимоги до системи.

– Обґрунтувати вибір технологічного стеку та підходів до взаємодії з мовною моделлю.

– Спроєктувати архітектуру системи, модель бази даних та алгоритм генерації завдань.

– Реалізувати веб-застосунок на основі стеку React та FastAPI з інтеграцією Gemini API.

– Провести тестування розробленої системи.

Об'єктом дослідження є процес автоматизованої генерації контрольних матеріалів на основі структурованих навчальних документів у закладах вищої освіти.

Предметом дослідження є методи та засоби розробки веб-застосунку для генерації тестових завдань і контрольних питань на основі силабусів з використанням великих мовних моделей.

Методи дослідження: аналіз літературних джерел та існуючих програмних рішень у предметній галузі; порівняльний аналіз систем генерації тестових завдань за функціональними характеристиками; UML-моделювання для проєктування архітектури системи та опису сценаріїв взаємодії.

Практичне значення дослідження полягає у тому, що розроблена система може бути використана викладачами закладів вищої освіти для скорочення часу на підготовку контрольних матеріалів та організації онлайн-тестування студентів безпосередньо на основі змісту навчальної програми дисципліни.

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної галузі та огляд існуючих рішень. Другий розділ

присвячено аналізу вимог та методологічним основам системи. У третьому розділі здійснено моделювання та проєктування системи. Четвертий розділ містить опис програмної реалізації та результати тестування.

Загальний обсяг роботи становить 45 сторінок, робота містить 16 рисунків, 1 таблицю, 11 лістингів. Список використаних джерел містить 21 найменування.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Силабус як основа для формування контрольних матеріалів

Силабус є офіційним документом навчальної дисципліни, що визначає її структуру, зміст та організацію навчального процесу. У закладах вищої освіти силабус, як правило, містить назву та код дисципліни, перелік тем із їх описом, очікувані результати навчання, компетентності, які має набути здобувач, рекомендовану літературу, а також критерії та систему оцінювання [5]. Такий документ виконує подвійну функцію: з одного боку, він є інструментом планування для викладача, з іншого – орієнтиром для студента щодо очікувань і вимог курсу.

З точки зору формування контрольних матеріалів силабус є інформаційно насиченим вхідним документом. Перелік тем із їх описами та ключовими поняттями безпосередньо визначає предметну область, в межах якої мають бути сформовані тестові завдання. Результати навчання та компетентності задають рівень складності і тип питань – від відтворення фактів до застосування знань у практичних ситуаціях. Таким чином, між структурою силабусу і структурою контрольних матеріалів існує прямий змістовий зв'язок, який може бути формалізований і використаний для автоматизованої генерації завдань [3, с. 125].

Варто зазначити, що силабуси різних закладів і дисциплін суттєво відрізняються за форматом і деталізацією. Частина документів має чітку структуру з окремими розділами для кожного змістового блоку, тоді як інші є менш формалізованими. Це ускладнює автоматичний розбір документа і вимагає підходу, що не залежить від конкретного шаблону оформлення. У контексті розроблюваної системи силабус розглядається як текстовий документ довільного формату (PDF або DOCX), з якого витягується повний текст для подальшої передачі до мовної моделі. Такий підхід знімає залежність від структури конкретного документа і перекладає задачу змістового аналізу на рівень мовної моделі [2].

РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет математики та інформатики
Кафедра інформаційних технологій та моделювання

СИЛАБУС

Назва дисципліни	Основи інженерії програмного забезпечення
Освітня програма	Інженерія програмного забезпечення
Компонент освітньої програми	Обов'язковий
Загальна кількість кредитів та кількість годин для вивчення дисципліни	5 кредитів / 150 годин
Вид підсумкового контролю	екзамен
Мова викладання	українська
Викладач	к.п.н., доц. Петренко Сергій Вікторович
CV викладача на сайті кафедри	https://kitm.rshu.edu.ua/sklad-kafedru/petrenko-sergii-victorovych/
E-mail викладача	serhii.petrenko@rshu.edu.ua
Консультації	вівторок, четвер 14:00-15:00 https://meet.google.com/gmw-ayke-iua

Мета та завдання навчальної дисципліни

Мета дисципліни полягає у ознайомленні здобувачів вищої освіти з основними поняттями, засобами та методами інженерії програмного забезпечення, а також формування у знань основних принципів розробки ефективного програмного забезпечення та набуття навичок використання основних принципів реалізації етапів життєвого циклу ПЗ.

Рисунок 1.1 – Приклад структури силабусу навчальної дисципліни

Таким чином, силабус є достатньою інформаційною основою для формування контрольних матеріалів за умови, що система здатна коректно витягти з нього текстовий зміст і передати його моделі у вигляді, придатному для генерації змістовних питань.

1.2. Застосування мовних моделей для обробки та генерації текстового контенту

Великі мовні моделі (Large Language Models, LLM) є класом нейронних мереж, що навчаються на масштабних корпусах текстових даних і здатні виконувати широкий спектр задач обробки природної мови: генерацію тексту, його класифікацію, узагальнення, переклад та відповіді на запитання. Архітектурною основою переважної більшості сучасних LLM є трансформер – модель, запропонована у 2017 році, що використовує механізм уваги (attention mechanism) для встановлення залежностей між довільно віддаленими елементами вхідної послідовності [6]. Саме ця властивість дозволяє моделі враховувати широкий

контекст при обробці довгих документів, що є принциповим для роботи з текстами силабусів.

З практичної точки зору взаємодія з LLM відбувається через механізм промптингу – формування текстового запиту, який містить інструкцію для моделі та вхідні дані. Якість і формат відповіді моделі безпосередньо залежать від того, наскільки точно сформульовано промпт. У задачах генерації структурованого контенту, зокрема тестових завдань, промпт, як правило, містить опис очікуваного формату виводу, тип питань, їх кількість та текст, на основі якого має здійснюватись генерація [3, с. 140]. Такий підхід дозволяє отримувати відповідь у заздалегідь визначеній структурі, наприклад у форматі JSON, що спрощує подальшу програмну обробку результату.

Окремим напрямом є задача автоматичної генерації питань (Automatic Question Generation, AQG) – перетворення вхідного тексту на набір питань, що перевіряють розуміння його змісту. До появи LLM ця задача вирішувалась переважно за допомогою шаблонних методів та статистичних моделей, які демонстрували обмежені результати на різноманітних текстах. З появою трансформерних моделей якість автоматично згенерованих питань суттєво зросла: моделі здатні формувати питання, що відповідають змісту тексту, мають природне формулювання і охоплюють різні рівні складності [3, с. 148]. Дослідження підтверджують, що питання, згенеровані сучасними LLM, за оцінками експертів порівнянні з питаннями, складеними викладачами вручну, хоча і поступаються їм у частині педагогічної точності формулювань [6].

Характеристикою LLM з точки зору розроблюваної системи є здатність моделі працювати з текстом довільної структури без попередньої розмітки чи шаблонізації вхідного документа. На відміну від правилових систем, які вимагають суворо визначеного формату вхідних даних, мовна модель здатна витягувати змістову інформацію з тексту незалежно від його оформлення. Це є принциповим для роботи із силабусами, які в різних закладах мають різну структуру і рівень деталізації [2].

Взаємодія із зовнішніми LLM у прикладних системах, як правило, реалізується через програмний інтерфейс (API), що надається постачальником моделі. Застосунок формує запит із промптом і вхідними даними, надсилає його до API і отримує у відповідь згенерований текст. Така архітектура знімає з розробника необхідність розгортати і обслуговувати модель самостійно, що суттєво спрощує інтеграцію LLM у веб-застосунки [6]. Саме цей підхід застосовується у розроблюваній системі: текст силабусу передається до API мовної моделі разом із промптом, що задає тип, кількість і формат завдань, а відповідь у форматі JSON опрацьовується бекендом і зберігається у базі даних.

1.3. Аналіз існуючих систем генерації тестових завдань

На сьогодні існує ряд програмних рішень, що реалізують автоматизовану або напівавтоматизовану генерацію тестових завдань. Вони суттєво відрізняються за підходами до генерації, ступенем інтеграції з навчальними платформами та можливостями роботи з вхідними документами. Для аналізу обрано три системи, що є репрезентативними для різних підходів до вирішення цієї задачі: Quizizz AI, Moodle з плагіном Quiz Generator та Questgen.

Quizizz AI – це веб-платформа для створення інтерактивних тестів, що у 2023 році отримала вбудований модуль генерації питань на основі штучного інтелекту. Система дозволяє завантажити текстовий документ або вставити текст вручну, після чого автоматично генерує питання з вибором відповіді. Результат одразу доступний для проведення тестування серед студентів через посилання або QR-код [7]. Водночас система не орієнтована на роботу із силабусами як структурованим вхідним документом – вона сприймає будь-який текст без урахування його змістової організації. Крім того, можливості редагування згенерованих питань є обмеженими, а експорт у стандартні формати документів у безкоштовній версії недоступний.

Moodle – відкрита система управління навчанням, що підтримує створення тестів через вбудований модуль Quiz. Для автоматизованої генерації питань

використовуються сторонні плагіни, зокрема Quiz Generator, який дозволяє імпортувати питання з текстових файлів певного формату або генерувати їх на основі навчального матеріалу [4]. Moodle забезпечує повний цикл тестування: від створення питань до збору та аналізу результатів. Проте розгортання і налаштування системи вимагає значних технічних ресурсів, а генерація питань безпосередньо з силабусу як документа не передбачена – викладач має самостійно підготувати вхідний текст у визначеному форматі.

Questgen – спеціалізований інструмент для автоматичної генерації питань на основі тексту з використанням NLP-моделей [8]. Система підтримує кілька типів питань, зокрема питання з вибором відповіді та питання типу «правда/хиба», і надає API для інтеграції з іншими застосунками. Questgen орієнтований на розробників і дослідників, а не на кінцевого користувача – викладача: інтерфейс є мінімалістичним, відсутня можливість організації проходження тестів студентами, збереження результатів та керування створеними матеріалами у межах єдиного робочого середовища.

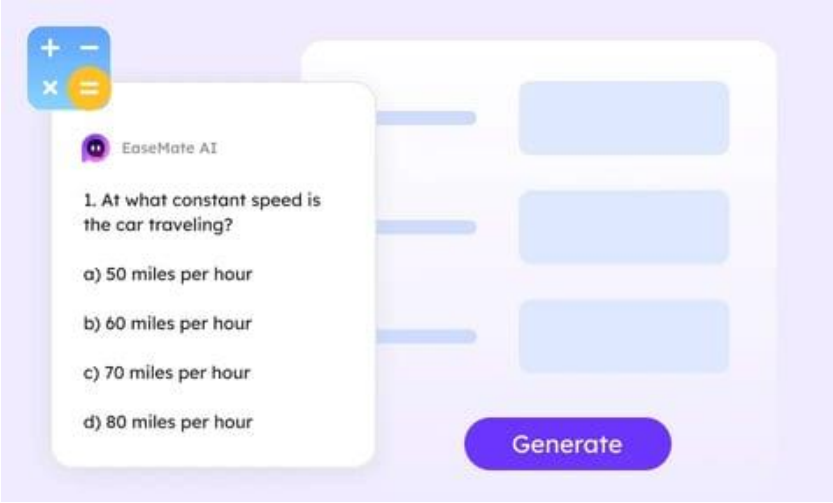


Рисунок 1.2 – Інтерфейс генерації питань у Quizizz AI

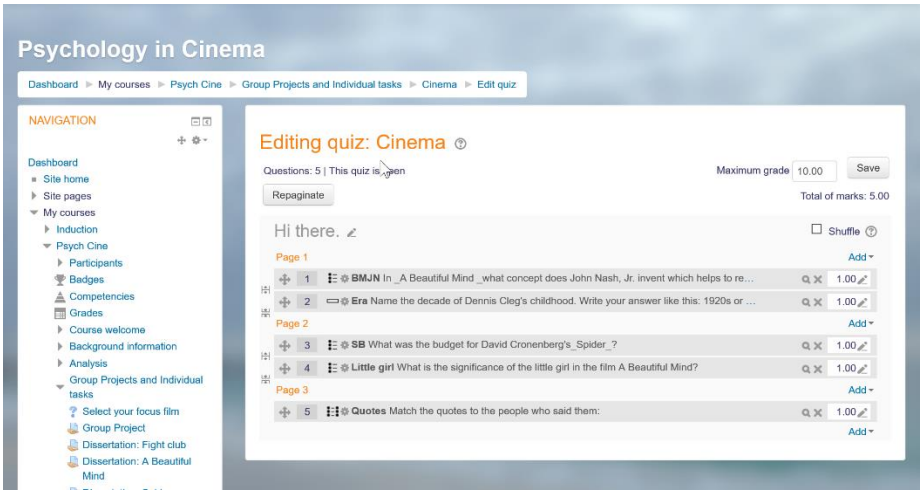


Рисунок 1.3 – Модуль Quiz у системі Moodle

Узагальнення результатів аналізу наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика існуючих систем генерації тестових завдань

Характеристика	**Quizizz AI**	**Moodle + Quiz Generator**	**Questgen**
Генерація на основі завантаженого документа	Так (текст/PDF)	Частково (визначений формат)	Так (текст)
Підтримка силабусу як вхідного документа	Ні	Ні	Ні
Типи питань	Multiple choice	Multiple choice, short answer, інші	Multiple choice, True/False
Проходження тесту студентами	Так	Так	Ні
Збереження та керування результатами	Так	Так	Ні
Ручне редагування згенерованих питань	Обмежено	Так	Обмежено
Експорт у DOCX/PDF	Платно	Так	Ні
Гостьовий режим без реєстрації	Ні	Ні	Ні
Розгортання	Хмарне	Локальне (сервер)	Хмарне/API

Аналіз показує, що жодна з розглянутих систем не забезпечує повного циклу роботи із силабусом як вхідним документом довільного формату – від його завантаження до організації проходження тесту студентами та перегляду результатів у межах єдиного інтерфейсу. Quizizz AI є найближчим за функціональністю рішенням, проте не орієнтований на силабуси і має обмеження у безкоштовній версії. Moodle забезпечує повний цикл тестування, але вимагає значних ресурсів для розгортання і не підтримує генерацію безпосередньо з документа. Questgen реалізує генерацію питань, але є інструментом для розробників без функцій організації тестування. Це обґрунтовує розробку спеціалізованого веб-застосунку, що поєднує генерацію на основі силабусу, редагування результатів, організацію проходження тестів студентами та перегляд детальної історії відповідей.

РОЗДІЛ 2
АНАЛІЗ ВИМОГ ТА МЕТОДОЛОГІЧНІ ОСНОВИ СИСТЕМИ

2.1. Функціональні та нефункціональні вимоги до системи

Формулювання вимог до системи є основою для прийняття проєктних рішень на наступних етапах розробки. Вимоги поділяються на функціональні, що визначають що система має виконувати, та нефункціональні, що визначають умови і обмеження, за яких вона має це виконувати [9].

Функціональні вимоги сформульовані на основі аналізу предметної галузі та виявлених недоліків існуючих рішень, розглянутих у попередньому підрозділі.

Модуль роботи з силабусами:

– ФВ-01. Система має забезпечувати завантаження силабусу у форматах PDF, DOCX та TXT.

– ФВ-02. Система має витягувати текстовий вміст із завантаженого документа для подальшої передачі до мовної моделі.

Модуль генерації:

– ФВ-03. Система має надавати можливість налаштування параметрів генерації: тип матеріалу (тестові завдання, контрольні питання або обидва), кількість одиниць кожного типу.

– ФВ-04. Система має генерувати тестові завдання двох типів: з вибором відповіді (multiple choice, 4 варіанти) та з короткою відповіддю (short answer, 1–3 слова).

– ФВ-05. Система має зберігати згенерований матеріал у базі даних для зареєстрованого користувача одразу після генерації.

Модуль керування матеріалами:

– ФВ-06. Система має надавати зареєстрованому викладачу список створених тестів та наборів контрольних питань.

– ФВ-07. Система має забезпечувати редагування тестових завдань: зміну тексту питання, варіантів відповідей та правильної відповіді, видалення та додавання питань вручну.

– ФВ-08. Система має забезпечувати видалення тесту або набору контрольних питань.

– ФВ-09. Система має надавати можливість експорту тесту або набору контрольних питань у форматах DOCX та PDF у двох варіантах: з відповідями та без відповідей.

Модуль проходження тесту:

– ФВ-10. Система має надавати публічне посилання на тест для зареєстрованого викладача.

– ФВ-11. Система має забезпечувати проходження тесту студентом без реєстрації з введенням імені перед початком.

– ФВ-12. Система має відображати всі питання тесту одночасно на одній сторінці.

– ФВ-13. Система має автоматично перевіряти відповіді після відправки: для multiple choice – точне співпадіння, для short answer – порівняння після trim та приведення до нижнього регістру.

– ФВ-14. Система має відображати студенту результат одразу після відправки відповідей.

Модуль результатів:

– ФВ-15. Система має надавати викладачу список студентів, що пройшли тест, з можливістю пошуку за іменем, фільтрації за датою та сортування за балом, датою та іменем.

– ФВ-16. Система має надавати детальний перегляд відповідей конкретного студента із зазначенням відповіді студента, правильної відповіді та виставленого балу.

– ФВ-17. Система має надавати викладачу можливість ручного коригування балу за окреме питання з автоматичним перерахунком загального балу.

– ФВ-18. Система має надавати можливість видалення окремого результату тестування.

Модуль авторизації:

– ФВ-19. Система має підтримувати реєстрацію та автентифікацію викладача за електронною поштою та паролем.

– ФВ-20. Система має підтримувати гостьовий режим: генерація, доступна без реєстрації редагування, а експорт, збереження в базі даних та публічне посилання – недоступні.

Нефункціональні вимоги визначають якісні характеристики системи незалежно від конкретного технологічного стеку, який буде обґрунтовано у наступному підрозділі.

– НВ-01. Веб-застосунок має коректно відображатись і функціонувати у актуальних версіях браузерів Chrome, Firefox та Edge.

– НВ-02. Інтерфейс має забезпечувати виконання основних операцій без необхідності ознайомлення з документацією.

– НВ-03. Час відповіді сервера на запити, що не включають звернення до зовнішнього API, не має перевищувати 1 секунду.

– НВ-04. Система має коректно обробляти помилки зовнішнього API (таймаут, невалідна відповідь) і повертати користувачу інформативне повідомлення про помилку.

– НВ-05. Паролі користувачів мають зберігатись у базі даних у хешованому вигляді.

– НВ-06. Автентифікація має реалізовуватись на основі токенів з обмеженим терміном дії.

– НВ-07. Система має зберігати цілісність даних при одночасному проходженні тесту кількома студентами.

2.2. Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку здійснювався на основі сформульованих вимог до системи, зокрема необхідності динамічного рендерингу змінної кількості елементів, інтеграції із зовнішнім API мовної моделі, реляційного зберігання даних та розділення клієнтської і серверної частин.

На стороні клієнта обрано бібліотеку React. Одним із визначальних факторів є підхід до рендерингу інтерфейсу: кількість питань у відповіді мовної моделі є змінною і визначається параметрами генерації. React забезпечує декларативний рендеринг списків через механізм відображення масивів (.map()), що дозволяє відображати довільну кількість компонентів без зміни коду інтерфейсу [10]. Це є прямою відповіддю на вимогу ФВ-03 щодо налаштування кількості питань користувачем. Додатково React забезпечує компонентну організацію коду, що спрощує повторне використання елементів інтерфейсу – зокрема, компонент відображення одного питання використовується як на сторінці перегляду тесту, так і на сторінці його проходження студентом.

На стороні сервера обрано мову програмування Python з фреймворком FastAPI. Python має розгалужену екосистему бібліотек для роботи з документами різних форматів: pdfplumber для витягування тексту з PDF та python-docx для роботи з DOCX, що безпосередньо забезпечує виконання вимог ФВ-01 та ФВ-02. FastAPI надає декларативний опис ендпоінтів з автоматичною валідацією даних через Pydantic та автоматично генерує документацію у форматі OpenAPI, що спрощує розробку і налагодження API [11]. Асинхронна природа FastAPI дозволяє не блокувати сервер під час очікування відповіді від зовнішнього Gemini API, що відповідає вимозі НВ-03.

Як систему управління базою даних обрано MySQL. Дані системи – тести, питання, результати тестування, відповіді студентів – мають чітко визначену реляційну структуру зі зв'язками між сутностями, що робить реляційну СКБД природним вибором для їх зберігання. MySQL є зрілим рішенням з добре документованою поведінкою при одночасних записах [12], що відповідає вимозі

НВ-07 щодо цілісності даних при паралельному проходженні тесту кількома студентами.

Для генерації тестових завдань і контрольних питань обрано Gemini API від Google. Модель gemini-2.5-flash підтримує довгий контекст, що дозволяє передавати повний текст силабусу в одному запиті без необхідності його розбиття на частини. API надає безкоштовний рівень доступу з лімітом 5 запитів на хвилину, що є достатнім для прототипу та академічного використання [13]. Належність API до інфраструктури Google забезпечує стабільність сервісу та наявність офіційної документації і клієнтських бібліотек для Python.

2.3. Підходи до взаємодії з мовною моделлю та формування запитів

Взаємодія з мовною моделлю у розроблюваній системі зводиться до формування текстового запиту – промпту – що містить інструкцію для моделі та вхідні дані, і подальшої програмної обробки відповіді. Від того, наскільки точно сформульовано промпт, залежить як якість згенерованих питань, так і можливість автоматичного розбору відповіді без додаткової обробки [3, с. 140].

Принциповим рішенням є вимога до моделі повертати відповідь у форматі JSON, а не у вільній текстовій формі. Мовні моделі за замовчуванням генерують текст, орієнтований на читання людиною, – з поясненнями, нумерацією, заголовками тощо. Такий формат не придатний для програмної обробки без складного парсингу, який є нестійким до варіацій у форматуванні відповіді. Натомість явна вимога повернути строго визначену JSON-структуру дозволяє обробляти відповідь стандартними засобами мови програмування без жодної додаткової обробки тексту [14]. Gemini API підтримує режим примусового виводу у форматі JSON через параметр response_mime_type: "application/json", що додатково знижує ймовірність отримання невалідної відповіді.

Промпт у системі складається з двох частин. Перша частина – системна інструкція – є фіксованою і визначає роль моделі, вимоги до формату відповіді та структуру очікуваного JSON. Друга частина – користувацький запит – формується

динамічно на основі параметрів, що надає викладач: тип матеріалу для генерації, кількість питань кожного типу та текст силабусу. Розділення промпту на фіксовану і змінну частини спрощує підтримку і модифікацію логіки генерації без зміни основного коду застосунку.

Очікувана структура JSON-відповіді від моделі має такий вигляд:

```
{
  "tests": [
    {
      "type": "multiple_choice",
      "question": "Текст питання",
      "options": ["A", "B", "C", "D"],
      "answer": "A"
    }
  ],
  "questions": [
    {
      "type": "short_answer",
      "question": "Текст питання",
      "answer": "відповідь"
    }
  ]
}
```

Поля tests та questions є масивами, кожен з яких може бути порожнім, якщо відповідний тип матеріалу не було запитано. Це дозволяє обробляти відповідь єдиним способом незалежно від того, що саме генерувалось. Після отримання відповіді від API бекенд виконує десеріалізацію JSON, валідує структуру через Pydantic-схеми і або зберігає результат у базі даних (для зареєстрованого користувача), або повертає безпосередньо у відповіді на запит (для гостя).

Окремою складовою є обробка помилок взаємодії з API. Можливі сценарії включають перевищення ліміту запитів, таймаут з'єднання та отримання відповіді, що не відповідає очікуваній JSON-структурі. У кожному з цих випадків система має повертати клієнту інформативне повідомлення про помилку відповідно до вимоги НВ-04, а не технічний стек-трейс.

РОЗДІЛ 3
МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ

3.1. Архітектура системи

Архітектура розроблюваної системи побудована за принципом клієнт-серверного розділення з виділенням зовнішнього сервісу мовної моделі як окремого компонента. Клієнтська частина реалізована у вигляді односторінкового застосунку (Single Page Application, SPA) на React, серверна – у вигляді REST API на FastAPI, база даних MySQL розгортається на тому самому сервері що і бекенд. Взаємодія між клієнтом і сервером здійснюється виключно через HTTP-запити у форматі JSON, що забезпечує незалежність клієнтської і серверної частин [11]. Для моделювання архітектури системи використано мову UML (Unified Modeling Language), яка є стандартом для опису структури та поведінки програмних систем [15].

Діаграма варіантів використання, що відображає акторів системи та їх сценарії взаємодії, наведена у додатку А (рисунок А.1). Система має трьох акторів: викладач-гість, зареєстрований викладач та студент. Викладач-гість має доступ до генерації матеріалів на основі завантаженого силабусу та їх експорту у форматах DOCX і PDF. Зареєстрований викладач додатково має доступ до збереження згенерованих матеріалів у базі даних, їх редагування, отримання публічного посилання на тест та перегляду результатів тестування. Студент взаємодіє із системою виключно через публічне посилання на тест без необхідності реєстрації.

Діаграма компонентів відображає внутрішню структуру системи та зв'язки між її складовими (рисунок 3.1). Клієнтська частина складається з компонентів сторінок та модуля взаємодії з API (Axios). На стороні сервера виділяються такі модулі: маршрутизатор (router), що приймає HTTP-запити і передає їх відповідним обробникам; модуль авторизації, що відповідає за перевірку JWT-токенів; модуль парсингу документів, що витягує текст із завантажених файлів; модуль генерації, що формує промпт і взаємодіє з Gemini API; модуль експорту, що формує DOCX і

PDF файли; модуль роботи з базою даних через ORM. Зовнішніми компонентами є Gemini API та база даних MySQL.

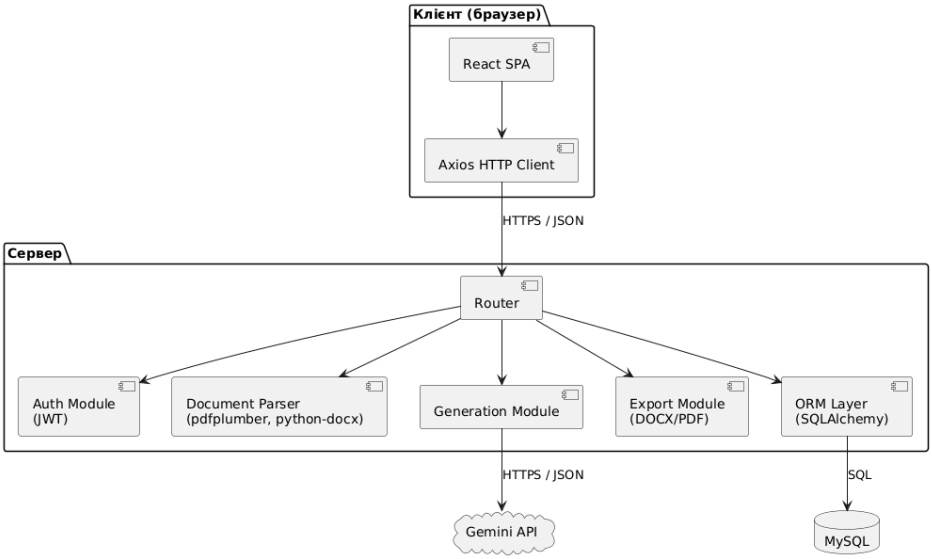


Рисунок 3.1 – Діаграма компонентів системи

Діаграма розгортання описує фізичне розміщення компонентів системи (рисунок 3.2). Браузер користувача виконує React SPA і звертається до серверного вузла через HTTPS. На серверному вузлі розгорнуто FastAPI-застосунок та екземпляр MySQL. FastAPI звертається до зовнішнього вузла Gemini API через HTTPS для виконання запитів генерації. Така конфігурація є стандартною для веб-застосунків подібного класу і не вимагає складної інфраструктури розгортання [12].

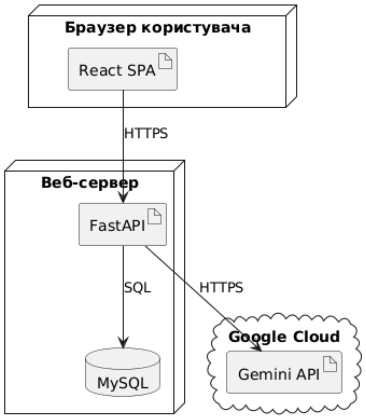


Рисунок 3.2 – Діаграма розгортання системи

Діаграма діяльності відображає загальний процес проходження тесту студентом (рисунок 3.3). Студент переходить за публічним посиланням, вводить ім'я і отримує сторінку з усіма питаннями тесту одночасно. Після заповнення відповідей і відправки форми сервер виконує автоматичну перевірку: для питань з вибором відповіді застосовується точне співпадіння, для питань з короткою відповіддю – порівняння після нормалізації рядка. Результат зберігається у базі даних і відображається студенту на окремій сторінці.

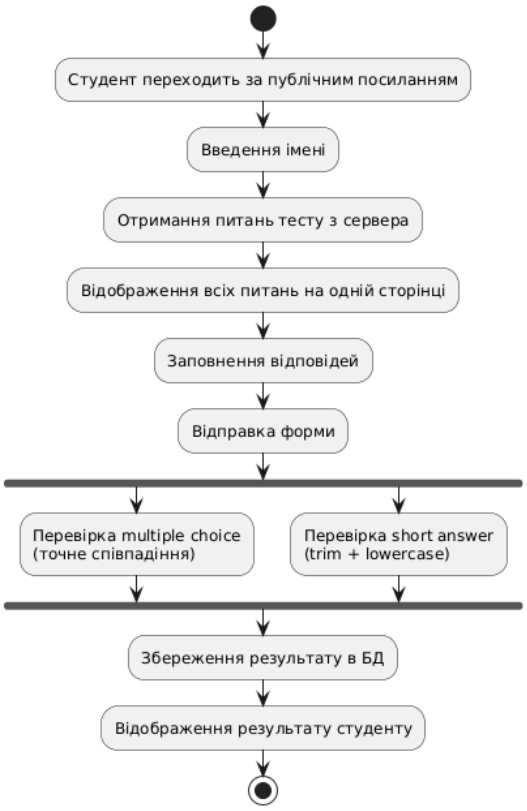


Рисунок 3.3 – Діаграма діяльності процесу проходження тесту

Описана архітектура забезпечує чітке розділення відповідальності між компонентами системи. Клієнтська частина відповідає виключно за відображення даних і взаємодію з користувачем, серверна – за бізнес-логіку, взаємодію з зовнішнім API та збереження даних. Така організація спрощує незалежне тестування компонентів і дозволяє масштабувати окремі частини системи без змін в інших.

3.2. Алгоритм обробки силабусу та генерації завдань

Процес генерації тестових завдань у розроблюваній системі складається з двох послідовних етапів: обробки завантаженого документа силабусу і безпосередньої генерації матеріалів через звернення до Gemini API. Кожен з етапів реалізований як окремий модуль на стороні сервера, що відповідає архітектурі, описаній у попередньому підрозділі.

На першому етапі клієнт надсилає файл силабусу та параметри генерації на бекенд через HTTP POST запит. Бекенд визначає формат файлу за його розширенням і передає його відповідному парсеру: для PDF використовується бібліотека pdfplumber, для DOCX – python-docx, для TXT – стандартні засоби читання файлів. Результатом парсингу є рядок із повним текстовим вмістом документа. Якщо файл пошкоджений або має непідтримуваний формат, модуль парсингу повертає виняток, який перехоплюється на рівні маршрутизатора і повертається клієнту у вигляді інформативного повідомлення про помилку відповідно до вимоги НВ-04 [16].

На другому етапі модуль генерації формує промпт на основі витягнутого тексту та параметрів запиту. Алгоритм формування промпту наведено на рисунку 3.4. Системна частина промпту є фіксованою і містить інструкцію повертати відповідь строго у форматі JSON із визначеною структурою. Динамічна частина формується залежно від прапорців generate_tests та generate_questions і відповідних значень кількості: якщо обидва прапорці активні, модель отримує інструкцію згенерувати обидва масиви; якщо активний лише один – інструкція стосується лише відповідного типу, а інший масив має бути порожнім. Після цього до промпту додається повний текст силабусу [14].

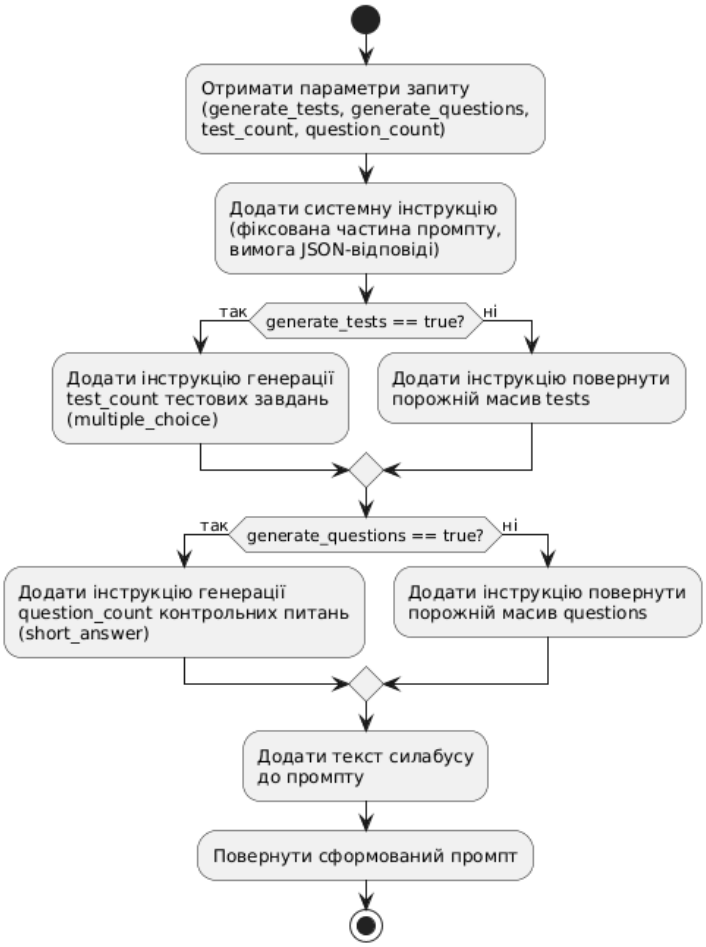


Рисунок 3.4 – Діаграма діяльності алгоритму формування промпту

Сформований промпт передається до Gemini API. Послідовність взаємодії компонентів у процесі генерації відображено на діаграмі послідовності (рисунок 3.5). Після отримання відповіді від API бекенд виконує десеріалізацію JSON та валідацію структури через Pydantic-схеми. У разі невідповідності структури очікуваній схемі система повертає клієнту повідомлення про помилку без збереження неповних даних. За умови успішної валідації подальша логіка залежить від того, чи є користувач зареєстрованим: для зареєстрованого викладача дані зберігаються у базі даних і клієнту повертається UUID створеного запису; для гостя дані повертаються безпосередньо у тілі відповіді без збереження [2].

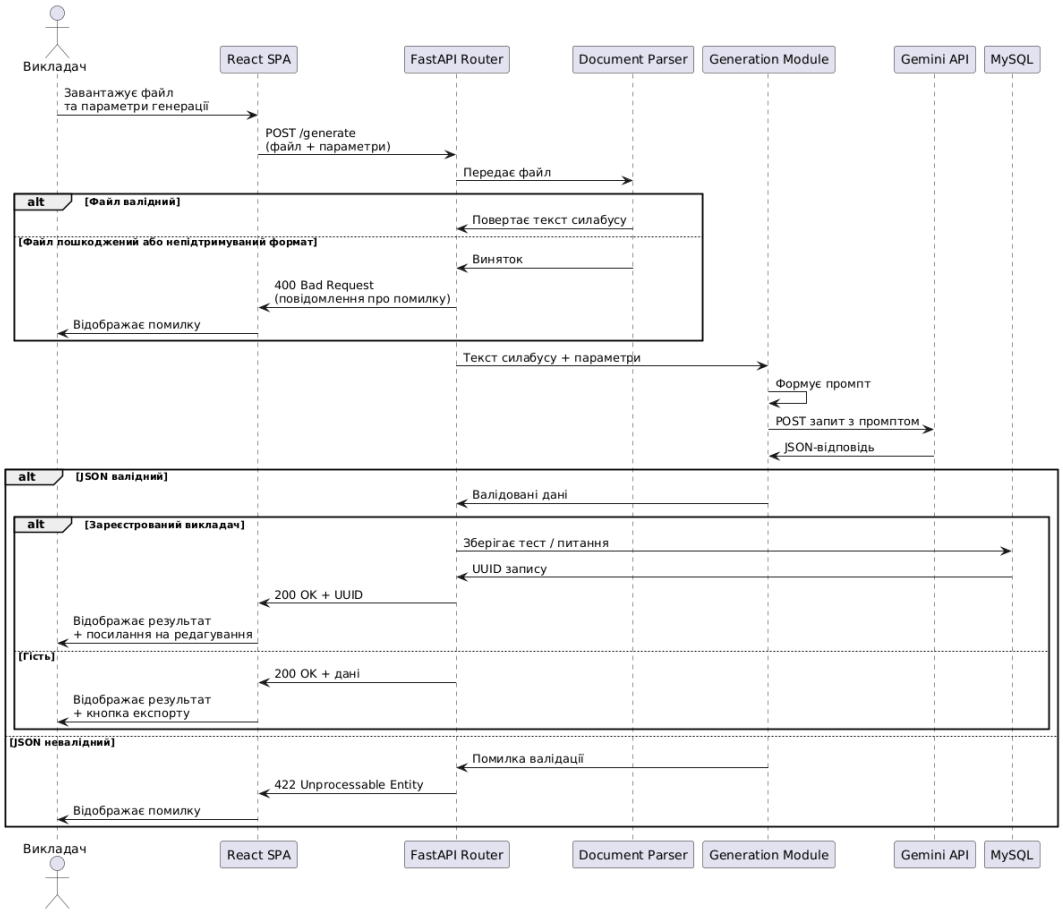


Рисунок 3.5 – Діаграма послідовності процесу генерації завдань

Описаний алгоритм забезпечує обробку силабусів довільного формату і структури без попередньої розмітки документа, що є прямим наслідком делегування задачі змістового аналізу мовній моделі. Розділення процесу на етап парсингу і етап генерації дозволяє незалежно обробляти помилки кожного з них і за потреби замінювати окремі компоненти без зміни загальної логіки.

3.3. Проєктування бази даних

База даних системи реалізована на основі реляційної моделі у СКБД MySQL. Схема містить сім таблиць, що відображають основні сутності предметної галузі та зв'язки між ними. При проєктуванні схеми дотримано вимог третьої нормальної форми (3NF): кожна таблиця містить лише атрибути, що функціонально залежать від первинного ключа і не залежать між собою транзитивно [12].

Таблиця teachers зберігає облікові дані зареєстрованих викладачів. Первинним ключем є автоінкрементний id. Поле hashed_password містить хеш пароля відповідно до вимоги НВ-05. Відповідність 3NF [17] забезпечується тим, що всі атрибути – email, hashed_password, created_at – залежать виключно від id і не мають транзитивних залежностей між собою.

Таблиці tests та question_sets є структурно аналогічними і зберігають відповідно тести та набори контрольних питань. Первинним ключем обох таблиць є uuid типу CHAR(36), що дозволяє формувати публічні посилання без розкриття внутрішніх числових ідентифікаторів. Поле teacher_id є зовнішнім ключем з посиланням на teachers(id) зі стратегією ON DELETE SET NULL – при видаленні викладача його матеріали зберігаються у базі даних з teacher_id = NULL, що відповідає логіці гостьового режиму.

Таблиця test_questions зберігає питання тесту з прив'язкою до tests через зовнішній ключ test_uuid зі стратегією ON DELETE CASCADE. Поле type має тип ENUM('multiple_choice', 'short_answer'), що обмежує допустимі значення на рівні схеми БД. Поле options має тип JSON і заповнюється лише для питань типу multiple_choice – для short_answer воно має значення NULL. Винесення варіантів відповідей у JSON-поле, а не в окрему таблицю, обґрунтовується тим, що варіанти є атомарним атрибутом питання і ніколи не використовуються незалежно від нього, що не порушує принципів нормалізації [12]. Таблиця short_questions аналогічно зберігає контрольні питання з прив'язкою до question_sets.

Таблиця test_results зберігає результат проходження тесту конкретним студентом. Поля total_score та max_score мають тип DECIMAL(6,2) для коректного збереження дробових значень балів після ручного коригування викладачем. Зв'язок з tests реалізовано через test_uuid зі стратегією ON DELETE CASCADE – при видаленні тесту всі його результати також видаляються.

Таблиця student_answers зберігає відповідь студента на кожне окреме питання з прив'язкою до результату через result_id та до питання через question_id. Поле score містить бал, виставлений системою автоматично, а поле manual_score – бал, скоригований викладачем вручну, і має значення NULL якщо коригування не

відбувалось. Фактичний бал за питання визначається як COALESCE(manual_score, score). Відповідність 3NF забезпечується тим, що жоден атрибут таблиці не залежить від іншого атрибута, що не є первинним ключем.

ER-діаграма [18] схеми бази даних наведена на рисунку 3.6.

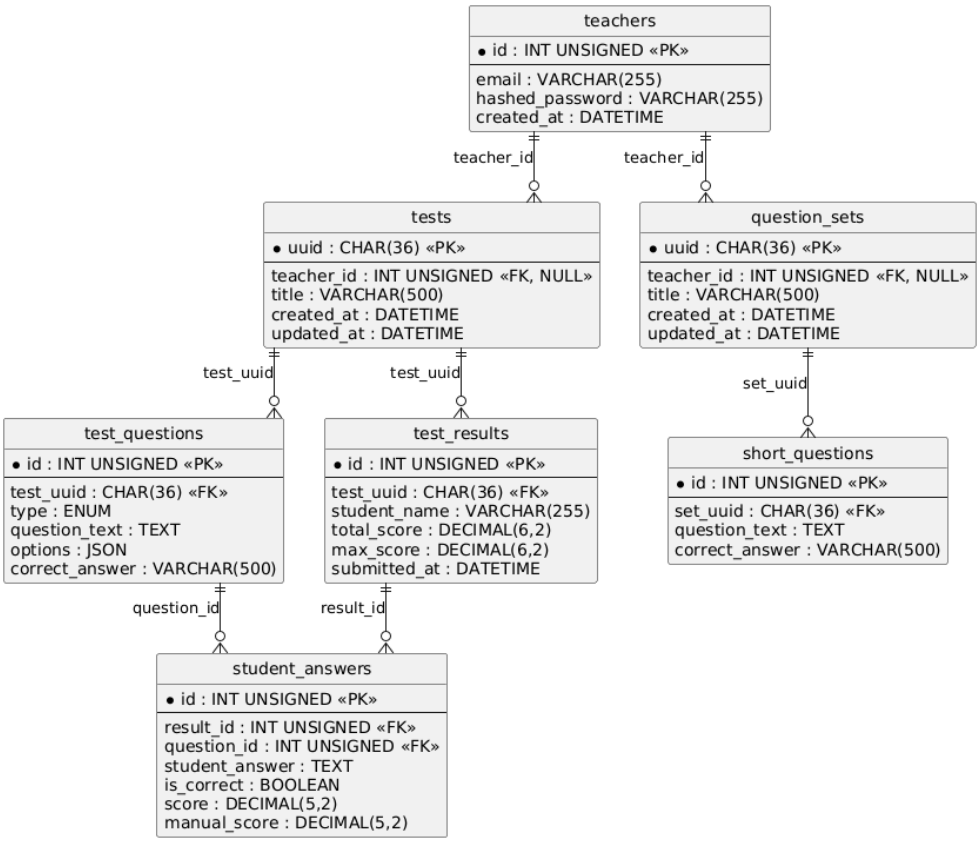


Рисунок 3.6 – ER-діаграма бази даних системи

Наведена схема забезпечує зберігання всіх сутностей системи з чітко визначеними зв'язками між ними. Каскадне видалення на рівні зовнішніх ключів гарантує цілісність даних без необхідності додаткової логіки на рівні застосунку: видалення тесту автоматично видаляє його питання та результати тестування, видалення результату – відповіді студента. Стратегія ON DELETE SET NULL для зв'язку між викладачем і його матеріалами дозволяє зберігати створені тести та набори питань після видалення облікового запису викладача.

РОЗДІЛ 4
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1. Реалізація модуля завантаження та парсингу силабусів

Модуль парсингу силабусів реалізовано у вигляді двох файлів: сервісного модуля parser.py, що містить логіку витягування тексту з документів різних форматів, та маршрутизатора parse.py, що визначає ендпоінт для прийому файлів від клієнта.

Сервісний модуль реалізує функцію parse_file, що приймає вміст файлу у вигляді байтів та його ім'я, визначає формат за розширенням і делегує обробку відповідній внутрішній функції (лістинг 4.1). Для PDF використовується бібліотека pdfplumber, яка послідовно обходить сторінки документа і витягує текст кожної з них. Для DOCX використовується бібліотека python-docx, що обходить абзаци документа і об'єднує непорожні з них у рядок. Для TXT вміст декодується безпосередньо з байтів з параметром errors="ignore" для уникнення винятків при некоректному кодуванні. Якщо розширення файлу не належить до підтримуваних, функція генерує виняток ValueError з інформативним повідомленням.

Лістинг 4.1 – Сервісний модуль парсингу документів

```
    import io
    import pdfplumber
    import docx

    def parse_file(content: bytes, filename: str) -> str:
        ext = filename.rsplit(".", 1)[-1].lower() if "." in filename
else ""
        if ext == "pdf":
            return _parse_pdf(content)
        elif ext == "docx":
            return _parse_docx(content)
        elif ext == "txt":
            return content.decode("utf-8", errors="ignore")
        else:
            raise ValueError(
```

```
                f"Unsupported format: .{ext}. Supported: PDF, DOCX,
TXT"
            )

    def _parse_pdf(content: bytes) -> str:
        parts = []
        with pdfplumber.open(io.BytesIO(content)) as pdf:
            for page in pdf.pages:
                text = page.extract_text()
                if text:
                    parts.append(text)
        return "\n".join(parts)

    def _parse_docx(content: bytes) -> str:
        doc = docx.Document(io.BytesIO(content))
        return   "\n".join(p.text   for   p   in   doc.paragraphs   if
p.text.strip())
```

Маршрутизатор parse.py визначає ендпоінт POST /parse, що приймає файл через multipart/form-data і повертає витягнутий текст разом із кількістю символів (лістинг 4.2). До виклику сервісного модуля виконується двоетапна валідація: перевірка розширення файлу на належність до допустимих форматів та перевірка розміру файлу – максимально допустимий розмір становить 10 МБ. Результатом є Pydantic-модель ParseResponse з полями text та char_count. Винятки типу ValueError з сервісного модуля перехоплюються і повертаються клієнту як HTTP 400, непередбачувані помилки парсингу – як HTTP 422 з повідомленням про пошкоджений або захищений паролем файл, що відповідає вимозі НВ-04.

Лістинг 4.2 – Ендпоінт POST /parse

```
    from fastapi import APIRouter, UploadFile, File, HTTPException
    from pydantic import BaseModel
    from ..services.parser import parse_file

    router = APIRouter(prefix="/parse", tags=["parse"])

    ALLOWED_EXTENSIONS = {"pdf", "docx", "txt"}
    MAX_FILE_SIZE = 10 * 1024 * 1024  # 10 MB

    class ParseResponse(BaseModel):
```

```
        text: str
        char_count: int

    @router.post("", response_model=ParseResponse)
    async def parse_syllabus(file: UploadFile = File(...)):
        if not file.filename:
            raise HTTPException(400, "No file provided")
        ext = file.filename.rsplit(".", 1)[-1].lower() \
            if "." in file.filename else ""
        if ext not in ALLOWED_EXTENSIONS:
            raise HTTPException(
                400,
                f"Unsupported format. Allowed: "
                f"{', '.join(ALLOWED_EXTENSIONS).upper()}"
            )
        content = await file.read()
        if len(content) > MAX_FILE_SIZE:
            raise HTTPException(400, "File too large (max 10 MB)")
        try:
            text = parse_file(content, file.filename)
        except ValueError as e:
            raise HTTPException(400, str(e))
        except Exception:
            raise HTTPException(
                422,
                "Could not parse the file. "
                "It may be corrupted or password-protected."
            )
        if not text.strip():
            raise HTTPException(
                422, "No text could be extracted from the file"
            )
        return ParseResponse(text=text, char_count=len(text))
```

Реалізований модуль забезпечує виконання вимог ФВ-01 та ФВ-02: підтримку трьох форматів вхідних документів та витягування повного текстового вмісту силабусу для подальшої передачі до модуля генерації.

4.2. Реалізація модуля взаємодії з ШІ та генерації завдань

Модуль генерації завдань реалізовано у трьох файлах: сервісний модуль gemini.py відповідає за формування промпту та взаємодію з Gemini API, файл

generate.py у пакеті схем визначає Pydantic-моделі запиту і відповіді, маршрутизатор generate.py реалізує ендпоінт POST /generate та логіку збереження результатів у базі даних.

Центральним елементом сервісного модуля є шаблон промпту _PROMPT_TEMPLATE (лістинг 4.3). Він складається з фіксованої системної частини та двох змінних полів: {text} – текст силабусу і {task} – динамічно сформована інструкція щодо типу і кількості питань. Системна частина містить чотири групи інструкцій для моделі: вимогу до мови відповіді – питання мають бути сформовані тією мовою, якою написано силабус; вимогу до змісту – питання мають перевіряти знання предметної галузі, а не структури силабусу як документа, з наведенням прикладів коректного і некоректного формулювання; вимоги до формату кожного типу питань; вимогу повертати виключно валідний JSON без жодного додаткового тексту [14].

Лістинг 4.3 – Шаблон промпту для генерації завдань

```
    You are an educational content generator. Based on the syllabus
text
    below, generate questions that test students' knowledge of the
    SUBJECT MATTER described in the syllabus.

    STRICT RULES:
    - LANGUAGE: ALL questions and answers MUST be written in the SAME
      language as the syllabus text. If the syllabus is in Ukrainian
—
      write in Ukrainian. If in English — write in English.
    - Questions must test knowledge of the TOPIC CONTENT, not
knowledge
      of the syllabus structure. NEVER ask "what is mentioned in topic
X".
    - Good example: "Що таке рефакторинг?" — tests knowledge of
concept.
    - Bad example: "Який процес згадується у Темі 3?" — tests the
syllabus.
    - multiple_choice: exactly 4 options formatted as
      ["A. text", "B. text", "C. text", "D. text"],
      answer is one of "A", "B", "C", "D"
    - short_answer: answer must be 1-3 words maximum
```

```
    - Return ONLY valid JSON — no markdown, no explanation, no code
blocks

    REQUIRED JSON FORMAT:
    {
      "tests": [
        {"type": "multiple_choice", "question": "...",
         "options": ["A. ...", "B. ...", "C. ...", "D. ..."],
"answer": "A"}
      ],
      "questions": [
        {"type": "short_answer", "question": "...", "answer": "..."}
      ]
    }

    SYLLABUS:
    {text}

    TASK: {task}
    Return ONLY the JSON object.
```

Функція generate_content формує динамічну частину промпту залежно від параметрів запиту, передає його до Gemini API і обробляє відповідь (лістинг 4.4). Динамічна частина {task} формується як об'єднання інструкцій: якщо активний прапорець generate_tests – додається інструкція згенерувати test_count питань типу multiple_choice, якщо generate_questions – інструкція згенерувати question_count питань типу short_answer. Після отримання відповіді від API виконується постобробка: регулярні вирази видаляють можливі markdown-огорожі ```json та ```, які модель може додати попри інструкцію, після чого рядок десеріалізується через json.loads [19]. Якщо відповідний тип питань не запитувався, відповідний масив примусово встановлюється порожнім незалежно від вмісту відповіді моделі.

Лістинг 4.4 – Функція генерації контенту через Gemini API

```
    def generate_content(
        text: str,
        test_count: int = 10,
        question_count: int = 5,
        generate_tests: bool = True,
        generate_questions: bool = True,
```

```
    ) -> dict:
        parts = []
        if generate_tests:
            parts.append(
                f"Generate {test_count} multiple_choice questions"
                f" for the 'tests' array"
            )
        if generate_questions:
            parts.append(
                f"Generate {question_count} short_answer questions"
                f" for the 'questions' array"
            )

        prompt = _PROMPT_TEMPLATE.format(
            text=text, task="; ".join(parts)
        )

        response = _client.models.generate_content(
            model=settings.gemini_model,
            contents=prompt,
        )

        raw = response.text.strip()
        raw = re.sub(r"^```(?:json)?\s*", "", raw)
        raw = re.sub(r"\s*```$", "", raw)

        data = json.loads(raw)

        if not generate_tests:
            data["tests"] = []
        if not generate_questions:
            data["questions"] = []

        return data
```

Pydantic-схеми у файлі generate.py визначають структуру запиту та відповіді ендпоінту (лістинг 4.5). Модель GenerateRequest містить текст силабусу та параметри генерації з визначеними значеннями за замовчуванням: 10 тестових завдань і 5 контрольних питань з увімкненою генерацією обох типів. Моделі GeneratedTestQuestion та GeneratedShortQuestion описують структуру окремого питання кожного типу, де поле options є необов'язковим і присутнє лише для multiple_choice. Валідація вхідних даних через Pydantic виконується автоматично

на рівні FastAPI до виклику функції-обробника, що знімає необхідність ручних перевірок у коді маршрутизатора [20].

Лістинг 4.5 – Pydantic-схеми запиту і відповіді

```
from pydantic import BaseModel
from typing import Optional, List

class GenerateRequest(BaseModel):
    text: str
    test_count: int = 10
    question_count: int = 5
    generate_tests: bool = True
    generate_questions: bool = True

class GeneratedTestQuestion(BaseModel):
    type: str
    question: str
    options: Optional[List[str]] = None
    answer: str

class GeneratedShortQuestion(BaseModel):
    type: str
    question: str
    answer: str

class GenerateResponse(BaseModel):
    tests: List[GeneratedTestQuestion] = []
    questions: List[GeneratedShortQuestion] = []
```

Маршрутизатор generate.py реалізує ендпоінт POST /generate (лістинг 4.6). Залежність get_optional_teacher повертає об'єкт викладача якщо запит містить валідний JWT-токен, або None якщо токен відсутній – це забезпечує розгалуження між зареєстрованим користувачем і гостем в межах одного ендпоінту без дублювання логіки [21]. Після отримання даних від сервісного модуля логіка збереження виконується лише якщо teacher is not None. Назва тесту формується функцією _title_from_text, що витягує перший непорожній рядок тексту силабусу довжиною понад 4 символи і обрізає його до 200 символів. Тест і набір контрольних питань зберігаються як окремі записи з власними UUID відповідно до вимоги ФВ-

05. Виклик db.flush() після додавання батьківського запису забезпечує наявність UUID у базі даних до збереження дочірніх питань без завершення транзакції, після чого єдиний виклик db.commit() фіксує всі зміни атомарно.

Лістинг 4.6 – Ендпоінт POST /generate

```
    import uuid as uuid_lib
    from fastapi import APIRouter, Depends, HTTPException
    from sqlalchemy.orm import Session
    from ..database import get_db
    from ..dependencies import get_optional_teacher
    from .. import models
    from ..schemas.generate import GenerateRequest, GenerateResponse
    from ..services import gemini

    router = APIRouter(prefix="/generate", tags=["generate"])

    def _title_from_text(text: str) -> str:
        for line in text.splitlines():
            line = line.strip()
            if len(line) > 4:
                return line[:200]
        return text.strip()[:200]

    @router.post("", response_model=GenerateResponse)
    def generate(
        payload: GenerateRequest,
        db: Session = Depends(get_db),
        teacher:        models.Teacher        |        None        =
Depends(get_optional_teacher),
    ):
        try:
            data = gemini.generate_content(
                text=payload.text,
                test_count=payload.test_count,
                question_count=payload.question_count,
                generate_tests=payload.generate_tests,
                generate_questions=payload.generate_questions,
            )
        except Exception as e:
            raise HTTPException(502, f"Gemini API error: {e}")

        if teacher:
            title = _title_from_text(payload.text)
```

```
        if payload.generate_tests and data.get("tests"):
            test = models.Test(
                uuid=str(uuid_lib.uuid4()),
                teacher_id=teacher.id,
                title=title,
            )
            db.add(test)
            db.flush()
            for q in data["tests"]:
                db.add(models.TestQuestion(
                    test_uuid=test.uuid,
                    type=q["type"],
                    question_text=q["question"],
                    options=q.get("options"),
                    correct_answer=q["answer"],
                ))

        if payload.generate_questions and data.get("questions"):
            qs = models.QuestionSet(
                uuid=str(uuid_lib.uuid4()),
                teacher_id=teacher.id,
                title=title,
            )
            db.add(qs)
            db.flush()
            for q in data["questions"]:
                db.add(models.ShortQuestion(
                    set_uuid=qs.uuid,
                    question_text=q["question"],
                    correct_answer=q["answer"],
                ))

        db.commit()

    return GenerateResponse(**data)
```

Реалізований модуль забезпечує виконання вимог ФВ-03, ФВ-04 та ФВ-05: налаштування параметрів генерації користувачем, підтримку двох типів питань та збереження результатів у базі даних для зареєстрованого викладача. Обробка помилок Gemini API через HTTPException(502) відповідає вимозі НВ-04.

4.3. Реалізація інтерфейсу користувача

Клієнтська частина системи реалізована як односторінковий застосунок на React з використанням React Router для маршрутизації між сторінками. Інтерфейс організовано у вигляді компонентів, що відповідають окремим функціональним блокам системи. Взаємодія з бекендом здійснюється через Axios з базовою URL-адресою серверного API [10].

Керування станом автентифікації реалізовано через React Context API у файлі AuthContext.jsx (лістинг 4.7). Контекст зберігає JWT-токен у localStorage і надає компонентам застосунку три значення: сам токен, булевий прапорець isAuth та функції login і logout. Функція login записує токен одночасно у localStorage і у стан React, що забезпечує збереження сесії між перезавантаженнями сторінки [22]. Функція logout видаляє токен з обох сховищ. Хук useAuth надає доступ до контексту з будь-якого дочірнього компонента без передачі пропсів через проміжні рівні [10].

Лістинг 4.7 – Контекст автентифікації

```
const [token, setToken] = useState(
  () => localStorage.getItem("token")
);

function login(newToken) {
  localStorage.setItem("token", newToken);
  setToken(newToken);
}

function logout() {
  localStorage.removeItem("token");
  setToken(null);
}
```

Компонент Layout.jsx реалізує навігаційну панель, що відображається на всіх сторінках застосунку (лістинг 4.8). Вміст панелі змінюється залежно від стану автентифікації: для неавторизованого користувача відображаються посилання

«Увійти» і «Реєстрація», для авторизованого – «Кабінет» і кнопка «Вийти». При виході викликається logout з контексту і виконується перенаправлення на головну сторінку через useNavigate.

Лістинг 4.8 – Навігаційна панель з умовним рендерингом

```
      {isAuth ? (
        <>
          <Link to="/dashboard">Кабінет</Link>
          <button                                    className="btn-link"
onClick={handleLogout}>Вийти</button>
        </>
      ) : (
        <>
          <Link to="/login">Увійти</Link>
          <Link        to="/register"        className="btn        btn-
primary">Реєстрація</Link>
        </>
      )}
```

Сторінка генерації Generate.jsx реалізована як багатокроковий процес зі станом step, що послідовно приймає значення upload → settings → loading → results. На кроці upload відображається зона drag-and-drop для завантаження файлу (лістинг 4.9). Зона реагує на події onDrop та onDragOver і за кліком відкриває прихований елемент <input type="file"> через реф. Після вибору файлу викликається функція handleFile, що надсилає файл на ендпоінт POST /parse і при успішній відповіді зберігає витягнутий текст у стані компонента та переходить до кроку settings.

Лістинг 4.9 – Зона drag-and-drop для завантаження силабусу

```
      <div
        className={`dropzone${dragOver ? " dragover" : ""}`}
        onClick={() => fileInputRef.current.click()}
        onDrop={onDrop}
        onDragOver={(e) => { e.preventDefault(); setDragOver(true); }}
        onDragLeave={() => setDragOver(false)}
      >
```

```
      <p>Перетягніть файл або <strong>натисніть для
вибору</strong></p>
      <p classNamе="dropzone-hint">PDF, DOCX, TXT, до 10 МБ</p>
      <input ref={fileInputRef} type="file" accept=".pdf,.docx,.txt"
        style={{ display: "none" }} onChange={onFileInput} />
    </div>
```

На кроці settings викладач налаштовує параметри генерації – тип матеріалу і кількість питань – після чого викликається функція handleGenerate (лістинг 4.10). Вона надсилає текст силабусу разом з параметрами на ендпоінт POST /generate і при успішній відповіді переходить до кроку results, де відображаються згенеровані питання.

Лістинг 4.10 – Функція відправки запиту на генерацію

```
    const res = await api.post("/generate", {
      text: parsedText,
      test_count: settings.testCount,
      question_count: settings.questionCount,
      generate_tests: settings.generateTests,
      generate_questions: settings.generateQuestions,
    });
    setResults(res.data);
    setStep("results");
```

Сторінка проходження тесту TakeTest.jsx реалізує відображення всіх питань одночасно з розгалуженням рендерингу за типом питання (лістинг 4.11). Для питань типу multiple_choice рендеруються радіокнопки, де значенням є перша літера варіанту відповіді. Для питань типу short_answer рендерується текстове поле. Стан відповідей зберігається у об'єкті answers з ключами question_id. Після натискання кнопки відправки формується масив відповідей і надсилається на ендпоінт POST /tests/:uuid/submit, після чого відображається сторінка з результатом.

Лістинг 4.11 – Рендер питань з розгалуженням за типом

```
    {test.questions.map((q, i) => (
```

```
        <div key={q.id} className="take-question-card">
          <p><strong>{i + 1}.</strong> {q.question_text}</p>

          {q.type === "multiple_choice" && (
            <div className="take-options">
              {q.options.map((opt) => (
                <label key={opt}
                  className={answers[q.id] === opt[0] ? "selected" :
""}>
                  <input type="radio" value={opt[0]}
                    checked={answers[q.id] === opt[0]}
                    onChange={() => setAnswer(q.id, opt[0])} />
                  {opt}
                </label>
              ))}
            </div>
          )}

          {q.type === "short_answer" && (
            <input type="text"
              placeholder="Ваша відповідь..."
              value={answers[q.id] ?? ""}
              onChange={(e) => setAnswer(q.id, e.target.value)} />
          )}
        </div>
      ))}
```

Реалізований інтерфейс забезпечує виконання вимог ФВ-11, ФВ-12 та ФВ-13: проходження тесту студентом без реєстрації, відображення всіх питань на одній сторінці та автоматичну перевірку відповідей після відправки форми. Багатокроковий процес на сторінці генерації відповідає вимогам ФВ-01, ФВ-03 та ФВ-04.

4.4. Тестування системи

Тестування розробленої системи проводилось методом ad-hoc – неформалізованого дослідницького тестування без заздалегідь складених тест-кейсів. Перевірялась основна функціональність системи: парсинг силабусів різних

форматів, генерація завдань через Gemini API, збереження результатів у базі даних та коректність відображення інтерфейсу.

Головна сторінка системи SyllabiQ відображає короткий опис функціоналу та покроковий опис процесу роботи (рисунок 4.1). Навігаційна панель коректно відображає різний набір елементів залежно від стану автентифікації користувача.

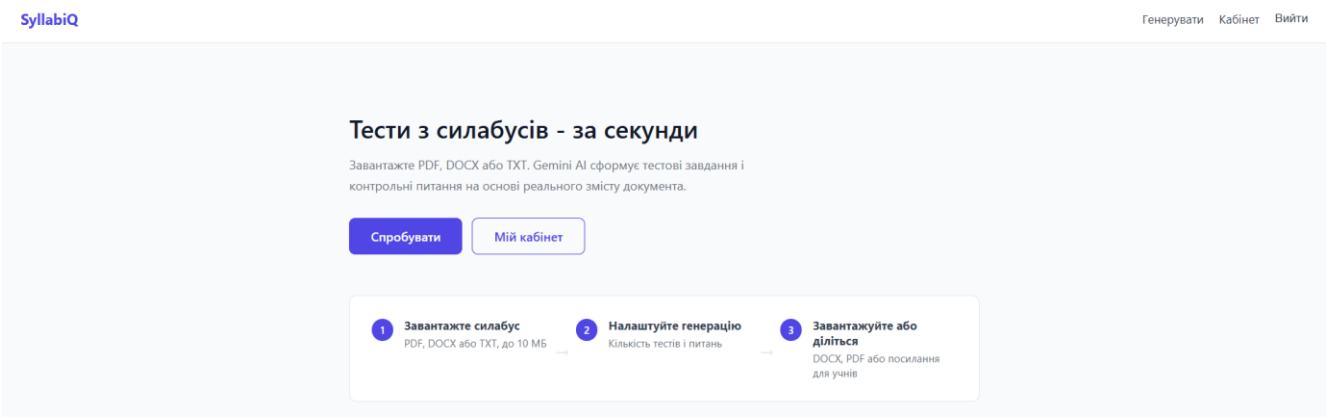


Рисунок 4.1 – Головна сторінка системи SyllabiQ

Особистий кабінет зареєстрованого викладача відображає списки створених тестів та наборів контрольних питань з інформацією про кількість питань, кількість студентів що пройшли тест, та дату створення (рисунок 4.2). Для кожного запису доступні дії: редагування, експорт у DOCX, експорт у DOCX з відповідями, отримання посилання та видалення.

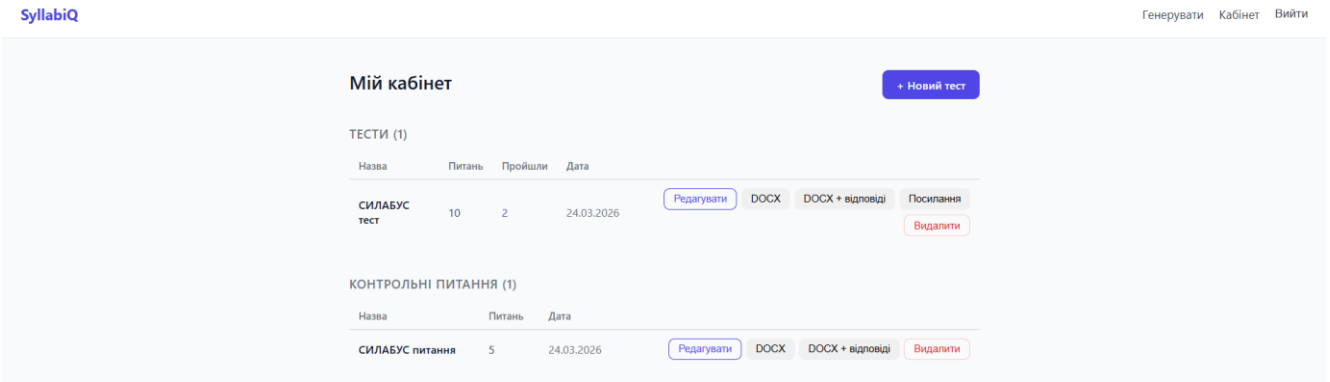


Рисунок 4.2 – Особистий кабінет викладача

Сторінка генерації реалізує двокроковий процес. На першому кроці відображається зона drag-and-drop для завантаження силабусу (рисунок 4.3). Під час тестування перевірено завантаження файлів у форматах PDF та DOCX – текст коректно витягується і передається до наступного кроку.

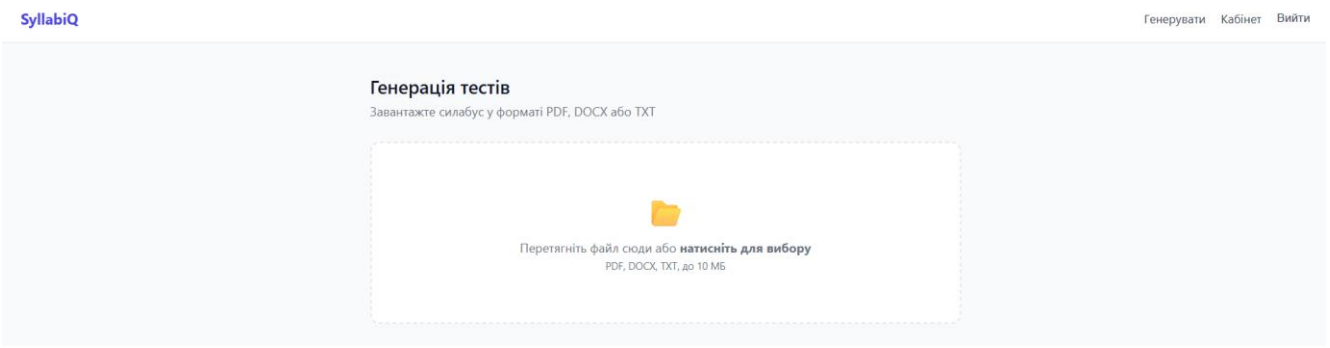


Рисунок 4.3 – Сторінка завантаження силабусу

На другому кроці відображається форма налаштувань генерації з чекбоксами для вибору типу матеріалу та числовими полями для введення кількості питань (рисунок 4.4). Значення за замовчуванням складають 10 тестових завдань і 5 контрольних питань.

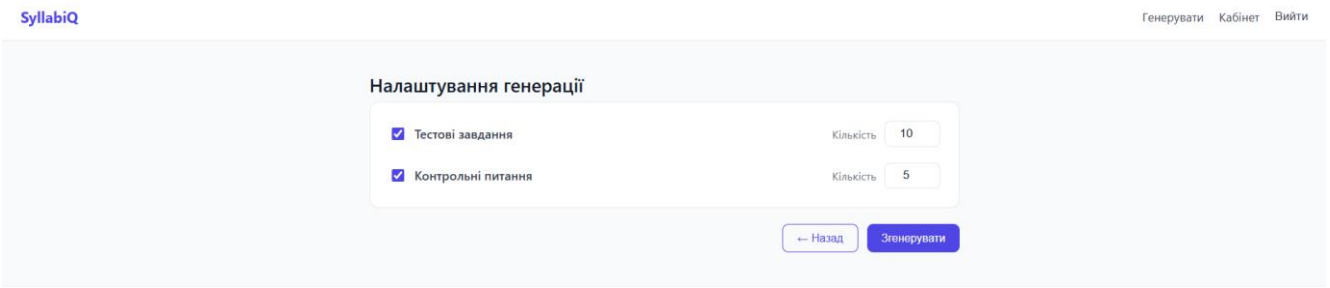


Рисунок 4.4 – Форма налаштувань генерації

Після генерації відображається сторінка результатів з повним списком тестових завдань та контрольних питань (рисунки 4.5, 4.6). Тестові завдання відображаються з виділенням правильної відповіді. Питання згенеровано на основі силабусу дисципліни «Основи інженерії програмного забезпечення» – всі питання

стосуються змісту предметної галузі, а не структури силабусу як документа, що підтверджує коректність промпту.

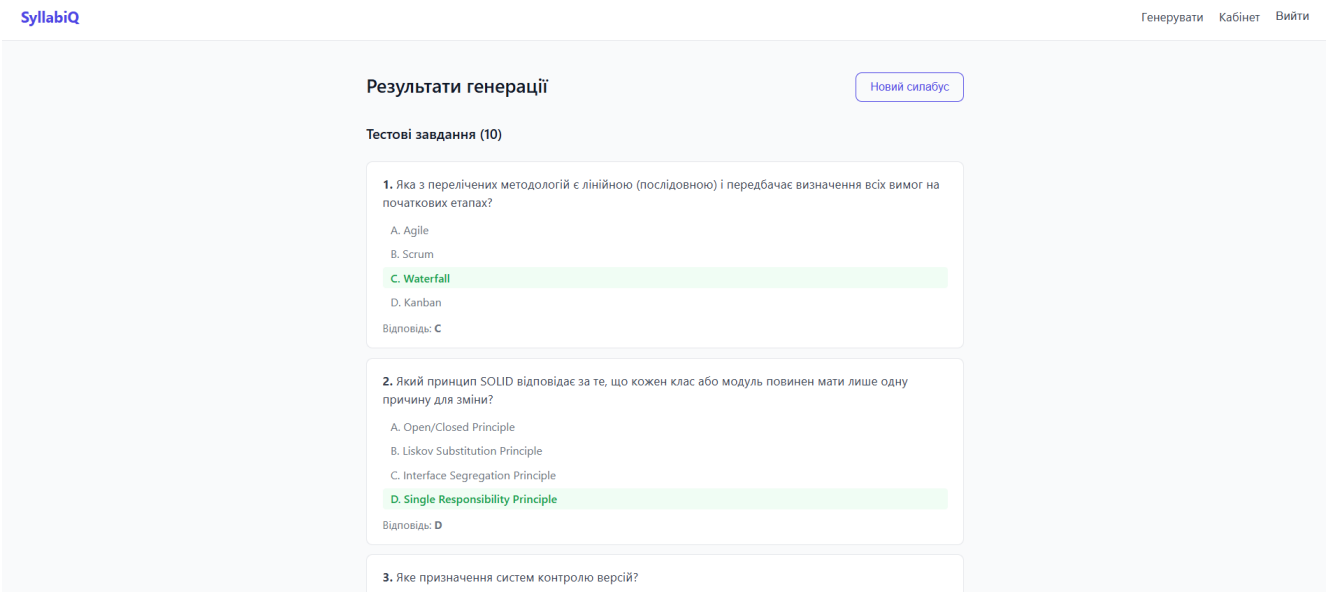


Рисунок 4.5 – Результати генерації: тестові завдання

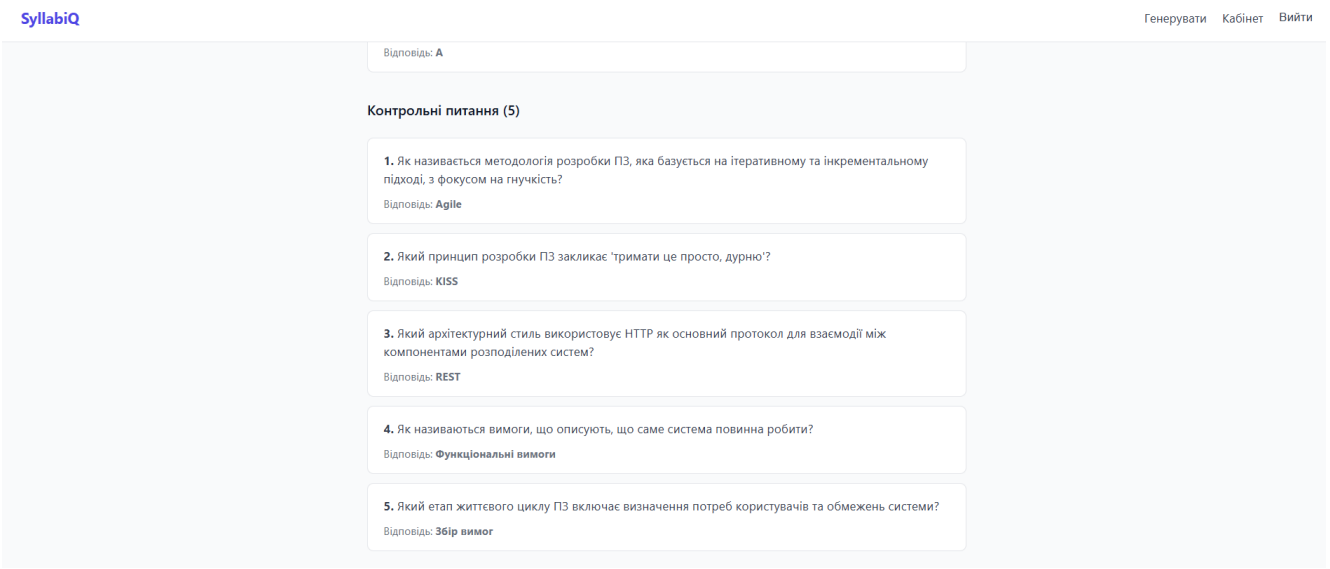
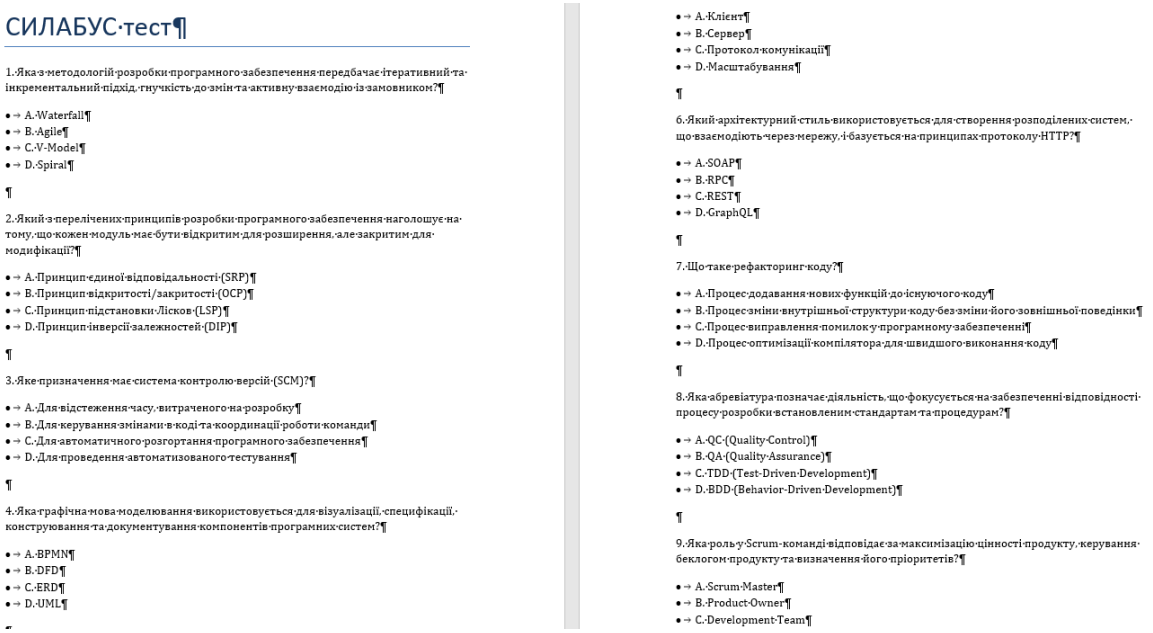


Рисунок 4.6 – Результати генерації: контрольні питання

Перевірено також функціонал експорту: завантажений DOCX-файл містить всі згенеровані питання з варіантами відповідей у коректному форматуванні (рисунок 4.7).



СИЛАБУС·тест¶

1.·Яка·з·методологій·розробки·програмного·забезпечення·передбачає·ітеративний·та·інкрементальний·підхід,·гнучкість·до·змін·та·активну·взаємодію·із·замовником?¶

• → A.·Waterfall¶
• → B.·Agile¶
• → C.·V-Model¶
• → D.·Spiral¶

¶

2.·Який·з·перелічених·принципів·розробки·програмного·забезпечення·наголошує·на·тому,·що·кожен·модуль·має·бути·відкритим·для·розширення,·але·закритим·для·модифікації?¶

• → A.·Принцип·єдиної·відповідальності·(SRP)¶
• → B.·Принцип·відкритості/закритості·(OCP)¶
• → C.·Принцип·підстановки·Лісков·(LSP)¶
• → D.·Принцип·інверсії·залежностей·(DIP)¶

¶

3.·Яке·призначення·має·система·контролю·версій·(SCM)?¶

• → A.·Для·відстеження·часу,·витраченого·на·розробку¶
• → B.·Для·керування·змінами·в·коді·та·координації·роботи·команди¶
• → C.·Для·автоматичного·розгортання·програмного·забезпечення¶
• → D.·Для·проведення·автоматизованого·тестування¶

¶

4.·Яка·графічна·мова·моделювання·використовується·для·візуалізації,·специфікації,·конструювання·та·документування·компонентів·програмних·систем?¶

• → A.·BPMN¶
• → B.·DFD¶
• → C.·ERD¶
• → D.·UML¶

¶

• → A.·Клієнт¶
• → B.·Сервер¶
• → C.·Протокол·комунікації¶
• → D.·Масштабування¶

¶

6.·Який·архітектурний·стиль·використовується·для·створення·розподілених·систем,·що·взаємодіють·через·мережу,·і·базується·на·принципах·протоколу·HTTP?¶

• → A.·SOAP¶
• → B.·RPC¶
• → C.·REST¶
• → D.·GraphQL¶

¶

7.·Що·таке·рефакторинг·коду?¶

• → A.·Процес·додавання·нових·функцій·до·існуючого·коду¶
• → B.·Процес·зміни·внутрішньої·структури·коду·без·зміни·його·зовнішньої·поведінки¶
• → C.·Процес·виправлення·помилок·у·програмному·забезпеченні¶
• → D.·Процес·оптимізації·компілятора·для·швидшого·виконання·коду¶

¶

8.·Яка·абревіатура·позначає·діяльність,·що·фокусується·на·забезпеченні·відповідності·процесу·розробки·встановленим·стандартам·та·процедурам?¶

• → A.·QC·(Quality·Control)¶
• → B.·QA·(Quality·Assurance)¶
• → C.·TDD·(Test-Driven·Development)¶
• → D.·BDD·(Behavior-Driven·Development)¶

¶

9.·Яка·роль·у·Scrum-команді·відповідає·за·максимізацію·цінності·продукту,·керування·беклогом·продукту·та·визначення·його·пріоритетів?¶

• → A.·Scrum·Master¶
• → B.·Product·Owner¶
• → C.·Development·Team¶

Рисунок 4.7 – Експортований DOCX-файл з тестовими завданнями

Тестування підтвердило працездатність основних функціональних модулів системи: парсингу документів, генерації завдань через Gemini API, збереження матеріалів у базі даних та експорту результатів. Виявлених критичних помилок у перевіреній функціональності не зафіксовано.

ВИСНОВКИ

У кваліфікаційній роботі розроблено веб-застосунок для автоматизованої генерації тестових завдань та контрольних питань на основі силабусів навчальних дисциплін з використанням великих мовних моделей. Поставлена мета досягнута – система забезпечує повний цикл роботи з навчальними матеріалами від завантаження силабусу до організації доступу студентів до тесту та перегляду результатів.

У ході дослідження вирішено всі поставлені завдання. За результатами аналізу предметної галузі встановлено, що існуючі системи – Quizizz AI, Moodle та Questgen – не забезпечують повного циклу роботи із силабусом як вхідним документом довільного формату. Жодна з розглянутих систем не поєднує в єдиному інтерфейсі генерацію на основі завантаженого документа, збереження матеріалів, організацію проходження тестів та перегляд детальної історії відповідей студентів, що обґрунтувало доцільність розробки.

Сформульовано 20 функціональних та 7 нефункціональних вимог до системи, що охоплюють модулі парсингу документів, генерації, керування матеріалами, проходження тесту, перегляду результатів та авторизації. Обґрунтовано вибір технологічного стеку: React забезпечує декларативний рендеринг довільної кількості питань, FastAPI – асинхронну обробку запитів до зовнішнього API, MySQL – реляційне зберігання даних зі збереженням цілісності при паралельних операціях, Gemini API – генерацію змістовних питань на основі тексту силабусу.

Спроєктовано архітектуру системи з чітким розділенням клієнтської і серверної частин, описано компонентну структуру, розроблено діаграми варіантів використання, компонентів, розгортання та діяльності. Спроєктовано реляційну схему бази даних з семи таблиць з дотриманням вимог третьої нормальної форми. Розроблено алгоритм формування промпту з динамічною частиною, що формується залежно від параметрів запиту, та механізм отримання структурованої відповіді у форматі JSON.

Реалізовано модуль парсингу документів з підтримкою форматів PDF, DOCX та TXT, модуль взаємодії з Gemini API з обробкою помилок зовнішнього сервісу, клієнтський інтерфейс з багатокроковим процесом генерації та розгалуженим рендерингом питань за типом. Проведено ad-hoc тестування основної функціональності системи, що підтвердило коректність парсингу документів, генерації завдань, збереження результатів у базі даних та експорту матеріалів у форматі DOCX.

Практичне значення розробленої системи полягає у скороченні часу викладача на підготовку контрольних матеріалів шляхом делегування генерації питань мовній моделі з одночасним збереженням можливості ручного редагування результату. Система підтримує гостьовий режим без реєстрації для разового використання та режим зареєстрованого викладача для постійного зберігання і розповсюдження матеріалів.

Напрямами подальшого розвитку системи є розширення підтримуваних форматів вхідних документів, додавання механізму налаштування промпту користувачем, реалізація аналітики результатів тестування та підтримка додаткових типів питань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Villalobos-Arias L., Quesada-López C., Martínez A. Automatic question generation for educational purposes: a systematic review. *Revista Educación*. 2022. Vol. 46, No. 1. P. 38–57. URL: https://doi.org/10.15517/revedu.v46i1.46485 .
2. Google DeepMind. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint*. 2024. arXiv:2403.05530. URL: https://arxiv.org/abs/2403.05530 .
3. Kurdi G., Leo J., Parsia B., Sattler U., Al-Emari S. A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*. 2020. Vol. 30. P. 121–204. URL: https://doi.org/10.1007/s40593-019-00186-y .
4. Zawacki-Richter O., Marín V. I., Bond M., Gouverneur F. Systematic review of research on artificial intelligence applications in higher education – where are the educators? *International Journal of Educational Technology in Higher Education*. 2019. Vol. 16. Art. 39. URL: https://doi.org/10.1186/s41239-019-0171-0.
5. Grunert O'Brien J., Millis B. J., Cohen M. W. *The Course Syllabus: A Learning-Centered Approach*. 2nd ed. San Francisco: Jossey-Bass, 2008. 288 p.
6. Zhao W. X., Zhou K., Li J. та ін. A survey of large language models. *arXiv preprint*. 2023. arXiv:2303.18223. URL: https://arxiv.org/abs/2303.18223.
7. Quizizz. AI-powered quiz and lesson creation. 2024. URL: https://quizizz.com/ai (дата звернення: 15.02.2026).
8. Steuer R., Filighera A., Tregel T. Questgen: An open-source NLP library for question generation. *Proceedings of the 15th International Conference on Computer Supported Education*. 2023. P. 312–319. URL: https://doi.org/10.5220/0011997400003470.
9. Robertson S., Robertson J. *Mastering the Requirements Process: Getting Requirements Right*. 3rd ed. Upper Saddle River: Addison-Wesley, 2012. 480 p.
10. Wieruch R. *The Road to React*. 2022. 329 p. URL: https://www.roadtoreact.com.

11. Ramírez S. FastAPI documentation. 2024. URL: https://fastapi.tiangolo.com (дата звернення: 20.02.2026).

12. Schwartz B., Zaitsev P., Tkachenko V. *High Performance MySQL*. 4th ed. Sebastopol: O'Reilly Media, 2022. 580 p.

13. Google. Gemini API documentation: Models and pricing. 2024. URL: https://ai.google.dev/gemini-api/docs/models (дата звернення: 10.03.2026).

14. Fowler M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Boston: Addison-Wesley, 2003. 208 p.

15. McKinney W. *Python for Data Analysis*. 3rd ed. Sebastopol: O'Reilly Media, 2022. 579 p.

16. Codd E. F. Further normalization of the data base relational model. *Data Base Systems*. Englewood Cliffs: Prentice-Hall, 1972. P. 33–64.

17. Elmasri R., Navathe S. B. *Fundamentals of Database Systems*. 7th ed. Hoboken: Pearson, 2016. 1280 p.

18. Google. Google Gen AI Python SDK. 2024. URL: https://github.com/googleapis/python-genai (дата звернення: 15.03.2026).

19. Pydantic. Pydantic documentation v2. 2024. URL: https://docs.pydantic.dev/latest (дата звернення: 20.03.2026)

20. Ramírez S. FastAPI: Dependencies. 2024. URL: https://fastapi.tiangolo.com/tutorial/dependencies (дата звернення: 05.04.2026).

21. React. Context – React documentation. 2024. URL: https://react.dev/reference/react/createContext (дата звернення: 15.04.2026).

ДОДАТКИ

Додаток А

Графічні матеріали

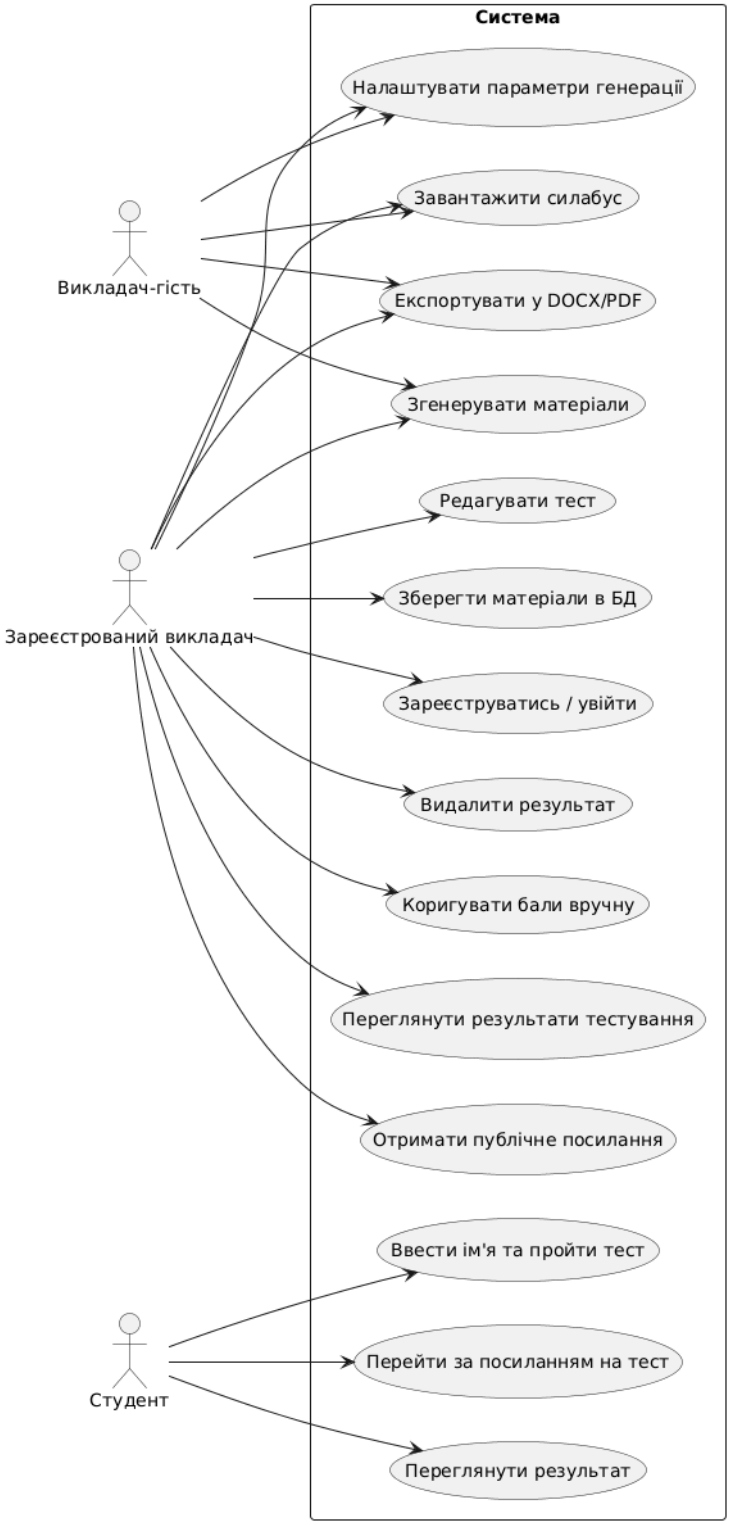


Рисунок А.1 – Use Case