НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету Інформаційних технологій	**Завідувач кафедри** Комп'ютерних систем, мереж та кібербезпеки
_____Ігор БОЛБОТ (підпис)	_____Дмитро КАСАТКІН (підпис)
«___»_____2026 р.	«___»_____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Розроблення комп'ютерної системи тестування знань із забезпеченням генерування звітів»

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

Гарант освітньої програми канд. фіз.-мат. наук, доцент	_____ (підпис)	**Євгеній НІКІТЕНКО**
Керівник бакалаврської кваліфікаційної роботи докт. техн. наук, доцент	_____ (підпис)	**Вадим ШКАРУПИЛО**
Виконав	_____ (підпис)	**Сергій БІЛОУСЬКО**

КИЇВ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
канд. пед. наук, доцент Дмитро КАСАТКІН
«_____»_____20____ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Білоусько Сергій Сергійович
(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Тема кваліфікаційної бакалаврської роботи
«Розроблення комп'ютерної системи тестування знань із забезпеченням генерування звітів»

затверджена наказом ректора НУБіП України від «10»___12_____2025 р. №3072«С»___
Термін подання завершеної роботи на кафедру «_»_____2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, що підлягають дослідженню:
1. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ
2. ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ
3. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ
4. ТЕСТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання "____" _____2026 р.

Керівник бакалаврської кваліфікаційної роботи докт. техн. наук, доцент	_____ (підпис)	**Вадим ШКАРУПИЛО**
Завдання взяв до виконання	_____ (підпис)	**Сергій БІЛОУСЬКО**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	04.08.2025 р.	Виконано
2	Проектування системи	15.01.2026 р.	Виконано
3	Реалізація системи	10.02.2026 р.	Виконано
4	Тестування системи	01.03.2026 р.	Виконано
5	Оформлення пояснювальної записки	13.04.2026 р.	Виконано
6	Оформлення графічного матеріалу	13.04.2026 р.	Виконано

Студент _____ **Білоусько С.С.**
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ **Шкарупило В.В.**
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 72 сторінки, 13 рисунків, 15 таблиць, 12 лістингів, 20 джерел.

КОМП'ЮТЕРНА СИСТЕМА, FLASK, PYTHON, POSTGRESQL, DOCKER, NGINX, БАЗА ДАНИХ, АВТОМАТИЧНА ПЕРЕВІРКА

Об'єкт аналізу - процес організації та проведення комп'ютерного тестування знань користувачів із використанням автоматизованих засобів оцінювання результатів.

Мета роботи - розроблення комп'ютерної системи для проведення тестування знань користувачів та формування звітів.

Проект складається з 4 розділів.

Перший розділ присвячено аналізу предметної області та існуючих підходів до організації комп'ютерного тестування знань.

У другому розділі виконано проєктування комп'ютерної системи. Описано структуру системи, її основні компоненти та взаємодію між ними.

У третьому розділі наведено реалізацію комп'ютерної системи. Описано використані технології програмування, структуру програмного коду, логіку роботи основних модулів системи.

У четвертому розділі описано процес тестування розробленої комп'ютерної системи.

У результаті було розроблено комп'ютерну систему тестування знань користувачів, яка забезпечує перевірку тестових завдань, збереження результатів у базі даних та формування звітів тест

ЗМІСТ

ВСТУП

Актуальність теми зумовлена масовим впровадженням ІТ-рішень в освітній процес, зокрема у сферу контролю та верифікації знань. На відміну від класичних форм перевірки (письмових екзаменів чи усних заліків), які є трудомісткими та тривалими в обробці, цифрові тестові платформи забезпечують миттєву автоматизацію всіх етапів оцінювання.

Впровадження автоматизованих систем суттєво спрощує моніторинг успішності, гарантує неупередженість виставлення балів та мінімізує час на очікування результатів. Подібний інструментарій є потрібним не лише в академічному середовищі, а й у центрах сертифікації чи корпоративному секторі для підтвердження кваліфікації персоналу.

Технічні переваги таких рішень полягають у можливості централізованого зберігання даних у БД, проведенні статистичного аналізу та автоматичній генерації звітності. Гнучкість систем забезпечується підтримкою варіативних форматів завдань, що адаптує процес тестування до потреб конкретного користувача.

Мета дослідження полягає у проектуванні та реалізації програмного комплексу для тестування, що забезпечить повний цикл роботи з даними: від проходження тесту користувачем до автоматичного аналізу відповідей і формування підсумкової документації.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Вимоги для розробки системи

Проєктуванню системи передує етап формування технічних вимог, які окреслюють архітектуру, функціональне наповнення та експлуатаційні особливості майбутнього продукту [5], [17].

Ключовим завданням платформи є менеджмент процесу оцінювання знань. Програмне забезпечення має реалізувати повний цикл взаємодії: від ідентифікації респондента до верифікації відповідей і фіксації підсумкових даних у сховищі [5].

Програмне рішення повинно підтримувати такі операції:

— Авторизація: обов'язкова реєстрація учасника перед стартом сесії.

— Інструктаж: ознайомлення з регламентом проведення тесту.

— Навігація: надання доступу до блоку запитань.

— Обробка даних: автоматичний аналіз відповідей та розрахунок фінального бала.

— Архівування: запис результатів у базу даних із можливістю подальшого перегляду.

— Варіативність контенту

— Система має коректно опрацьовувати різні формати завдань:

— Вибір єдиної вірної опції (Single Choice).

— Множинний вибір варіантів (Multiple Choice).

— Текстове введення відповіді (Open-ended questions).

— Логічне зіставлення пар (Matching tasks).

Для управління контентом передбачено панель адміністратора. Модератор отримує інструменти для створення нових модулів, коригування наявних запитань та моніторингу статистики успішності всіх користувачів [1], [2].

Окрім базових опцій, до системи висуваються вимоги щодо якості:

— Юзабіліті: ергономічний та інтуїтивно зрозумілий інтерфейс.

— Надійність: стійкість до відмов та стабільна робота під навантаженням.
— Безпека: запобігання дублюванню спроб проходження одним і тим самим учасником.

Визначений перелік специфікацій є фундаментом для подальшого розроблення та впровадження системи у робоче середовище.

1.2 Огляд існуючих систем

Сучасний ринок програмного забезпечення пропонує різноманітні інструменти для автоматизації контролю знань та управління освітнім процесом. Такі системи є критично важливими для академічних установ і корпоративного сектора, оскільки вони мінімізують трудовитрати на перевірку робіт, прискорюють отримання результатів та забезпечують високу об’єктивність оцінювання. Переважна більшість актуальних рішень базується на веб-технологіях, що дозволяє взаємодіяти з платформою через будь-який сучасний браузер без інсталяції додаткового ПЗ, забезпечуючи мобільність та кросплатформність [5], [7], [8], [19].

Серед лідерів ринку варто виділити систему Moodle - потужну платформу для управління навчанням. Вона володіє широким інструментарієм для створення комплексних курсів, підтримує численні типи запитань та автоматизує збір статистики. Проте її багатофункціональність має зворотний бік: інтерфейс є досить громіздким, а процес конфігурації - складним, що не завжди доцільно для невеликих проєктів.

Іншим популярним рішенням є Google Forms. Цей сервіс приваблює інтуїтивним інтерфейсом та швидкістю розгортання опитувань. Користувачі можуть легко структурувати питання та аналізувати результати за допомогою автоматично згенерованих діаграм. Водночас функціонал сервісу обмежений: він не дозволяє реалізовувати складні алгоритми тестування, інтегруватися з

приватними базами даних або гнучко змінювати бізнес-логіку системи.

Спеціалізовані ресурси, такі як Testportal або ClassMarker, зосереджені саме на проведенні іспитів. Вони пропонують інструменти для дотримання академічної доброчесності, обмеження часу та детальної звітності. Однак, попри наявність таких рішень, використання сторонніх платформ часто супроводжується проблемами: надлишковим функціоналом, складністю кастомізації або неможливістю повної інтеграції з іншим програмним забезпеченням організації [1-5].

Саме тому розробка власної системи тестування часто є більш раціональним кроком. Створення індивідуального програмного продукту дозволяє реалізувати лише необхідні функції, спростити інтерфейс під потреби конкретних користувачів та забезпечити повний контроль над обробкою і зберіганням інформації. У межах поточної роботи планується створення саме такої системи, яка поєднає підтримку різних форматів питань, автоматичну верифікацію відповідей та ефективне управління базою даних результатів. Аналіз існуючих аналогів підтверджує, що індивідуальна розробка дозволяє отримати максимально адаптований та ефективний інструмент для автоматизації перевірки знань [7-12].

1.3 Висновки до розділу

Підсумовуючи результати аналізу технічного завдання, було сформовано перелік ключових вимог до майбутнього програмного забезпечення. Основним функціональним вектором визначено автоматизацію циклу оцінювання, що включає безпосереднє тестування респондентів, миттєву верифікацію наданих відповідей та архівацію отриманих балів у базі даних.

Дослідження ринку існуючих аналогів підтвердило затребуваність таких інструментів в освітньому середовищі. Попри розвинену функціональність

популярних платформ, було обґрунтовано доцільність розробки власного програмного продукту. Це дозволить уникнути надмірної складності інтерфейсу, реалізувати специфічні запити та забезпечити повну відповідність системи поставленим цілям.

Висновки, отримані на етапі аналізу, стануть фундаментом для подальшої архітектурної побудови та практичної реалізації комп’ютерної системи.

2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

2.1 Загальна архітектура системи

Архітектурна побудова програмного комплексу визначає його структурну організацію та принципи комунікації між ключовими вузлами. В основу розробленого рішення покладено клієнт-серверну модель, що є стандартом для сучасних інформаційних платформ [18].

Дана архітектура передбачає чіткий розподіл функцій між трьома рівнями:

— Клієнтський рівень (Frontend): забезпечує візуальне представлення даних та інтерфейс взаємодії. Користувач отримує доступ до функціоналу через веб-браузер, який генерує запити та відображає отриману від сервера інформацію.

— Серверний рівень (Backend): виступає ядром системи, де зосереджена вся бізнес-логіка. Тут здійснюється маршрутизація запитів, генерація тестових варіантів, алгоритмічна перевірка правильності відповідей та підготовка даних для запису.

— Рівень бази даних (Database): забезпечує надійне збереження та структуризацію масивів інформації, необхідних для коректної роботи додатка.

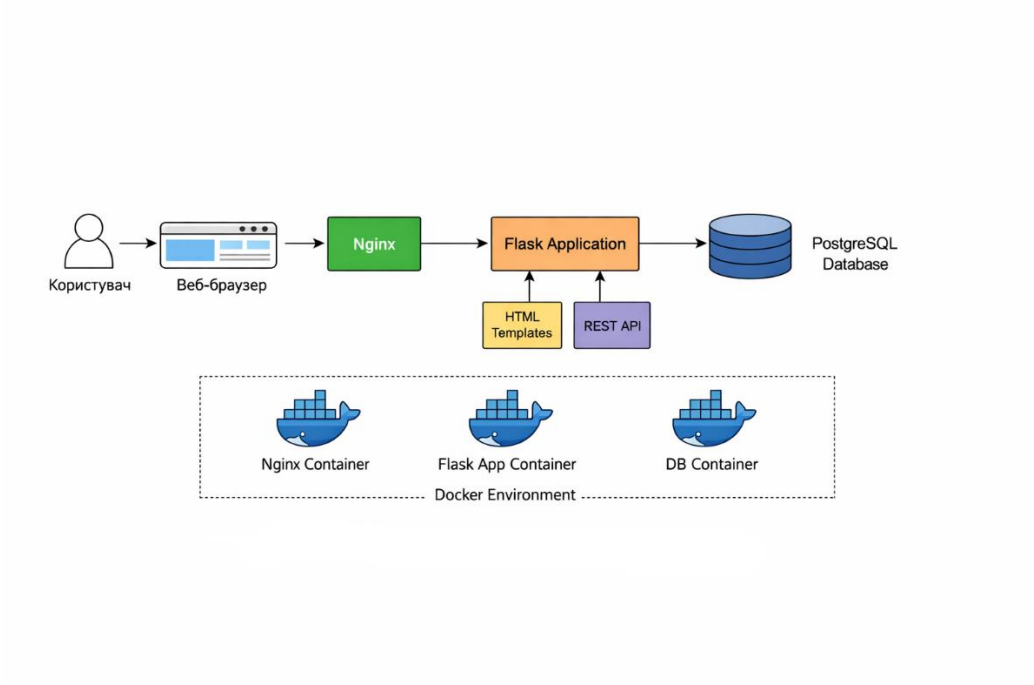

Рисунок 2.1 - Загальна архітектура комп’ютерної системи

Для реалізації системи було використано ряд сучасних технологій [7-12]. Основні компоненти системи наведені у таблиці 2.1.

Таблиця 2.1 - Основні компоненти комп’ютерної системи

Компонент	Опис	Призначення
Flask	Веб-фреймворк мови Python	Реалізація серверної логіки системи
PostgreSQL	Реляційна база даних	Зберігання питань, відповідей та результатів
Nginx	Веб-сервер	Обробка HTTPS-запитів та проксування до Flask
Docker	Система контейнеризації	Ізоляція та запуск компонентів системи
HTML/CSS	Мова розмітки та стилів	Формування інтерфейсу користувача

Застосування обраного технологічного стеку забезпечує високу адаптивність та потенціал для розширення програмного комплексу. Такий підхід

гарантує коректне функціонування системи на різноманітних апаратних конфігураціях і пристроях [6], [20].

2.2 Функціональна структура системи

Архітектура функціональних можливостей визначає перелік ключових завдань, які реалізує програмний продукт. Створена платформа для тестування структурована за модульним принципом і охоплює кілька пріоритетних напрямів [18].

До базового функціонала додатка належать:

— ідентифікація та реєстрація учасників перед стартом сесії;
— безпосередня взаємодія з тестовим контентом;
— миттєвий алгоритмічний аналіз отриманих відповідей;
— фіксація та архівування підсумкових балів у системі;
— доступ до перегляду статистики успішності;
— генерація вихідної документації та звітів у форматі PDF.

Основні функції комп’ютерної системи наведені у таблиці 2.2.

Таблиця 2.2 - Основні функції системи

№	Функція	Опис
1	Реєстрація користувача	Користувач вводить свої дані перед початком тесту
2	Проходження тесту	Система відображає тестові питання
3	Перевірка відповідей	Система автоматично перевіряє відповіді
4	Збереження результату	Результати тестування зберігаються у базі даних
5	Перегляд результатів	Користувач може переглянути результати
6	Генерація PDF	Система формує PDF-звіт з результатами

Логіка роботи та взаємодії всередині системи візуалізована за допомогою UML-діаграми прецедентів (use case diagram), яка наочно демонструє ролі користувачів та доступні їм операції [6], [20].

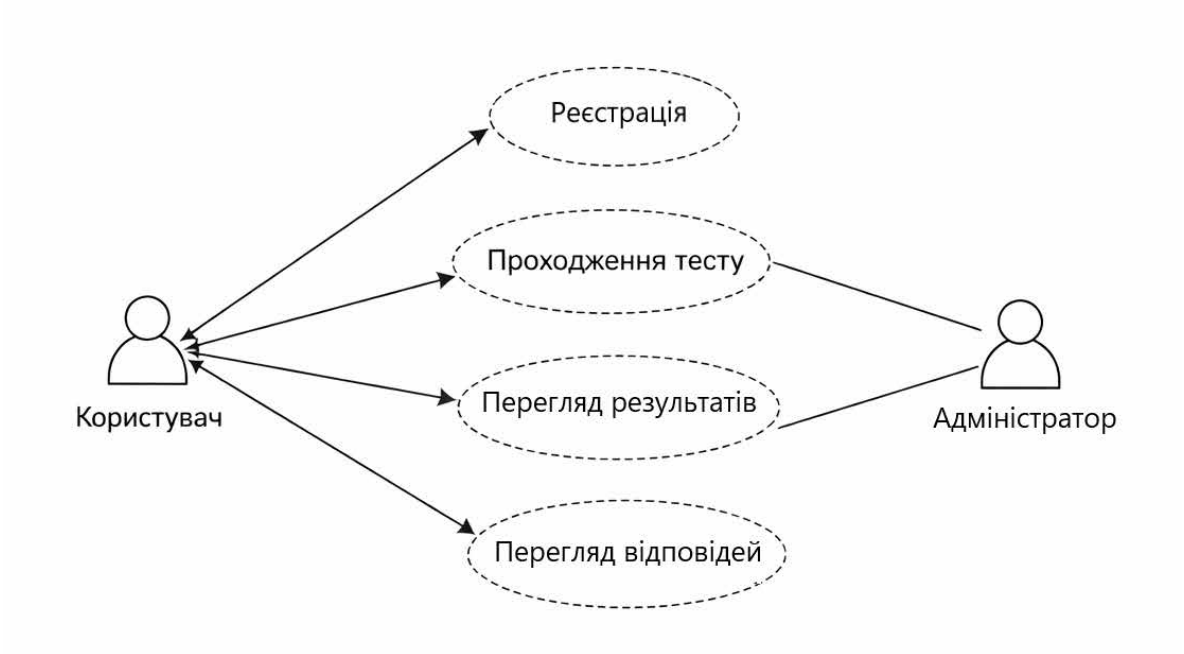

Рисунок 2.2 - Діаграма варіантів використання системи

2.3 Типи тестових питань

Програмний комплекс розроблений із підтримкою варіативних форматів запитань. Такий підхід не лише розширює можливості перевірки компетенцій респондентів, а й робить процес оцінювання більш гнучким та адаптивним до специфіки різних навчальних дисциплін [5].

Основні типи тестових питань, що використовуються у системі, наведені у таблиці 2.3.

Таблиця 2.3 - Типи тестових питань

Тип питання	Опис	Приклад
Single choice	Один правильний варіант відповіді	Столиця України
Multiple choice	Кілька правильних відповідей	Які з перелічених мов є мовами програмування
Open question	Відкрита текстова відповідь	Що таке HTTP
Matching	Встановлення відповідності	Термін - визначення

Застосування комбінованих форматів завдань надає можливість комплексно проаналізувати підготовку респондентів, охоплюючи як рівень засвоєння теоретичного матеріалу, так і навички його практичного використання у конкретних ситуаціях.

2.4 Проєктування структури бази даних

Сховище даних виступає критично важливим компонентом програмного комплексу, оскільки саме воно відповідає за збереження та цілісність усіх

масивів інформації.

В основу системи покладено реляційну модель бази даних. Вона структурована у вигляді взаємопов'язаних таблиць, призначених для систематизованого зберігання бази тестових завдань, доступних варіантів відповідей, а також персональних результатів проходження тестів [9].
Основні таблиці бази даних наведені у таблиці 2.4.

Таблиця 2.4 - Основні таблиці бази даних

Таблиця	Опис
Questions	Зберігає тестові питання
Options	Зберігає варіанти відповідей
Results	Зберігає результати проходження тесту

Структура таблиці результатів тестування наведена у таблиці 2.5.

Таблиця 2.5 - Структура таблиці результатів

Поле	Тип даних	Опис
id	Integer	Ідентифікатор запису
email	Text	Email користувача
fullname	Text	Ім’я користувача
score	Integer	Кількість правильних відповідей
total	Integer	Загальна кількість питань
time_spent	Integer	Час проходження тесту
violations	Integer	Кількість порушень

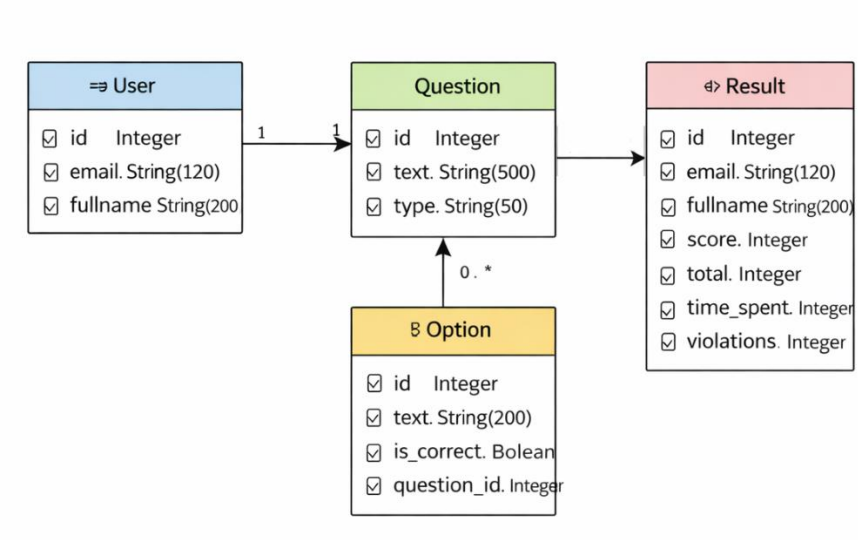

Рисунок 2.3 - Діаграма структури бази даних

2.5 Проєктування алгоритму роботи системи

Логіка функціонування програмного продукту базується на чіткій послідовності операцій, що регламентують процес взаємодії респондента з інтерфейсом [4], [16].

Ключові фази робочого циклу платформи:

— Ініціалізація: перехід користувача на головну сторінку веб-додатка.

— Ідентифікація: заповнення форми з персональними даними для авторизації в системі.

— Генерація контенту: автоматичне формування переліку завдань на основі наявних у базі даних матеріалів.

— Сесія тестування: безпосереднє надання відповідей на запропоновані питання.

— Верифікація: програмний аналіз отриманих даних та звірка з еталонними значеннями.

— Фіксація результатів: автоматичний запис підсумкового бала та статистики сесії у реляційну базу даних.

— Зворотний зв'язок: візуалізація персональних досягнень та відображення звіту про проходження тесту.

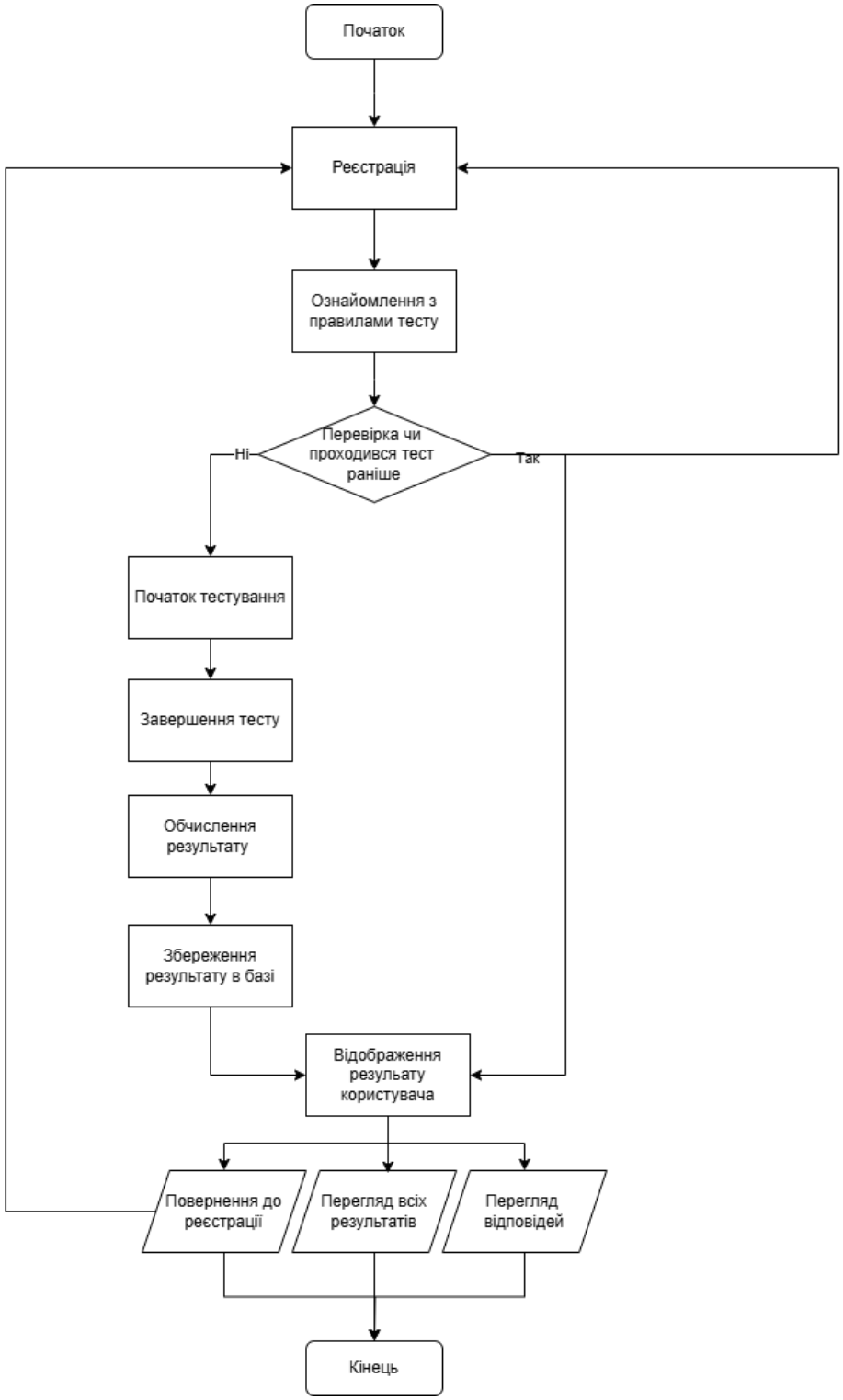

Рисунок 2.4 - Блок-схема алгоритму роботи системи

2.6 Висновки до розділу

У межах поточного розділу було завершено етап проєктування програмного комплексу для оцінювання знань. Зокрема, обґрунтовано вибір системної архітектури, ідентифіковано ключові вузли та розроблено логічну структуру функціональних модулів.

Додатково було систематизовано формати тестових завдань, інтегрованих у систему, та спроєктовано схему реляційної бази даних, яка забезпечує впорядковане зберігання екзаменаційного контенту й результатів користувачів.

Сформовані проектні рішення та технічні специфікації стали фундаментом для практичної розробки системи, детальний опис якої наведено у наступній частині роботи.

3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

3.1 Структура програмного проєкту

Розробка програмного забезпечення була здійснена на базі мови Python із залученням мікрофреймворку Flask для реалізації веб-частини [1], [2], [19]. В основу структури проєкту покладено модульний підхід, що значно спрощує процеси програмування, технічного супроводу та масштабування платформи в майбутньому [7].

Файлова ієрархія системи розділена на спеціалізовані директорії та окремі компоненти, де кожен елемент відповідає за конкретний аспект функціонування додатка. Основні компоненти програмного проєкту наведено в таблиці 3.1 [12].

Таблиця 3.1 - Структура програмного проєкту

Файл / каталог	Призначення
app.py	основний файл серверної частини системи
requirements.txt	перелік необхідних Python-бібліотек
Dockerfile	інструкції для створення Docker-образу
docker-compose.yml	конфігурація запуску контейнерів
nginx/nginx.conf	конфігурація веб-сервера Nginx
templates/	HTML-шаблони сторінок
static/	статичні файли (CSS, стилі)
results.db	база даних результатів тестування

Ядро програмного комплексу зосереджене у файлі app.py, який виконує роль головного контролера. У ньому реалізовані такі ключові компоненти:

— Маршрутизація: опис шляхів комп'ютерної системи для переходу між сторінками.

— Логіка оцінювання: алгоритми, що керують процесом тестування.

— Обробка вводу: механізми аналізу відповідей, наданих користувачами.

— Робота з даними: процедури взаємодії із серверною базою даних.

— Генерація вихідних форм: функціонал для створення підсумкових звітів.

Для візуального представлення системи використовуються спеціалізовані директорії:

— templates: зберігає HTML-шаблони, що формують каркас користувацького інтерфейсу.

— static: містить CSS-стилі та інші статичні ресурси, що відповідають за графічне оформлення та адаптивність сторінок.

Така організація файлової структури дозволяє чітко розмежувати бізнес-логіку, зовнішній інтерфейс та налаштування системи, що значно полегшує технічне обслуговування коду.

3.2 Використані програмні технології

Під час створення програмного комплексу було впроваджено актуальний технологічний стек та інструментарій, що відповідає сучасним стандартам розробки ПЗ [1-3], [7-12], [13], [14].

Основні технології системи наведено в таблиці 3.2.

Таблиця 3.2 - Використані програмні технології

Технологія	Призначення
Python	мова програмування
Flask	веб-фреймворк
SQLAlchemy	робота з базою даних
PostgreSQL	система керування базами даних
HTML / CSS	інтерфейс користувача
Docker	контейнеризація системи
Nginx	веб-сервер
ReportLab	генерація PDF-звіту

Застосування обраного інструментарію забезпечило можливість проектування адаптивного та нарощуваного програмного продукту. Такий підхід гарантує безперешкодну інсталяцію та стабільне функціонування системи в межах різних операційних середовищ.

3.3 Ініціалізація веб-додатку системи

Процес ініціалізації програмного комплексу стартує з генерації екземпляра додатка Flask, після чого здійснюється конфігурація та встановлення зв'язку із сервером баз даних [7], [19].

Лістинг 3.1 - Ініціалізація Flask-додатку

```
from flask import Flask, render_template, request, redirect, session, url_for
from flask_sqlalchemy import SQLAlchemy
```

```
import random
import datetime
import os

app = Flask(__name__)
app.secret_key = "secret_key"

app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv("DATABASE_URL")
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

У представленому коді здійснюється базова підготовка середовища:

— Імпорт модулів: підключення зовнішніх залежностей та інструментарію.

— Ініціалізація: створення основного об'єкта додатка Flask.

— Безпека: визначення секретного ключа для коректного функціонування механізму сесій.

— Конфігурація БД: встановлення параметрів з'єднання з базою даних [9].

— ORM-інтеграція: ініціалізація об'єкта SQLAlchemy для зручної обробки даних через програмні моделі.

3.4 Реалізація моделей бази даних

Для управління даними та збереження підсумкових показників тестування впроваджено систему баз даних.

Процес комунікації з базою реалізовано через ORM-інструментарій SQLAlchemy. Це дозволяє структурувати реляційні таблиці у форматі об’єктно-орієнтованих класів Python, що значно спрощує маніпуляції з даними та забезпечує прозорість архітектури сховища [1], [2].

Лістинг 3.2 - Модель результатів тестування

```
class Result(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120))
    fullname = db.Column(db.String(200))
    score = db.Column(db.Integer)
    total = db.Column(db.Integer)
    time_spent = db.Column(db.Integer)
    violations = db.Column(db.Integer)
```

Ця модель зберігає основні дані про проходження тесту користувачем:

— електронну адресу;

— ПІБ користувача;

— кількість правильних відповідей;

— загальну кількість питань;

— час проходження тесту;

— кількість порушень правил тестування.

3.5 Реалізація механізму тестування

Фундаментальною можливістю платформи є безпосередня організація процесу перевірки знань.

Технічне виконання цього модуля зосереджене в рамках маршруту /test, який відповідає за логіку взаємодії користувача з екзаменаційним контентом.

Лістинг 3.3 - Реалізація маршруту тестування

```
@app.route('/test', methods=['GET', 'POST'])
def test():
    if 'email' not in session:
        return redirect('/')
```

```
                                                existing           =
Result.query.filter_by(email=session['email']).first()
    if existing:
                      return   render_template("already_passed.html",
result=existing)

    # ОДоразово генеруємо питання
    if 'questions' not in session:
        db_questions = Question.query.all()
        shuffled_questions = []

        for q in db_questions:
            question_data = {
                "id": q.id,
                "question": q.text,
                "type": q.type
            }

            # SINGLE питаня
            if q.type == "single":
                options = [opt.text for opt in q.options]
                random.shuffle(options)

                question_data["options"] = options
                question_data["answer"] = next(
                    opt.text for opt in q.options if opt.is_correct
                )

            # MULTIPLE питання
            elif q.type == "multiple":
                options = [opt.text for opt in q.options]
                random.shuffle(options)

                question_data["options"] = options
                question_data["answers"] = [
```

```
                    opt.text for opt in q.options if opt.is_correct
                ]

            # OPEN питання
            elif q.type == "open":
                question_data["keywords"] = [
                        opt.text.lower() for opt in q.options if
opt.is_correct
                ]

            # MATCHING питання (не реалізовано)
            elif q.type == "matching":
                    pairs = [(opt.text, opt.is_correct) for opt in
q.options]
                question_data["pairs"] = pairs

            shuffled_questions.append(question_data)

        # перемішуємо 1 раз
        random.shuffle(shuffled_questions)

        # зберігаємо в session
        session['questions'] = shuffled_questions

    # POST (обробка результатів)
    if request.method == 'POST':
        try:
            time_spent = int(request.form.get("time_spent", 0))
        except ValueError:
            time_spent = 0

        violations = int(request.form.get("violations", 0) or 0)
        session['violations'] = violations

        score = 0
        user_answers = []
```

```
        for i, q in enumerate(session['questions']):
            q_type = q.get("type")

            # SINGLE питання
            if q_type == "single":
                user_answer = request.form.get(f"q{i}")
                user_answers.append(user_answer)

                if user_answer == q.get("answer"):
                    score += 1

            # MULTIPLE питання
            elif q_type == "multiple":
                user_answer = request.form.getlist(f"q{i}")
                user_answers.append(user_answer)

                correct_answers = q.get("answers", [])

                if set(user_answer) == set(correct_answers):
                    score += 1

            # OPEN питання
            elif q_type == "open":
                user_answer = request.form.get(f"q{i}", "").lower()
                user_answers.append(user_answer)

                keywords = q.get("keywords", [])

                  matches = sum(1 for word in keywords if word in
user_answer)

                if keywords and matches / len(keywords) >= 0.5:
                    score += 1

            # MATCHING питання (не реалізовно )
```

```
            elif q_type == "matching":
                user_answers.append("matching_not_checked")

        result = Result(
            email=session['email'],
            fullname=session['fullname'],
            score=score,
            total=len(session['questions']),
            time_spent=time_spent,
            violations=session.get('violations', 0)
        )

        db.session.add(result)
        db.session.commit()

                total_questions = len(session['questions']) if
session.get('questions') else 0
          percentage = round((score / total_questions) * 100) if
total_questions > 0 else 0

        session['last_score'] = score
        session['last_percentage'] = percentage
        session['last_time'] = time_spent
        session['answers'] = user_answers

        return redirect(url_for('result'))

                        return render_template("test.html",
questions=session['questions'])
```

Логіка функціонування модуля тестування базується на декількох послідовних кроках, що забезпечують коректність процедури та захист даних:

— Верифікація сесії: перевірка наявності прав доступу та авторизації учасника.

— Контроль повторних спроб: системна перевірка бази даних на предмет наявності завершених тестів у даного користувача для запобігання дублюванню.
— Вибірка контенту: завантаження переліку завдань із реляційного сховища.
— Рандомізація: застосування алгоритму випадкового перемішування черговості питань для підвищення об'єктивності.
— Рендеринг інтерфейсу: передача підготовленого набору даних у відповідний HTML-шаблон для відображення клієнту.
— Аналіз вводу: отримання та програмна обробка результатів, надісланих респондентом.

Всі дані, вилучені з бази, впорядковуються у спеціалізовані структури, що оптимізує швидкість їх обробки на серверному рівні.

Лістинг 3.4 - Формування списку питань

```
for q in db_questions:
    question_data = {
        "id": q.id,
        "question": q.text,
        "type": q.type
    }
```

Для того щоб кожен користувач отримував індивідуальний список питань, вони перемішуються за допомогою:

```
random.shuffle(shuffled_questions)
```

3.6 Реалізація перевірки відповідей

Коли користувач завершує тест, запускається алгоритм перевірки відповідей.

Лістинг 3.5 - Алгоритм перевірки відповідей

```
for i, q in enumerate(session['questions']):
    user_answer = request.form.get(f"q{i}")

    if user_answer == q.get("answer"):
        score += 1
```

Алгоритмом перевірки виконуються такі дії:

- Циклічна обробка: послідовний перегляд кожного пункту екзаменаційного блоку.
- Збір даних: отримання масиву значень, введених або обраних респондентом.
- Компаративний аналіз: зіставлення отриманих відповідей із еталонними ключами, що зберігаються в системі.
- Кількісна оцінка: автоматичне обчислення сумарної кількості вірних рішень для формування фінального бала.

3.7 Реалізація HTML-шаблонів

Графічна оболонка програмного продукту побудована на базі HTML-шаблонів, розміщених у спеціалізованій директорії templates. Такий підхід дозволяє відокремити структуру сторінок від логіки обробки даних, спрощуючи редагування зовнішнього вигляду системи [12].

Основні шаблони системи наведено в таблиці 3.3.

Таблиця 3.3 - Основні HTML-шаблони системи

Шаблон	Призначення
register.html	сторінка реєстрації
rules.html	сторінка правил
test.html	сторінка тестування
result.html	сторінка результату
review.html	перегляд відповідей
admin_dashboard.html	панель адміністратора
admin_questions.html	керування питаннями

3.8 Реалізація HTML-шаблонів інтерфейсу системи

3.8.1 Загальна інформація про шаблони

Візуальна оболонка платформи базується на системі HTML-шаблонів, що зберігаються в директорії templates. Ці компоненти відповідають за генерацію веб-сторінок, які бачить кінцевий користувач у своєму браузері.

Для інтеграції динамічного контенту використовується інструмент **Jinja2**. Він дозволяє гнучко поєднувати статичну HTML-розмітку з даними, що надходять від сервера (наприклад, перелік питань або ім'я користувача).

До базового набору шаблонів системи належать:

- register.html - форма для реєстрації та ідентифікації;
- rules.html - сторінка з інструкціями та регламентом тестування;
- test.html - основний інтерфейс для проходження завдань;
- result.html - сторінка з підсумковим балом;
- results.html - сторінка з результатами користувачів;
- review.html - модуль для перегляду допущених помилок;
- admin_dashboard.html - робоча панель адміністратора;

— admin_questions.html - інструментарій для керування базою запитань.

Кожен файл у цій структурі відповідає за окремий етап взаємодії користувача з програмою, забезпечуючи цілісність інтерфейсу.

3.8.2 Шаблон сторінки реєстрації (register.html)

Компонент register.html призначений для ідентифікації респондентів перед доступом до екзаменаційного модуля.

Через інтерфейс цієї сторінки користувач надає базові відомості, а саме:

— повне ім'я (прізвище та ім'я);

— контактну електронну пошту.

Програмна реалізація базується на стандартній HTML-формі, яка використовує метод POST для безпечної передачі інформації на серверну частину. Отримані дані згодом фіксуються в активній сесії, що дозволяє системі персоналізувати процес тестування та коректно формувати звітну документацію.

Приклад фрагменту шаблону наведено у лістингу 3.6.

Лістинг 3.6 - Фрагмент шаблону реєстрації

```
<div class="container">
    <h2>Реєстрація</h2>

    <form method="POST">
            <input type="email" name="email" placeholder="Email"
required>

        {% if error %}
            <p class="error">{{ error }}</p>
        {% endif %}

            <input type="text" name="fullname" placeholder="ПІБ"
required>

        <button type="submit">Продовжити</button>
    </form>
</div>
```

3.8.3 Шаблон сторінки правил тестування (rules.html)

Після завершення реєстрації система перенаправляє користувача до модуля rules.html. Ця сторінка виконує роль інструктажу та містить регламент проведення оцінювання, зокрема:

— Порядок проходження: опис механіки взаємодії з тестом.
— Часові ліміти: інформація про встановлені часові обмеження (якщо вони передбачені конфігурацією).
— Регламент спроб: офіційне повідомлення про неможливість повторного складання тесту після його завершення.

Для переходу до безпосереднього виконання завдань користувач має обов’язково підтвердити свою згоду з правилами, що мінімізує виникнення організаційних непорозумінь під час тестування.

3.8.4 Шаблон сторінки тестування (test.html)

Сторінка test.html є центральним інтерфейсом взаємодії, що забезпечує візуалізацію екзаменаційного процесу. Вона відповідає за динамічне відображення бази запитань та збір клієнтських відповідей для подальшого аналізу.

Програмна реалізація базується на ітераційному виведенні даних:

— Передача масиву: серверна частина Flask надсилає впорядкований список об'єктів із питаннями до шаблону.
— Динамічний рендеринг: за допомогою керуючих конструкцій (циклів) шаблонізатора Jinja2 система автоматично генерує необхідну кількість HTML-блоків, адаптуючи інтерфейс під обсяг конкретного тесту.
— Структура форми: кожне питання супроводжується інтерактивними елементами (радіокнопками або чекбоксами) для фіксації вибору користувача.

Лістинг 3.7 - Фрагмент шаблону сторінки тестування

```
<div class="container">
    <h2>Тестування</h2>

    <h3 id="timer">60:00</h3>

    <!--ПРОГРЕС-БАР -->
    <div class="progress-bar">
        <div id="progress"></div>
    </div>

    <form id="testForm" method="POST">
        <input type="hidden" name="time_spent" id="time_spent">
      <input type="hidden" name="violations" id="violations_input">

        {% for q in questions %}
        {% set q_index = loop.index0 %}

        <div class="question-block">

            <p class="question-title">
                {{ loop.index }}. {{ q.question }}
            </p>

            <!-- SINGLE -->
            {% if q.type == "single" %}
                {% for option in q.options %}
                    <label class="option">
                          <input type="radio" name="q{{ q_index }}"
value="{{ option }}">
                        {{ option }}
                    </label>
                {% endfor %}
            {% endif %}

            <!-- MULTIPLE -->
```

```
            {% if q.type == "multiple" %}
                {% for option in q.options %}
                    <label class="option">
                       <input type="checkbox" name="q{{ q_index }}"
value="{{ option }}">
                        {{ option }}
                    </label>
                {% endfor %}
            {% endif %}

            <!-- OPEN -->
            {% if q.type == "open" %}
                <input
                    type="text"
                    name="q{{ q_index }}"
                    class="open-input"
                    placeholder="Введіть відповідь..."
                >
            {% endif %}

            <!-- MATCHING -->
            {% if q.type == "matching" %}
                <p style="opacity:0.6;">(Питання на відповідність -
скоро буде)</p>
            {% endif %}

        </div>
        {% endfor %}

       <button type="button" id="finishBtn">Завершити тест</button>
    </form>
</div>

<!-- МОДАЛКА -->
<div id="confirmModal" class="modal">
    <div class="modal-content">
```

```
        <p>Ви впевнені, що бажаєте завершити тест?</p>

        <button id="confirmSubmit">Завершити</button>
        <button id="cancelBtn" class="cancel">Повернутися</button>
    </div>
</div>

<!-- МОДАЛКА ПОРУШЕННЯ -->
<div id="warningModal" class="modal">
    <div class="modal-content">
        <p id="warningText"></p>

        <button onclick="submitForm()">Завершити тест</button>
                                <button    onclick="closeWarning()"
class="cancel">Повернутися</button>
    </div>
</div>
```

Шаблон test.html реалізує комплексний функціонал для проведення сесії оцінювання:

- Динамічне виведення: візуалізація повного переліку сформованих питань.
- Варіативність вибору: відображення відповідних варіантів відповідей для кожного завдання.
- Інтерактивність: надання користувачеві інструментарію для фіксації обраних варіантів.
- Синхронізація з сервером: безпечна передача накопиченого масиву відповідей для обробки після натискання кнопки завершення.
- Система сповіщень: виведення попереджень у разі порушення встановленого регламенту (наприклад, спроби покинути сторінку).
- Валідація завершення: відображення модального вікна для підтвердження наміру користувача закінчити тест, що запобігає випадковому перериванню сесії.

3.8.5 Шаблон сторінки результатів (result.html)

По завершенню сесії система перенаправляє користувача до інтерфейсу result.html, де консолідуються підсумкові показники успішності.

На цій сторінці візуалізуються такі метрики:

— Кількісний показник: точна кількість успішно розв'язаних завдань.
— Масштаб тесту: загальна кількість питань у пройденому модулі.
— Відсоткова частка: рівень успішності, обчислений відносно загального обсягу завдань.
— Таймінг: тривалість поточної тестової сесії.

Додатково інтерфейс містить елементи навігації, що дозволяють перейти до детального аналізу допущених помилок у модулі перегляду або повернутися до стартового екрана системи.

3.8.6 Шаблон сторінки перегляду відповідей (review.html)

Інтерфейс review.html призначений для ретроспективного аналізу результатів тестування. Цей модуль забезпечує прозорість оцінювання, надаючи респонденту повний доступ до деталізації кожної відповіді.

Для кожного пункту тесту на сторінці візуалізується:

— Форлювання питання: повний текст завдання для контексту.
— Перелік варіантів: усі опції, що були доступні під час проходження.
— Еталонна відповідь: маркування варіанта, який система вважає правильним.
— Вибір респондента: чітка ідентифікація відповіді, наданої користувачем (з кольоровим виділенням для позначення помилок або збігів).

Такий функціонал є критично важливим для освітнього процесу, оскільки дозволяє користувачу провести роботу над помилками, зіставити власну логіку з правильними варіантами та підвищити рівень засвоєння матеріалу.

3.8.7 Шаблон панелі адміністратора (admin_dashboard.html)

Управління програмним комплексом здійснюється через спеціалізовану

панель модерації, реалізовану в шаблоні admin_dashboard.html. Доступ до цього модуля суворо обмежений і надається виключно користувачам із відповідним рівнем привілеїв.

Інтерфейс адміністратора містить повний набір інструментів для маніпуляції контентом, що включає:

— Моніторинг бази: перегляд актуального переліку всіх тестових завдань.
— Наповнення системи: можливість інтеграції нових запитань.
— Коригування: внесення змін до тексту питань або варіантів відповідей.
— Оптимізація: видалення неактуальних або помилкових записів із реляційного сховища.

Така централізована модель керування дозволяє оперативно оновлювати екзаменаційні матеріали без втручання в програмний код системи.

3.8.8 Шаблон керування питаннями (admin_questions.html)

Модуль admin_questions.html виступає безпосереднім інтерфейсом взаємодії з контентом бази даних. Він забезпечує адміністратору повний цикл управління тестовими матеріалами.

Функціональні можливості сторінки дозволяють:

— Аудит контенту: візуалізацію повного переліку запитань, зареєстрованих у системі.
— Модифікацію текстів: оперативне виправлення формулювань або уточнення умов завдань.
— Коригування відповідей: зміну доступних варіантів та призначення правильних ключів.
— Очищення бази: вилучення застарілих або нерелевантних записів.

Реалізація такого інтерфейсу відокремлює контент від програмної логіки, дозволяючи вносити зміни до структури тесту в режимі реального часу через веб-інтерфейс [19].

3.9 Реалізація контейнеризації системи

Для забезпечення стабільної роботи та швидкого розгортання програмного комплексу впроваджено технологію Docker. Контейнеризація дозволяє ізолювати додаток та його залежності від особливостей хостової операційної системи, створюючи уніфіковане середовище для виконання.

Використання Docker надає наступні переваги [10], [15]:

— Портативність: гарантія ідентичної роботи системи на етапах розробки, тестування та фінального продакшену.

— Швидке розгортання: автоматизація процесу інсталяції всіх необхідних бібліотек та компонентів (Python, Flask, SQLAlchemy).

— Масштабованість: можливість легко запускати кілька екземплярів додатка для розподілу навантаження.

— Ізоляція: мінімізація конфліктів між системними налаштуваннями та програмними залежностями проєкту.

Завдяки наявності Dockerfile та файлів конфігурації, розгортання повної інфраструктури системи зводиться до виконання кількох стандартних команд, що значно спрощує технічну підтримку продукту.

Лістинг 3.8 - docker-compose.yml

```
services:
  web:
    build: .
    expose:
      - "5000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/mydb

  db:
    image: postgres:15
```

```
    restart: always
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydb
    volumes:
      - postgres_data:/var/lib/postgresql/data

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
    depends_on:
      - web

volumes:
  postgres_data:
```

Конфігурація Docker передбачає поділ системи на два функціональні вузли, кожен з яких ізольований у власному контейнері:

— web: Містить Flask-додаток, усі необхідні Python-залежності та програмний код системи. Він відповідає за обробку HTTP-запитів та рендеринг інтерфейсу.

— db: Служить виділеним середовищем для розгортання та функціонування PostgreSQL [9]. Це забезпечує надійне зберігання даних та незалежність бази від життєвого циклу прикладного ПЗ.

3.10 Реалізація веб-сервера

Веб-сервер Nginx виступає як точка входу для зовнішнього трафіку та забезпечує стабільну роботу системи за рахунок наступного функціоналу [11], [19]:

— Маршрутизація трафіку: приймання вхідних HTTP-запитів від користувачів та їх коректний розподіл.
— Захист з'єднань: термінація SSL/TLS-сертифікатів для забезпечення безпечного передавання даних через протокол HTTPS.
— Реверс-проксі: проксіювання запитів до Gunicorn (або іншого WSGI-сервера), який обслуговує Flask-додаток, що дозволяє ізолювати прикладне ПЗ від прямої взаємодії з інтернетом.
— Оптимізація ресурсів: кешування статичного контенту та балансування навантаження, що суттєво підвищує загальну продуктивність і відмовостійкість системи.

3.11 Реалізація генерації PDF-звіту

Функціонал системи передбачає автоматичну генерацію звітів у форматі PDF для документування результатів тестування. Технічна реалізація цього модуля базується на можливостях бібліотеки ReportLab.

Процес формування звіту включає:

— Збір даних: вилучення підсумкових показників користувача (ПІБ, бали, дата) з бази даних.
— Динамічна верстка: програмне створення структури документа, включаючи заголовки, таблиці з результатами та графічні елементи.

— Рендеринг: перетворення сформованих об'єктів Python у двійковий потік даних формату PDF.
— Видача файлу: надання користувачеві можливості завантаження готового документа безпосередньо через браузер.

Використання ReportLab забезпечує високу швидкість генерації звітів та дозволяє точно налаштувати візуальне оформлення документів відповідно до встановлених стандартів.

Лістинг 3.9 - Генерація PDF-звіту

```
pdf = SimpleDocTemplate(file_path)

elements.append(Paragraph("Звіт тестування", title_style))
elements.append(Paragraph(f"ПІБ: {fullname}", header_style))

pdf.build(elements)
```

Цей фрагмент коду відповідає за:

— створення PDF-документа;

— додавання текстових елементів;

— формування фінального файлу звіту.

3.12 Реалізація додаткових механізмів функціонування системи

Окрім базового функціоналу, архітектура системи включає допоміжні модулі, що забезпечують комплексну роботу сервісу та його безпеку. Кожен компонент розв'язує специфічні завдання:

— Система авторизації: Відповідає за ідентифікацію користувачів та розмежування прав доступу (користувач/адміністратор). Це гарантує персоналізацію результатів та захист конфіденційної інформації.

— Адміністративна панель: Надає графічний інтерфейс для керування контентом бази даних (CRUD-операції над питаннями) без необхідності прямого втручання в структуру SQL-таблиць.
— Алгоритм рандомізації: Забезпечує випадковий порядок виведення завдань для кожного нового тестування. Це підвищує об'єктивність оцінювання та мінімізує можливість списування.
— Тайм-менеджмент: Механізм контролю часових лімітів, який автоматично фіксує тривалість сесії та може обмежувати час на надання відповідей, що критично для стандартизованих іспитів.

Нижче наведено детальний технічний опис реалізації кожного з цих механізмів.

3.12.1 Реалізація системи авторизації

Механізм авторизації в системі базується на використанні сесій Flask, що забезпечує безперервність користувацького досвіду без необхідності повторного введення даних на кожному етапі.

Технічні особливості реалізації:

— Ідентифікація: Дані зберігаються на стороні сервера (або у зашифрованих клієнтських cookies), що дозволяє системі «впізнавати» користувача при переході між маршрутами (/rules, /test, /result).
— Конфіденційність: Для захисту сесій використовується секретний ключ (secret_key), який гарантує цілісність даних та запобігає їх підробці.

Склад об'єкта сесії:

— Персональні дані: електронна адреса, ім'я та прізвище для персоналізації інтерфейсу та звітів.
— Динамічні дані: поточні відповіді користувача, що накопичуються під час тестування.
— Підсумкові дані: результати оцінювання, що передаються на фінальний екран та до модуля генерації PDF.

Використання сесій дозволяє коректно обробляти стан тестування навіть у разі випадкового перезавантаження сторінки користувачем.

Приклад використання сесії наведено у лістингу 3.10.

Лістинг 3.10 - Збереження даних користувача у сесії

```
def register():
    if request.method == 'POST':
        email = request.form.get("email")
        fullname = request.form.get("fullname")

        if not email.endswith("@nubip.edu.ua"):
             return render_template("register.html", error="Введіть університетську пошту (домен має бути @nubip.edu.ua)", email=email, fullname=fullname)

        session['email'] = email
        session['fullname'] = fullname
        session.pop('questions', None)
        return redirect(url_for('rules'))
```

Перед переходом на сторінку тестування, система перевіряє наявність домену @nubip.edu.ua у пошті користувача, у разі відсутності необхідного домену, користувача не авторизує і повідомить про не виконані умови для авторизаії.

Такий механізм дозволяє обмежити доступ до тесту.

3.12.2 Реалізація адміністративної панелі

Для управління контентом системи розроблено адміністративний модуль, який базується на принципах CRUD (Create, Read, Update, Delete). Це дозволяє модераторам підтримувати актуальність тестової бази без втручання в архітектуру бази даних через сторонні утиліти.

Технічна реалізація адміністративного режиму:

— Маршрутизація: Використання окремих Flask-endpoint-ів (наприклад, /admin, /admin/add, /admin/edit/<id>, /admin/delete/<id>) для розмежування логіки управління.

— Інтерфейсна частина: Спеціалізовані шаблони admin_dashboard.html та admin_questions.html, що забезпечують зручну візуалізацію таблиць із даними.

Функціональні можливості:

— Перегляд: Формування повного переліку запитань із відображенням варіантів відповідей та правильних ключів.

— Додавання: Інтерактивна форма для введення тексту нового питання та відповідних варіантів.

— Редагування: Можливість миттєвого виправлення помилок або оновлення застарілих даних у вже існуючих записах.

— Видалення: Механізм очищення бази від неактуальних тестових завдань.

Така структура забезпечує повний контроль над змістовною частиною тестування, гарантуючи гнучкість та масштабованість системи.

Приклад маршруту адміністративної сторінки наведено у лістингу 3.11.

Лістинг 3.11 - Маршрут панелі адміністратора

```
@app.route('/admin/questions')
def admin_questions():
    if 'admin' not in session:
        return redirect('/admin/login')

    questions = Question.query.all()
                return    render_template("admin_questions.html",
questions=questions)
```

Для забезпечення ефективного менеджменту контенту інтерфейс адміністратора використовує табличну структуру представлення даних. Це рішення дозволяє систематизувати великі обсяги інформації та забезпечує високу швидкість навігації.

Завдяки відокремленню контенту від програмної логіки, адміністратор отримує інструмент для налаштування змісту тестування. Це виключає ризик

внесення помилок у вихідний код при необхідності простої актуалізації питань.

3.12.3 Реалізація алгоритму випадкового перемішування питань

Для забезпечення академічної доброчесності та мінімізації ризику копіювання відповідей, у системі впроваджено механізм динамічної рандомізації контенту.

Технічна реалізація алгоритму:

— Екстракція даних: Система виконує запит до бази даних і отримує повний набір питань, визначених для конкретного тесту.

— Формування масиву: Отримані об'єкти записуються у структуру даних «список» (list).

— Рандомізація: За допомогою методу random.shuffle() бібліотеки random, порядок елементів у списку змінюється випадковим чином перед кожною новою сесією.

Приклад реалізації наведено у лістингу 3.12.

Лістинг 3.12 - Перемішування питань тесту

```
# Одноразово генеруємо питання
    if 'questions' not in session:
        db_questions = Question.query.all()
        shuffled_questions = []

        for q in db_questions:
            question_data = {
                "id": q.id,
                "question": q.text,
                "type": q.type
            }
 shuffled_questions.append(question_data)
        # перемішуємо 1 раз
        random.shuffle(shuffled_questions)
        # зберігаємо в session
```

```
session['questions'] = shuffled_questions
```

Таким чином, комбінація динамічного перемішування та сесійної фіксації гарантує одночасно і унікальність кожного тесту, і технічну стабільність процесу оцінювання.

3.12.4 Реалізація механізму контролю часу тестування

Контроль часових лімітів є важливою складовою об'єктивного оцінювання, що реалізується через комбінацію клієнтських та серверних технологій.

Програмний механізм контролю часу функціонує за наступним алгоритмом:

— Клієнтська логіка (JavaScript): Таймер запускається безпосередньо у браузері користувача після завантаження сторінки test.html. Це забезпечує плавне відображення зворотного відліку в реальному часі без необхідності постійного звернення до сервера.

— Фіксація даних: Весь час, витрачений на відповіді, акумулюється скриптом. Перед відправкою форми JavaScript записує фінальне значення у приховане поле введення: <input type="hidden" name="time_spent" id="time_spent">.

— Серверна обробка (Flask): Після натискання кнопки «Завершити тест» дані з прихованого поля передаються разом із відповідями методом POST. Сервер зчитує цей параметр та зберігає його у відповідному стовпці таблиці результатів у базі даних PostgreSQL.

3.13 Висновок до розділу

Цей розділ підбиває підсумок технічної реалізації системи.

Опис охоплює повний цикл створення продукту:

— Архітектура: Організація проєкту за принципом розділення логіки (Flask), представлення (Jinja2) та збереження даних (PostgreSQL) [9].
— Технологічний стек: Використання Python як основної мови, Flask як мікрофреймворку та SQLAlchemy для взаємодії з базою даних.
— Логіка тестування: Реалізація бекенд-алгоритмів для рандомізації питань, валідації доменної пошти (@nubip.edu.ua) та миттєвої перевірки відповідей.
— Інтерфейс та UX: Створення адаптивних HTML-шаблонів, що включають таймери на JavaScript для контролю часу та зручні форми для реєстрації та перегляду помилок.
— Інфраструктура: Застосування Docker для створення ізольованих контейнерів, що гарантує ідентичність робочого середовища на будь-якому сервері.
— Звітність: Впровадження модуля ReportLab для автоматичного перетворення цифрових результатів у офіційні PDF-документи.

Результатом є масштабована та надійна комп’ютерна система, яка автоматизує процес оцінювання знань, забезпечує високу швидкість обробки даних та надає адміністратору гнучкі інструменти для керування контентом.

4 ТЕСТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Мета та завдання тестування

Тестування розробленої системи є фінальним етапом контролю якості, що підтверджує готовність продукту до експлуатації. Використання тестових сценаріїв дозволяє верифікувати кожен програмний модуль на відповідність технічним вимогам.

Процес тестування охопив усі критичні вузли системи:

— Автентифікація та доступ: Верифікація форми реєстрації та алгоритму перевірки доменної пошти. Перевірено відмову в доступі для адрес, що не належать до домену @nubip.edu.ua.

— Функціонал тестування: Контроль коректності виведення питань у випадковому порядку, роботи таймера на клієнтській стороні та стабільності сесії при оновленні сторінок.

— Обробка даних: Перевірка алгоритму порівняння відповідей користувача з еталонними значеннями та точність розрахунку відсоткового показника успішності.

— Цілісність БД: Підтвердження успішного запису результатів (ПІБ, бали, час) у відповідні таблиці PostgreSQL.

— Модерація: Тестування CRUD-операцій в адміністративній панелі (створення, редагування та видалення питань) і розмежування прав доступу.

— Документування: Валідація структури та змісту згенерованого PDF-файлу на відповідність фактичним результатам тестування.

Тестування проводилося за методом чорної скриньки (Black Box Testing), де основна увага приділялася вхідним даним та очікуваним результатам [5], [17]. Виконання імітаційних сценаріїв підтвердило стабільність інтерфейсу та відсутність критичних помилок у логіці додатка, що свідчить про повну реалізацію функціонала, передбаченого технічним завданням.

4.2 Середовище тестування системи

Для проведення випробувань та верифікації функціоналу було використано стандартизоване програмне середовище, що забезпечило коректну роботу всіх компонентів системи [5].

Основні характеристики середовища тестування наведено в таблиці 4.1.

Таблиця 4.1 - Середовище тестування системи

Компонент	Характеристика
Операційна система	Windows 10
Процесор	Intel Core i5
Оперативна пам’ять	8 ГБ
Мова програмування	Python 3
Веб-фреймворк	Flask
Система керування базами даних	PostgreSQL
Веб-сервер	Nginx
Система контейнеризації	Docker
Веб-браузер	Google Chrome
Компонент	Характеристика
Операційна система	Windows 10

Використання Docker для розгортання системи забезпечило повну ізоляцію компонентів та стабільність програмного оточення. Це дозволило уникнути проблеми «працює лише на моїй машині», гарантуючи ідентичність середовища розробки та виконання [10].

4.3 Тестування функціональних можливостей системи

Функціональне тестування було спрямоване на верифікацію того, що кожен програмний модуль виконує завдання, визначені в технічних вимогах. Результати тестування наведено в таблиці 4.2.

Таблиця 4.2 - Функціональне тестування системи

№	Сценарій тестування	Вхідні дані	Очікуваний результат	Фактичний результат
1	Відкриття сторінки реєстрації	URL системи	відкривається сторінка реєстрації	успішно
2	Введення даних користувача	ПІБ, email	дані приймаються системою	успішно
3	Перехід до правил тестування	натискання кнопки	відкривається сторінка правил	успішно
4	Підтвердження правил	натискання кнопки	відкривається сторінка тесту	успішно
5	Відображення питань тесту	список питань	питання відображаються	успішно
6	Вибір відповіді	варіант відповіді	відповідь зберігається	успішно
7	Завершення тесту	натискання кнопки	відкривається сторінка результату	успішно
8	Перегляд результату	-	відображається результат	успішно
9	Перегляд відповідей	натискання кнопки	відображається review	успішно

Результати тестування показали, що всі основні функції системи працюють вірно.

4.4 Тестування інтерфейсу користувача

Оцінювання користувацького інтерфейсу (UI) проводилося з метою підтвердження високої якості взаємодії користувача з платформою (UX). Система розроблялася з урахуванням того, що інтерфейс має бути інтуїтивно зрозумілим навіть для недосвідченого користувача.

- Зручність використання: Елементи керування (кнопки, поля введення) розташовані логічно. Форми реєстрації та вибору відповідей мають чіткі підписи, що мінімізує когнітивне навантаження на респондента.
- Зрозумілість навігації: Переходи між етапами тестування відбуваються послідовно. Користувач завжди розуміє свій поточний стан завдяки інформативним заголовкам та системі сповіщень.
- Коректність відображення: Використання шаблонізатора Jinja2 у поєднанні з Bootstrap забезпечує стабільну візуалізацію всіх компонентів. Текст питань, варіанти відповідей та таблиці в адмін-панелі відображаються без накладання елементів або порушення верстки.
- Адаптивність: Завдяки використанню сітки Bootstrap, сторінки коректно підлаштовуються під різні розміри екранів. Це дозволяє проходити тестування як з персональних комп'ютерів, так і з мобільних пристроїв без втрати функціональності.

Результати тестування наведено у таблиці 4.3.

Таблиця 4.3 - Тестування інтерфейсу користувача

№	Елемент інтерфейсу	Дія	Очікуваний результат	Результат
1	форма реєстрації	введення даних	форма працює коректно	успішно
2	кнопка початку тесту	натискання	відкривається тест	успішно
3	відображення питань	перегляд	питання відображаються	успішно
4	кнопка завершення тесту	натискання	відкривається результат	успішно
5	підтвердження завершення тесту	-	відкривається модалка з підтвердженням	успішно

4.5 Тестування механізму тестування користувачів

Процес тестування функціонального модуля проведення іспиту був спрямований на підтвердження точності алгоритмів взаємодії користувача з контентом.

— Коректність відображення питань: Перевірено роботу шаблонізатора Jinja2 при виведенні даних із бази. Питання відображаються у повному обсязі, без спотворень тексту чи кодування. Алгоритм рандомізації успішно змінює послідовність запитань для кожної нової сесії.

— Правильність вибору відповідей: Верифіковано роботу HTML-форм та атрибутів value. Кожен вибір користувача через кнопки коректно

фіксується та передається у запиті POST без втрати даних або підміни варіантів.

— Коректність перевірки відповідей: Проведено аудит серверної логіки Python. Система безпомилково зіставляє ID вибраної відповіді з еталонним значенням (вірними), що зберігається в таблиці бази даних. Розрахунок підсумкової оцінки та відсотка успішності відповідає математичній моделі оцінювання.

Тестування підтвердило, що механізм проведення іспиту працює синхронно на всіх етапах - від завантаження сторінки test.html до фінальної обробки даних сервером.

Результати тестування наведено в таблиці 4.4.

Таблиця 4.4 - Тестування механізму тестування

№	Тип питання	Дія	Очікуваний результат	Результат
1	single	вибір одного варіанта	відповідь зберігається	успішно
2	multiple	вибір кількох варіантів	відповіді зберігаються	успішно
3	open	введення тексту	відповідь перевіряється	успішно

4.6 Тестування роботи бази даних

Для перевірки роботи бази даних виконувались операції збереження та отримання даних.

Таблиця 4.5 - Тестування бази даних

№	Операція	Очікуваний результат	Результат
1	збереження результату	дані записуються у БД	успішно
2	отримання результату	дані читаються з БД	успішно
3	збереження відповіді	відповідь зберігається	успішно

4.7 Тестування адміністративної панелі

Адміністративна панель дозволяє керувати спробами і питаннями тесту. Виконано перевірки основних функцій, наведено у таблиці 4.6.

Таблиця 4.6 - Тестування адміністративної панелі

№	Дія	Очікуваний результат	Результат
1	відкриття панелі	відкривається сторінка	успішно
2	додавання питання	питання додається	успішно
3	редагування питання	зміни зберігаються	успішно
4	видалення питання	питання видаляється	успішно
5	видалення спроб проходження	спроби видаляються	успішно

4.8 Тестування формування PDF-звіту

Система дозволяє формувати PDF-звіти результатів тестування. Перевірено відповідні сценарії, наведено у таблиці 4.7.

Таблиця 4.7 - Тестування PDF-звіту

№	Дія	Очікуваний результат	Результат
1	натискання кнопки PDF	створюється файл	успішно
2	відкриття файлу	файл відкривається	успішно
3	відображення даних	результати присутні	успішно

4.9 Аналіз результатів тестування

Аналіз результатів фінальних випробувань підтверджує технічну зрілість та експлуатаційну готовність системи. Успішне проходження всіх контрольних сценаріїв свідчить про цілісність архітектури та коректність взаємодії між фронтендом, бекендом та базою даних.

— Інтерфейсна стабільність: Підтверджено коректність візуалізації шаблонів на базі Jinja2 та адаптивність верстки Bootstrap. Навігація між реєстрацією, тестуванням та результатами працює без затримок.

— Логіка оцінювання: Алгоритми рандомізації питань та автоматичної перевірки відповідей продемонстрували стовідсоткову точність. Таймер на JavaScript синхронно передає дані для фіксації тривалості сесії.

— Цілісність даних: Перевірка бази даних PostgreSQL показала, що всі записи про користувачів та їхні бали зберігаються без втрат і доступні для миттєвого відображення в адмін-панелі.

— Адміністративний менеджмент: Модуль керування контентом дозволяє безпечно виконувати всі CRUD-операції, забезпечуючи актуалізацію тестових матеріалів у реальному часі.
— Документообіг: Модуль ReportLab успішно генерує PDF-звіти, які коректно відображають кириличні символи та статистичні дані сесії.

Відсутність критичних багів та повна відповідність функціоналу вимогам технічного завдання дозволяють рекомендувати систему до впровадження в освітній процес. Розроблений програмний продукт є стабільним інструментом для проведення автоматизованого контролю знань.

4.10 Демонстрація роботи комп’ютерної системи

Для наочності роботи розробленої комп’ютерної системи наведено приклади основних сторінок інтерфейсу користувача. Демонстрація інтерфейсу дозволяє показати принцип взаємодії користувача з системою, а також підтверджує коректність реалізації основних функцій програмного забезпечення.

4.10.1 Сторінка реєстрації користувача

Початковою сторінкою системи є сторінка реєстрації користувача. На цій сторінці користувач вводить свої персональні дані, зокрема ім’я та адресу електронної пошти.

Після введення необхідної інформації користувач може перейти до наступного етапу роботи системи - ознайомлення з правилами тестування.
Зовнішній вигляд сторінки реєстрації наведено на рисунку 4.1.

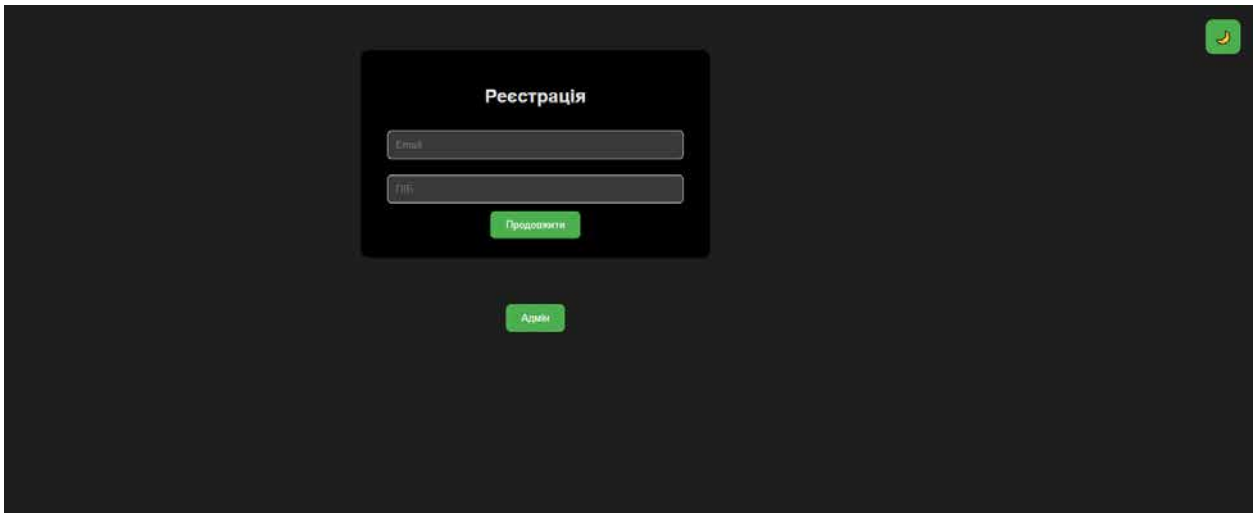

Рисунок 4.1 - Сторінка реєстрації користувача

4.10.2 Сторінка проходження тестування

Сторінка test.html є функціональним ядром системи, де зосереджено основні інструменти взаємодії респондента з екзаменаційним контентом. Її інтерфейс розроблений таким чином, щоб забезпечити безперервний процес оцінювання та надати користувачеві повний контроль над своїми діями.

— Мультимодальність відповідей: Система підтримує як закриті питання, так і відкриті питання, що дозволяє використовувати тест для перевірки знань різного рівня складності.
— Інтерактивний перегляд: Питання структуровані у зручний список, згенерований за допомогою Jinja2, що забезпечує чітку візуальну ієрархію між текстом завдання та опціями відповідей.
— Часовий контроль: Динамічний таймер, реалізований на JavaScript, виконує роль візуального індикатора, допомагаючи користувачеві ефективно розподіляти час на виконання завдань.
— Завершення сесії: Кнопка завершення ініціює передачу масиву даних на сервер. Перед фінальною відправкою система викликає вікно підтвердження, щоб уникнути передчасного закінчення тесту.

Зовнішній вигляд сторінки тестування наведено на рисунку 4.2.

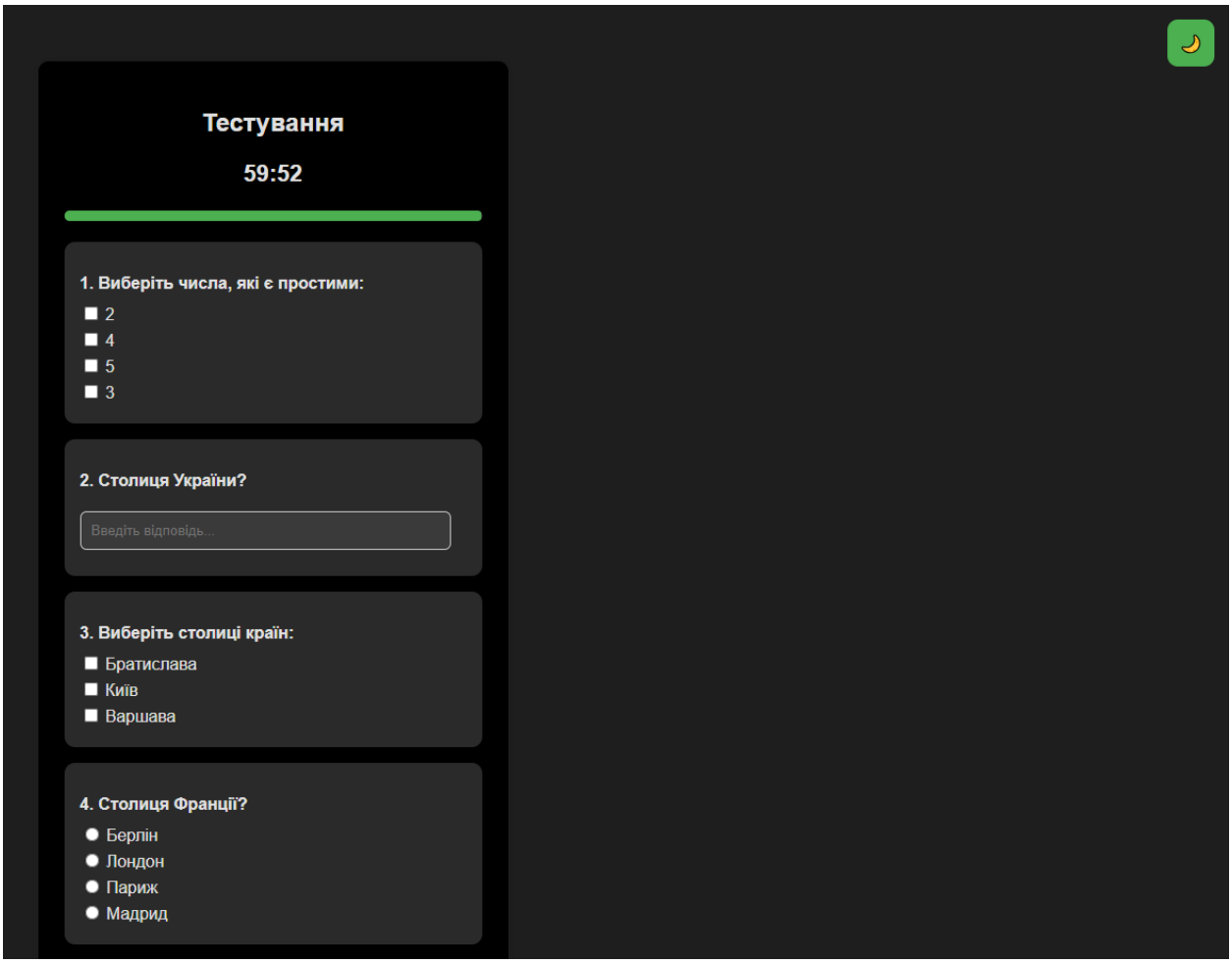

Рисунок 4.2 - Сторінка проходження тестування

4.10.3 Сторінка результату тестування

Після отримання даних від клієнта, серверна частина на базі Flask виконує фінальну обробку сесії. Сторінка результатів (result.html) є підсумковим етапом, де користувач отримує вичерпну інформацію про свою успішність.

Система виводить дані у структурованому вигляді, що дозволяє швидко оцінити рівень знань:

— Кількісні показники: Відображається співвідношення правильних відповідей до загальної кількості запитань (наприклад, 18/20). Це дає базове розуміння обсягу засвоєного матеріалу.

— Відсотковий рейтинг: Автоматично розрахований відсоток успішності, який часто є ключовим критерієм для виставлення підсумкової оцінки за іспит.

— Часова статистика: Відображається чистий час, витрачений на тест. Ці дані беруться з прихованого поля форми, яке заповнював JavaScript-таймер.
— Контроль доброчесності: Окремим пунктом виводиться кількість зафіксованих порушень (наприклад, спроби виходу зі сторінки або перемикання вкладок, якщо така логіка була активована), що впливає на об’єктивність результату.

Вся ця інформація одночасно записується до бази даних PostgreSQL, що гарантує збереження історії тестування для подальшого аудиту адміністратором або генерації офіційного PDF-звіту.

Зовнішній вигляд сторінки результату тестування представлено на рисунку 4.3.

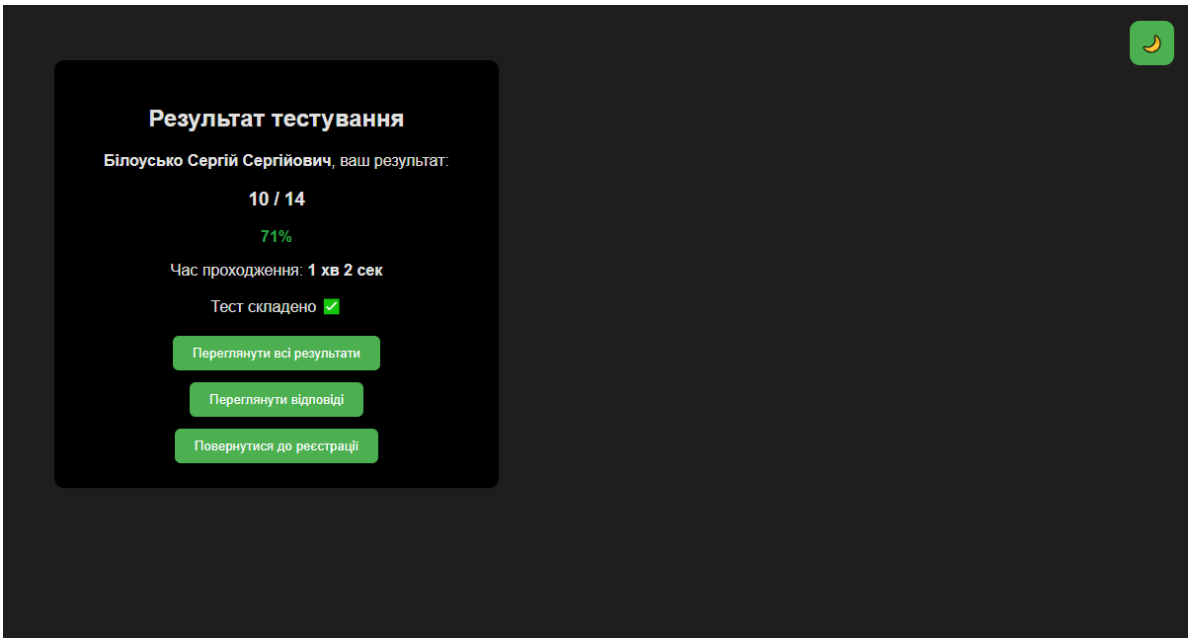

Рисунок 4.3 - Сторінка результату тестування

4.10.4 Сторінка перегляду всіх результатів тестування

Сторінка моніторингу результатів є ключовим інструментом аналітики, що дозволяє систематизувати дані про успішність усіх респондентів. Табличне представлення забезпечує високу щільність інформації та зручність її сприйняття.

Використання динамічної таблиці дозволяє виконувати комплексний аудит процесу тестування:

— Ідентифікація: Чітке прив'язування результатів до конкретного користувача через ПІБ та електронну адресу (домен @nubip.edu.ua).
— Порівняльний аналіз: Завдяки відображенню балів та відсотків в єдиному реєстрі, адміністратор може легко порівнювати успішність різних користувачів або груп.
— Оцінка складності: Аналіз часу проходження тесту у поєднанні з кількістю правильних відповідей допомагає визначити, наскільки складними є питання для цільової аудиторії.
— Валідація результатів: Наявність колонки з кількістю порушень дозволяє швидко відфільтрувати сумнівні результати, де могла бути порушена академічна доброчесність.
— Сортування результатів: Для зручності будо додано можливість сортувати табличку за різними показниками, а також пошук за ПІБ.

Зовнішній вигляд сторінки перегляду результатів тестування представлено на рисунку 4.4.

Рисунок 4.4 - Сторінка перегляду результатів тестування

4.10.5 Сторінка перегляду відповідей користувача

Окрім загального результату тестування система також дозволяє переглянути детальну інформацію про відповіді користувача на кожне питання

тесту.

На сторінці перегляду відповідей відображаються:

— текст кожного питання;

— варіанти відповідей;

— відповідь, обрана користувачем;

— правильна відповідь.

При цьому система використовує кольорове позначення для зручності аналізу результатів:

— правильні відповіді виділяються зеленим кольором;

— неправильні відповіді позначаються червоним кольором.

Такий підхід дозволяє користувачу швидко визначити, у яких саме питаннях були допущені помилки.

Зовнішній вигляд сторінки перегляду відповідей представлено на рисунку 4.5.

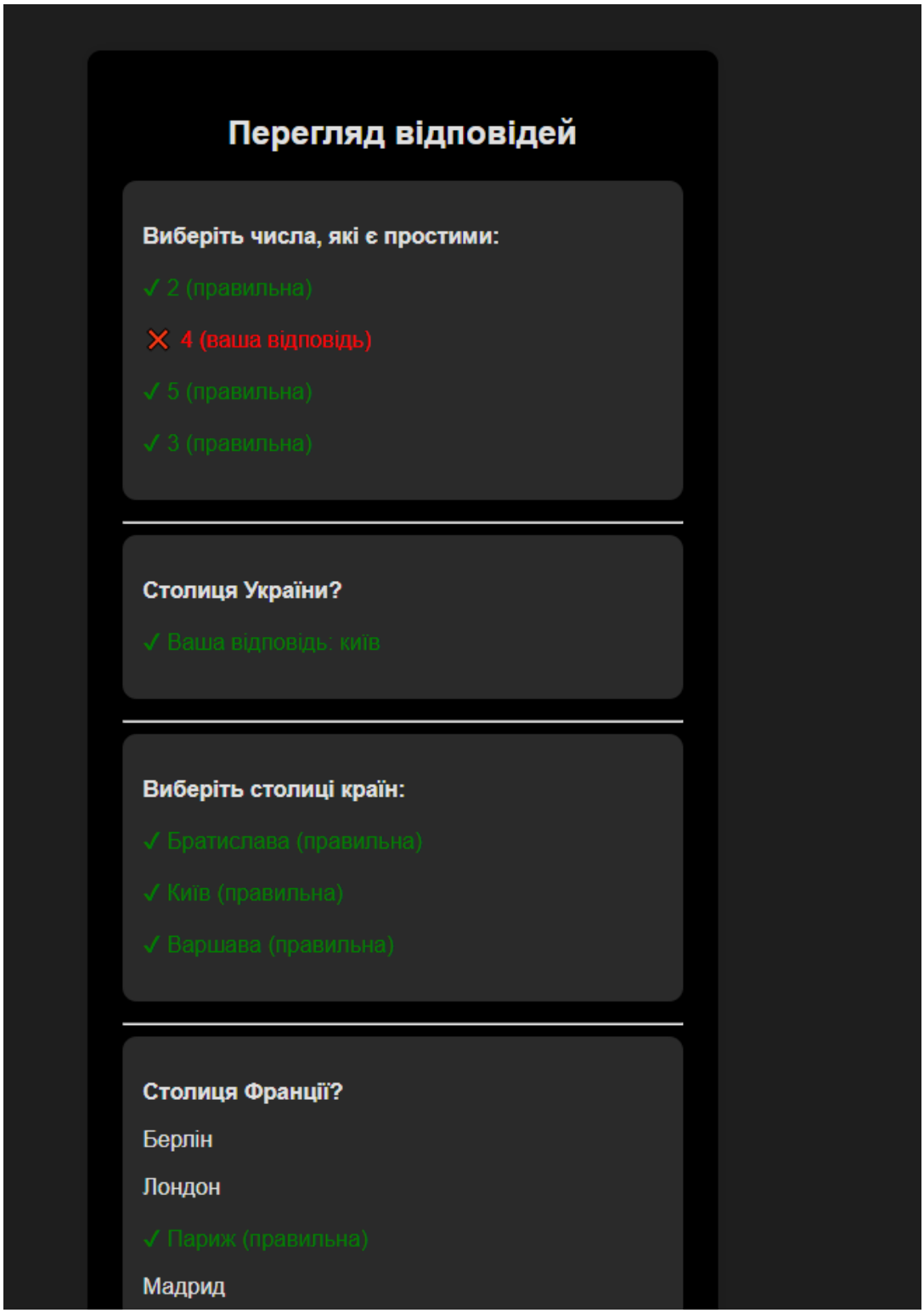

Рисунок 4.5 - Сторінка перегляду відповідей користувача

4.10.6 Адміністративна панель системи

Адміністративна панель є ключовим інструментом для підтримки актуальності навчального контенту. Вона реалізована як захищений сегмент системи, що надає повний контроль над життєвим циклом тестових завдань.

Робота адміністратора базується на чотирьох базових операціях:

— Моніторинг: Централізована таблиця відображає всі наявні в базі запитання, їхні типи та правильні відповіді. Це дозволяє швидко проводити аудит змісту тесту.

— Розширення бази: Форма додавання дозволяє вносити нові питання, варіанти відповідей та встановлювати правильні ключі. Дані автоматично валідуються перед записом у PostgreSQL.

— Коригування: Функція редагування забезпечує можливість виправлення помилок або оновлення застарілої інформації в існуючих записах без ризику порушити структуру бази даних.

— Очищення: Механізм видалення дозволяє вилучати неактуальні або некоректні питання, підтримуючи базу в чистоті та порядку.

Зовнішній вигляд адміністративної панелі представлено на рисунку 4.6.

Рисунок 4.6 - Адміністративна панель системи

Також на рисунку 4.7 представлено сторінку керування питаннями. Для адміністрування тестової системи реалізовано сторінку керування питаннями. Дана сторінка доступна лише для користувачів з правами адміністратора та

дозволяє виконувати основні операції з базою тестових питань.

За допомогою цієї сторінки адміністратор може:

— переглядати список усіх питань тесту;

— додавати нові питання;

— редагувати існуючі питання;

— видаляти питання з бази даних.

Сторінка керування питаннями відображає список усіх доступних питань. Для кожного питання передбачено відповідні елементи керування, які дозволяють виконувати редагування або видалення запису.

Використання такої сторінки значно спрощує процес адміністрування системи та дозволяє швидко змінювати зміст тесту без необхідності редагування програмного коду.

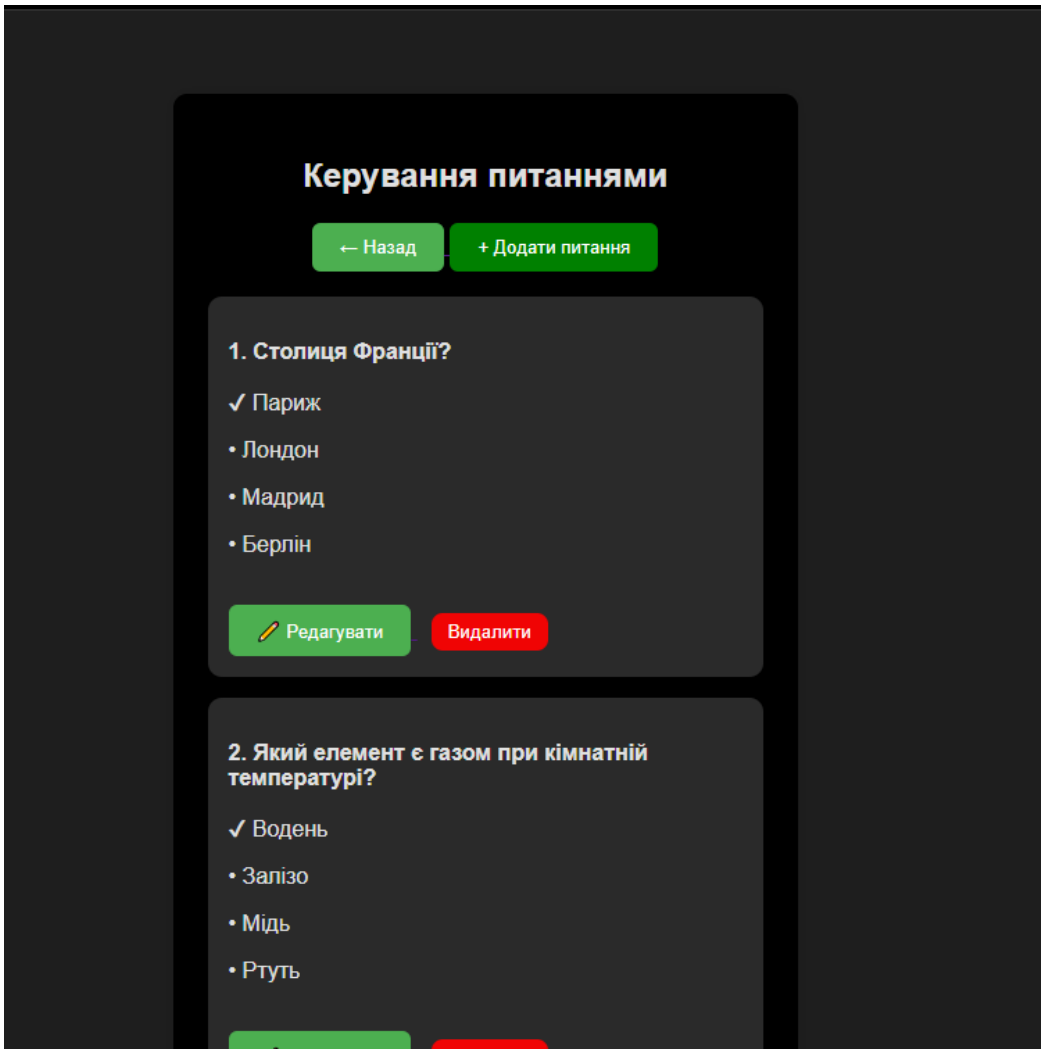

Рисунок 4.7 - Сторінка керування питаннями

4.10.7 Приклад сформованого PDF-звіту

Розроблена система дозволяє формувати PDF-звіт результатів тестування. Даний звіт містить інформацію про користувача, результати проходження тесту та список відповідей.

Формування звіту здійснюється після натискання відповідної кнопки на сторінці результатів.

Приклад сформованого PDF-документа наведено на рисунках 4.8 і 4.9.

Звіт результатів тестування

Дата: 08.04.2026 12:17

ID	Email	ПІБ	Бали	Всього	%	Час	Порушення	Статус
8	1@nubip.edu.ua	Білоуcько Сергій Сергійович	13	15	87%	1хв 53с	2	✔ Складено
9	2@nubip.edu.ua	Азар Мат Гат	11	15	73%	1хв 37с	0	✔ Складено
10	12@nubip.edu.ua	Білоуcько Сергій Сергійович	11	14	79%	3хв 20с	2	✔ Складено
11	3@nubip.edu.ua	Білоуcько Сергій Сергійович	10	14	71%	1хв 2с	1	✔ Складено

Рисунок 4.8 - Приклад сформованого PDF-звіту результатів

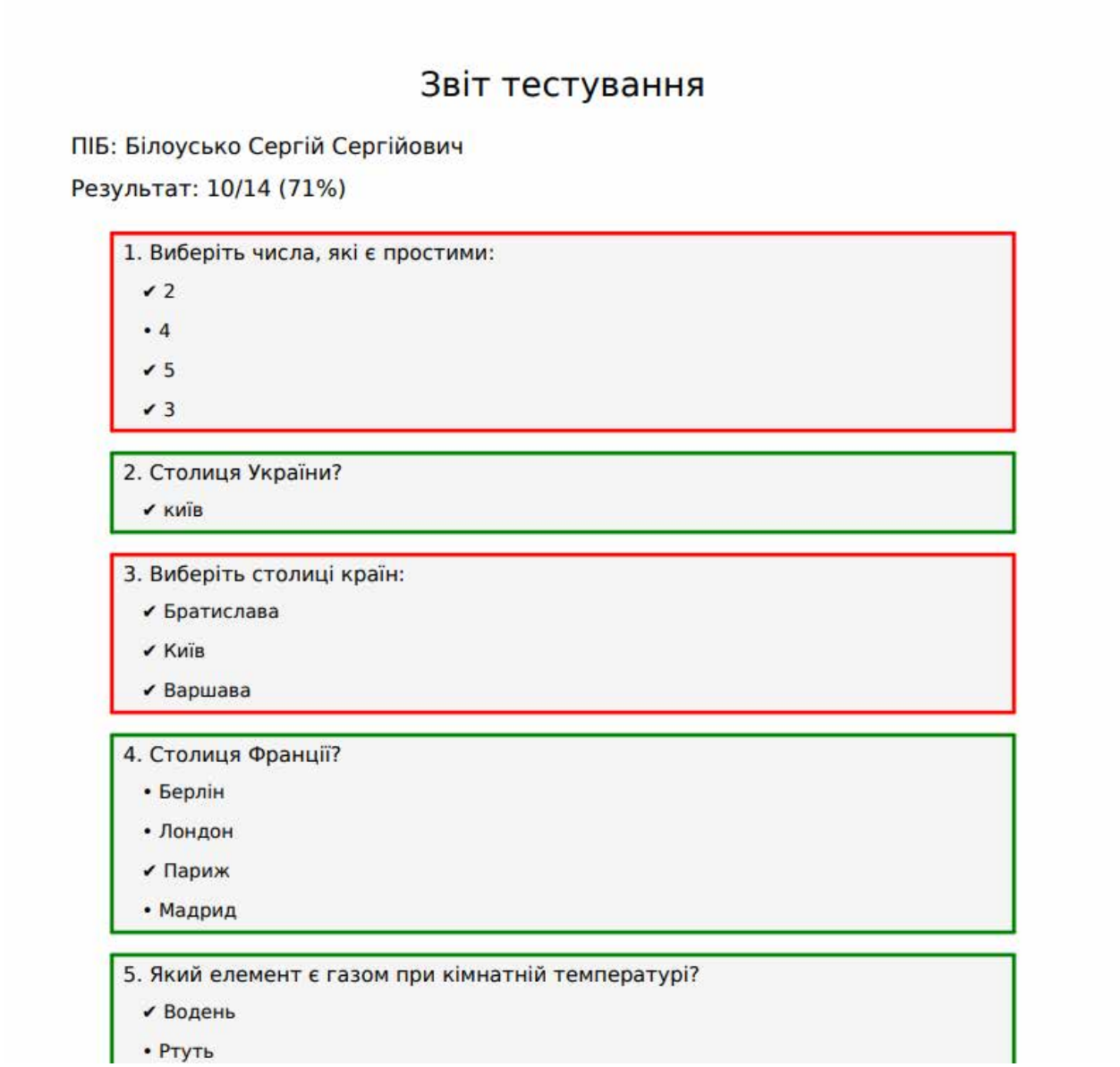

Рисунок 4.9 - Приклад сформованого PDF-звіту ревю

4.11 Висновок до розділу

Цей розділ фіксує успішне завершення етапу верифікації та валідації програмного продукту. Проведені випробування підтвердили, що всі компоненти системи - від фронтенд-інтерфейсу до серверної логіки та контейнеризованої інфраструктури - функціонують як єдине ціле.

Інтерфейс пройшов перевірку на інтуїтивність та адаптивність.

Користувачі мають стабільний доступ до функціоналу через браузер незалежно від типу пристрою. Механізми рандомізації, таймінгу та автоматичної перевірки відповідей працюють без збоїв, забезпечуючи об'єктивність оцінювання. Взаємодія з PostgreSQL продемонструвала високу швидкість обробки запитів та цілісність збереженої інформації про проходження тестів [9]. Панель керування підтвердила свою ефективність як інструмент для оперативного редагування контенту без втручання в сирцевий код. Модуль генерації звітів ReportLab стабільно формує PDF-файли, готові до завантаження та друку.

Розроблена система є повністю завершеним програмним рішенням, яке відповідає стандартам безпеки, надійності та зручності.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено комп'ютерну систему для проведення тестування користувачів. Основною метою роботи було створення програмного рішення, яке забезпечує автоматизоване проведення тестування, перевірку відповідей користувачів, збереження результатів та зручний перегляд отриманих даних.

У першому розділі було проведено аналіз технічного завдання та виконано огляд існуючих систем тестування. Розглянуто основні підходи до організації комп'ютерного тестування, визначено функціональні вимоги до системи та сформовано основні задачі, які необхідно реалізувати під час розроблення програмного забезпечення. Також було проаналізовано існуючі програмні рішення, їх переваги та недоліки.

У другому розділі виконано проєктування комп'ютерної системи. Було визначено загальну архітектуру системи, описано структуру бази даних та розроблено UML-діаграми, які відображають основні компоненти системи та взаємодію між ними. Проєктування дозволило сформувати чітке уявлення про структуру програмного забезпечення та логіку роботи системи.

У третьому розділі розглянуто процес реалізації комп'ютерної системи. Було описано структуру програмного проєкту, реалізовано серверну частину системи з використанням мови програмування Python та веб-фреймворку Flask. Для збереження результатів тестування використано систему керування базами даних. Також було реалізовано веб-інтерфейс користувача, адміністративну панель для керування питаннями тесту, механізм випадкового перемішування питань, контроль часу проходження тесту та формування PDF-звіту результатів тестування. Для розгортання системи використано технологію контейнеризації Docker, а також веб-сервер Nginx для обробки запитів.

У четвертому розділі було проведено тестування розробленої комп'ютерної системи. Виконано перевірку основних функціональних можливостей системи, інтерфейсу користувача, механізму тестування, роботи бази даних та адміністративної панелі. Результати тестування підтвердили

коректність роботи програмного забезпечення та відповідність системи поставленим вимогам.

У результаті виконаної роботи було створено працездатну комп’ютерну систему тестування користувачів, яка забезпечує проведення тестування, перевірку відповідей, збереження результатів у базі даних, перегляд детальної інформації про відповіді користувачів, а також формування звітів у форматі PDF.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Головач В. В. Основи програмування мовою Python : навчальний посібник. - Київ : Видавництво НТУУ «КПІ», 2020. - 320 с.

2. Лутц М. Вивчаємо Python. - Київ : Діалектика, 2021. - 1600 с.

3. Рамалхо Л. Python. К вершинам майстерності. - Київ : Діалектика, 2020. - 800 с.

4. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Прийоми об’єктно-орієнтованого проєктування. Патерни проєктування. - Київ : Вільямс, 2019. - 368 с.

5. Соммервіль І. Інженерія програмного забезпечення. - Київ : Вільямс, 2018. - 912 с.

6. Буч Г., Рамбо Дж., Якобсон А. UML. Керівництво користувача. - Київ : Діалектика, 2017. - 496 с.

7. Документація Flask [Електронний ресурс]. - Режим доступу: https://flask.palletsprojects.com

8. Документація Python [Електронний ресурс]. - Режим доступу: https://docs.python.org

9. Документація PostgreSQL [Електронний ресурс]. - Режим доступу: https://www.postgresql.org/docs

10. Документація Docker [Електронний ресурс]. - Режим доступу: https://docs.docker.com

11. Документація Nginx [Електронний ресурс]. - Режим доступу: https://nginx.org/en/docs

12. Grinberg M. Flask Web Development: Developing Web Applications with Python. - Sebastopol : O'Reilly Media, 2018. - 258 c.

13. Matthes E. Python Crash Course. - San Francisco : No Starch Press, 2019. - 544 c.

14. Beazley D., Jones B. Python Cookbook. - Sebastopol : O'Reilly Media, 2019. - 706 c.

15. Newman S. Building Microservices. - Sebastopol : O'Reilly Media, 2021. - 520 c.

16. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of

Reusable Object-Oriented Software. -Boston : Addison-Wesley, 2018. - 395 c.
17. Sommerville I. Software Engineering. - Boston : Pearson, 2016. - 816 c.
18. Bass L., Clements P., Kazman R. Software Architecture in Practice. - Boston : Addison-Wesley, 2021. - 448 c.
19. Welling L., Thomson L. PHP and MySQL Web Development. - Boston : Addison-Wesley, 2017. - 672 c.
20. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. - Boston : Addison-Wesley, 2018. - 208 c.