МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ ПРОВЕДЕННЯ ТЕСТОВИХ ОПИТУВАНЬ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Владислав БАГНЮК

«___»_____ 2025 р.

Керівник д-рка техн. наук, доцент _____ Ірина КАЛІНІНА

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Багнюка Владислава Юрійовича

((прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для проведення тестових опитувань».

Керівник роботи: Калініна Ірина Олександрівна, професор кафедри інтелектуальних інформаційних систем, д-рка техн. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «19» червня 2025 р.

3. Очікуваним результатом роботи є розробка сучасного вебзастосунку для організації, проведення та аналізу тестових опитувань у різних сферах (освіта, корпоративний сектор, дослідження).

4. Перелік питань, що підлягають розробці: аналіз сучасного стану задачі

вибору технології бездротової передачі даних в IoT мережах; огляд існуючих методів багатокритеріального прийняття рішень; експертне оцінювання технологій бездротової передачі даних в IoT-мережах за визначеними критеріями; порівняльний аналіз результатів застосування обраних методів багатокритеріального прийняття рішень для розв'язання поставленої задачі.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Ірина КАЛІНІНА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Владислав Багнюк

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для проведення тестових опитувань

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд аналогічних систем, щодо проведення тестових опитувань	31.01.2025	01.03.2025	Виконано
4	Огляд провідних систем та платформ для тестування	02.03.2025	01.04.2025	Виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	12.06.2025	12.06.2025	

Керівник роботи _____ Ірина КАЛІНІНА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Владислав БАГНЮК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 402 ЧНУ ім. Петра Могили

Багнюка Владислава Юрійовича

на тему: **«ВЕБЗАСТОСУНОК ДЛЯ ПРОВЕДЕННЯ ТЕСТОВИХ ОПИТУВАНЬ»**

У роботі застосовано сучасні методи веб-розробки, модульного та компонентного проектування, розробки REST API, реалізації клієнт-серверної архітектури, а також принципи UI/UX-дизайну. Для забезпечення масштабованості й надійності використовувались технології React, TypeScript, Node.js, MongoDB, та інші.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків і списку використаних джерел. У першому розділі здійснено аналіз предметної області, визначено основні вимоги до сучасних платформ для тестування, розглянуто зарубіжний і український досвід. У другому розділі описано архітектуру розробленої системи. Третій розділ присвячено практичній реалізації застосунку. У четвертому розділі наведено результати апробації системи, аналіз ефективності, описано виявлені переваги та недоліки.

Результатом роботи є прототип сучасної платформи для організації онлайн-тестування, що забезпечує повний цикл: від створення тестів до отримання детальної аналітики та вивантаження звітів.

Кваліфікаційна робота містить: 77 сторінок, 7 таблиць, 16 рисунків, 1 додатків, 33 використаних джерел.

Ключові слова: вебзастосунок, тестування, опитування, автоматизація, аналітика, інформаційна система, Python, React, TypeScript, Node.js, MongoDB.

ABSTRACT

to the qualification thesis
of student of group 402 of Petro Mohyla Black Sea National University

Bahniuk Vladislav

on the topic: **«WEB APPLICATION FOR CONDUCTING TEST SURVEYS»**

The work applies modern methods of web development, modular and component-based design, REST API development, client-server architecture implementation, and UI/UX design principles. Technologies such as React, TypeScript, Node.js, MongoDB are used to ensure scalability and reliability. Particular attention is paid to security, integration with external information systems, and automation of result processing and analytics.

The qualification paper consists of an introduction, four main chapters, conclusions, and a list of references. The first chapter analyzes the subject area, defines the main requirements for modern testing platforms, and reviews international and Ukrainian experience. The second chapter describes the system architecture, technology selection. The third chapter focuses on the practical implementation of the application. The fourth chapter presents the results of system testing, effectiveness analysis, identified advantages and shortcomings, and provides recommendations for further development.

The outcome is a prototype of a modern online testing platform that covers the full cycle: from test creation to advanced analytics and report generation.

The qualification paper includes: 77 pages, 7 tables, 16 figures, 1 appendices, and 33 references.

Keywords: web application, testing, surveys, automation, analytics, information system, Python, React, TypeScript, Node.js, MongoDB.

ЗМІСТ

ВСТУП

Актуальність полягає у зростаючій потребі в ефективних вебзастосунках для автоматизації процесу організації опитувань, що зумовлено цифровізацією освіти, поширенням дистанційного навчання та необхідністю оперативного збору й аналізу даних.

Об’єктом роботи є процес розробки, проведення та аналізу тестових опитувань у веб-середовищі

Предметом роботи є методи та інструменти розробки вебзастосунків для автоматизації процесу організації опитувань.

Метою роботи є підвищення ефективності вебзастосунків для автоматизації процесу організації опитувань.

Вебзастосунки для проведення тестових опитувань надають зручний інструментарій для створення динамічних тестів, автоматизованого збору відповідей у реальному часі та гнучкої візуалізації результатів. Згідно з дослідженнями MDPI, глобальний ринок онлайн-опитувань зріс на понад 15 % у 2023–2024 рр., а в Україні понад 60 % закладів вищої освіти впровадили онлайн-опитування для контролю якості навчання

Під час кваліфакаційної роботи було здійснено системний пошук та аналіз інформаційних матеріалів, що стосуються: методів побудови тестових завдань, принципів адаптивного тестування, архітектурних підходів до створення масштабованих вебзастосунків та технологічного стеку

1 АНАЛІЗ СУЧАСНИХ ПРОГРАМНИХ ЗАСОБІВ У СФЕРІ ОРГАНІЗАЦІЇ ТЕСТОВИХ ОПИТУВАНЬ

1.1 Характеристика предметної області вебзастосунків для тестових опитувань

Вебзастосунок для проведення тестових опитувань – це комплексна інформаційна система, що об’єднує модулі створення тестових завдань, керування респондентами, виконання опитувань та аналізу результатів. Можна поділити на блоки:

– конструктор запитань: підтримує різні формати (single-choice, multiple-choice, open-ended, Likert-scale, матричні, ranking), умовні переходи (branching logic) і мультимедіа-елементи;

– менеджер респондентів: реєстрація та автентифікація (email-верифікація, OAuth2, JWT), сегментація за групами та ролями;

– модуль виконання тестів: адаптивний інтерфейс з можливістю автозбереження відповідей, таймером та динамічним відображенням запитань.

1.2 Функціональні підсистеми та процеси обробки результатів опитувань

У сучасних вебзастосунках для тестових опитувань функціональність реалізується через чітко структуровані підсистеми, кожна з яких відповідає за певний етап життєвого циклу опитування — від його створення до формування та аналізу звітів. Такий підхід забезпечує не лише модульність і масштабованість платформи, а й зручність обслуговування, можливість подальшого розширення й інтеграції з іншими інформаційними системами.

Однією з базових функціональних підсистем є модуль адміністрування. Його основним завданням виступає управління користувачами, призначення ролей, контроль доступу до різних частин системи та аудит активності. В сучасних рішеннях адміністратор має інструменти для створення облікових записів, групування користувачів за підрозділами чи класами, а також може налаштовувати

багаторівневу систему доступу. Особливої ваги набуває інтеграція з зовнішніми сервісами автентифікації (наприклад, через протоколи SSO, LDAP чи OAuth2), що дозволяє уніфікувати вхід до корпоративних екосистем.

Наступною критичною підсистемою є конструктор тестів та опитувань. Його функціонал охоплює створення і редагування тестових завдань різного типу, налаштування параметрів проходження (таймінг, випадковий порядок питань, кількість спроб), додавання мультимедійних матеріалів. Зазвичай сучасний конструктор підтримує імпорт і експорт тестів, що спрощує обмін контентом між викладачами чи між різними платформами.

Система керування й аналізу тестових опитувань обробляє вхідні дані від респондентів і надає висновки для замовників досліджень. ЇЇ можна розділити на етапи:

- реєстрація та автентифікація користувачів;
- створення та налаштування опитування;
- збір відповідей;
- попередня обробка даних;
- аналітика та візуалізація;
- зберігання результатів.

Користувач починає взаємодію з системою з реєстрації та автентифікації. Реєстрація може відбуватися через стандартний механізм введення електронної пошти та пароля. Для гарантії конфіденційності та захищеності персональних даних електронна пошта та паролі хешуються за допомогою алгоритмів bcrypt або Argon2. Після успішної реєстрації користувач отримує доступ до основного інтерфейсу, де йому автоматично призначається роль (адміністратор, редактор або респондент). Роль визначає перелік доступних функцій та обмежень у системі. Адміністратор або редактор, успішно авторизувавшись, переходить до інтерфейсу побудови опитування, де за допомогою інструменту SurveyBuilder формує структуру тесту: додає запитання різних форматів (single-choice, multiple-choice,

open-ended, шкальні, матричні), встановлює текст запитання, описи варіантів відповідей та налаштовує умовні переходи.

1.3 Порівняльний аналіз та огляд провідних систем та платформ для тестування

1.3.1 Огляд ключових міжнародних рішень

Станом на 2025 рік існує низка високотехнологічних вебзастосунків для проведення тестових опитувань, що задовольняють потреби освітніх установ, HR-відділів та маркетингових аналітиків. Розглянемо кілька найпоширеніших рішень.

1. «SurveyMonkey» – комерційна SaaS-платформа з інтуїтивним інтерфейсом та широким набором шаблонів тестів. Застосунок підтримує умовні переходи (branching logic), гнучку кастомізацію зовнішнього вигляду та інтеграцію з CRM- і BI-системами (Salesforce, Tableau). Основною перевагою є швидкий початок роботи й готові аналітичні звіти, однак розширені функції вимагають оплати підписки [1]. На рис 1.1 та 1.2 показані логотип та інтерфейс застосунку.

Рисунок 1.1 – Логотип застосунку SurveyMonkey

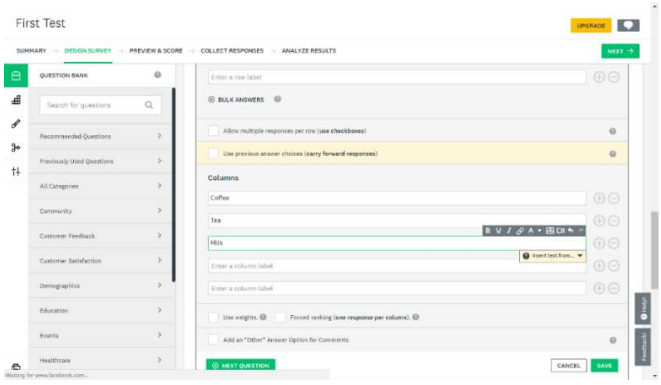

Рисунок 1.2 –Інтерфейс застосунку

2. «Qualtrics» – корпоративне рішення із розвиненими модулями прогнозної аналітики, A/B-тестування та AI-driven insights [2]. Система забезпечує детальну сегментацію респондентів, комплексну автентифікацію і підтримку ISO2700. Недоліком є висока вартість ліцензій та складність адміністрування, що робить її непридатною для невеликих організацій. На рис. 1.3 та 1.4 показані логотип та інтерфейс.

Рисунок 1.3 – Логотип застосунку Qualtrics

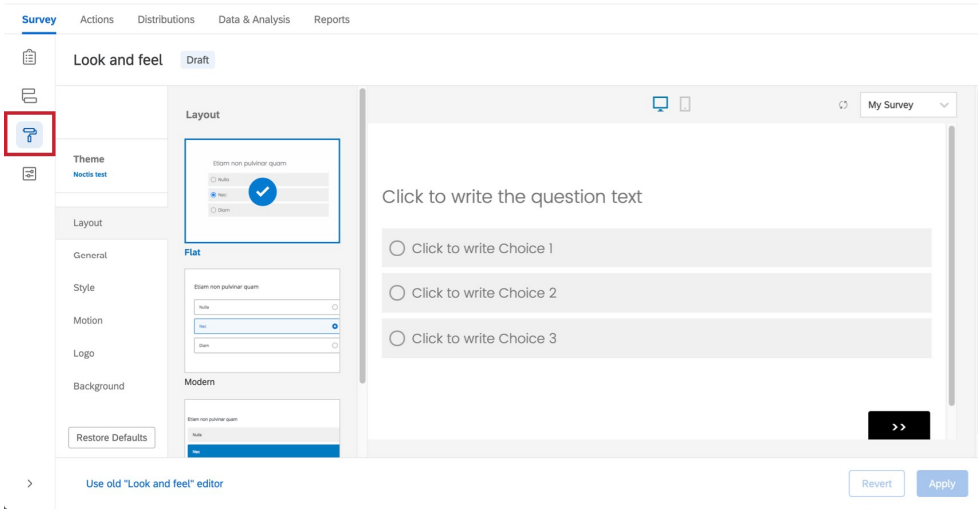

Рисунок 1.4 – Інтерфейс застосунку

3. «LimeSurvey» – open-source платформа з можливістю повної кастомізації та встановлення на власні сервери. Система підтримує понад 80 типів запитань, багатомовність та плагіни для розширення функціоналу. Основна перевага – відсутність vendor lock-in і можливість настройки алгоритмів обробки даних. Недолік – необхідність технічної експертизи для встановлення та супроводу. На рис. 1.3 та 1.4 показані логотип та інтерфейс.

Рисунок 1.5 – Логотип застосунку LimeSurvey

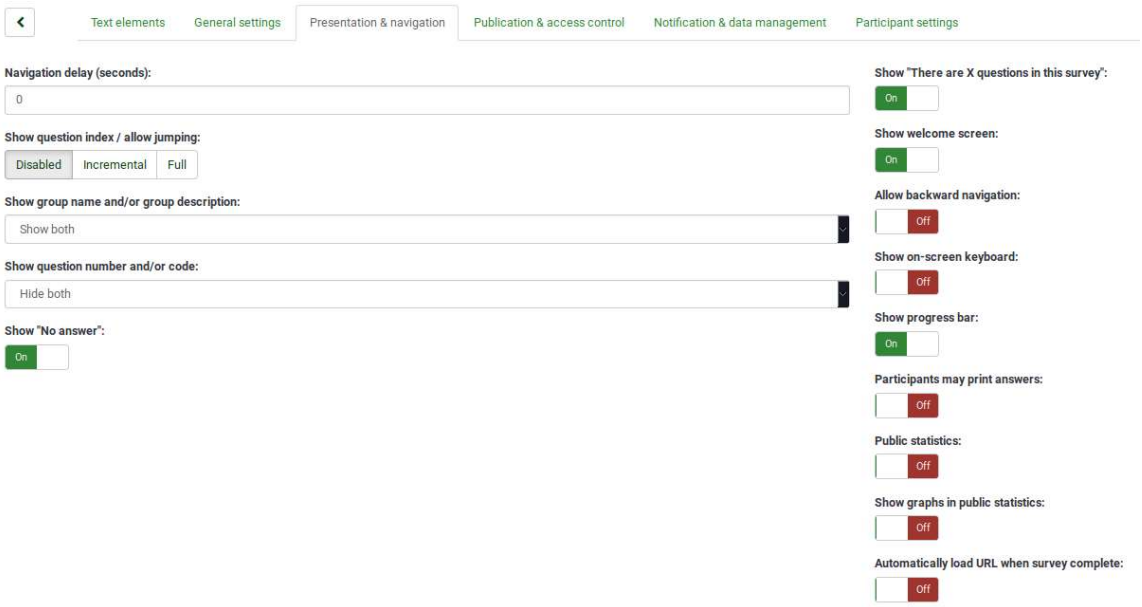

Рисунок 1.6 – Інтерфейс застосунку

4. «Google Forms» – безкоштовний інструмент, інтегрований із Google Workspace [3]. Підтримує лише базові типи запитань і обмежену умовну логіку, але дозволяє швидко створити та надати доступ до форми через посилання. Аналітика реалізована через Google Sheets, що потребує додаткової обробки даних вручну.

Google Forms

Рисунок 1.7 – Логотип застосунку

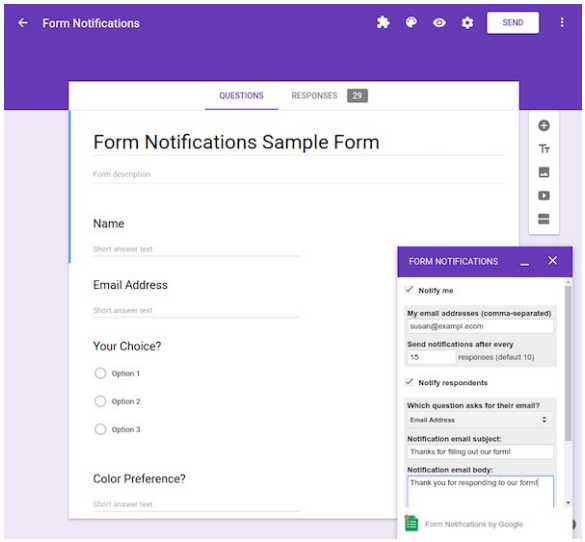

Рисунок 1.8 – Інтерфейс застосунку

1.3.2 Порівняння платформ за критеріями

Для отримання всебічного уявлення про те, яким чином різні сервіси реалізують функціонал тестових опитувань, доцільно провести детальний аналіз їх архітектури, набору доступних інструментів та умов використання. Основна мета порівняння – виявити переваги й обмеження кожної системи з огляду на реальні бізнес- і освітні кейси, а також оцінити, наскільки простої інтеграції та розширення можна очікувати від платформи в майбутньому.

Для детального порівняння цих платформ буде актуально подати їх у вигляді таблиці 1.1.

Нижче наведено критерії, за якими буде порівняно застосунки:

– платформа;
– тип;
– ключові функції;
– переваги;
– недоліки;

1.3.3 Таблиця порівняння платформ для тестових опитувань

Перед початком розробки власного вебзастосунку для тестових опитувань було проведено аналіз і порівняння сучасних популярних платформ, що надають послуги у сфері онлайн-опитувань та анкетування. Мета порівняння — визначити сильні та слабкі сторони існуючих рішень, окреслити функціональні можливості та виявити потенційні напрями для вдосконалення власного програмного продукту. Результати цього аналізу подано у таблиці нижче.

Таблиця 1.1 – Порівняльна характеристика платформ для тестових опитувань

Платформа	**Тип**	**Ключові функції**	**Переваги**	**Недоліки**
SurveyMonkey	SaaS	Шаблони, умовні переходи, CRM-інтеграції	Інтуїтивний UI, готовий до корпоративного використання	Висока вартість
Qualtrics	Enterprise	AI-аналітика, A/B-тестування, прогнозні модулі	Потужна аналітика, розширюваність	Складність налаштування, дорожнеча
LimeSurvey	Open-source	Плагіни, кастомні теми, багатомовність	Відкритий код, гнучкість	Потребує технічних навичок для підтримки
Google Forms	Free	Простий конструктор, інтеграція з Sheets	Простота реалізації	Обмежена аналітика, відсутність branching

Проведене порівняння засвідчує, що жодна з наведених платформ не пропонує повний набір функцій, необхідний для універсального вирішення завдань проведення тестових опитувань. SurveyMonkey вирізняється швидкістю запуску та корпоративним UX, але висока вартість ліцензії може бути неприйнятною для невеликих організацій та освітніх установ. Qualtrics забезпечує глибоку аналітику,

підтримку AI-модулів та можливості A/B-тестування, але потребує значних зусиль з налаштування та великих фінансових витрат. LimeSurvey, як інструмент з відкритим кодом, надає максимальну гнучкість і масштабованість, проте вимагає наявності технічних фахівців для встановлення, налаштування та супроводу. Google Forms підійде для найпростіших завдань та швидкого збирання даних, але функціонал обмежений відсутністю умовних переходів та розвиненої статистики.

1.4 Архітектурні підходи та моделі розгортання

1.4.1 Мікросервісна архітектура

У цій моделі система складається з низки відносно незалежних сервісів, кожен із яких відповідає за власну частину функціоналу — наприклад, аутентифікація, управління тестами, аналітика, розсилки, API тощо. Мікросервіси можуть розроблятися різними командами, оновлюватися й масштабуватися незалежно один від одного, що значно підвищує гнучкість, дозволяє швидше впроваджувати інновації й легше локалізувати й виправляти помилки. Важливим плюсом є й підвищена стійкість до збоїв: відмова одного сервісу не «завалює» всю систему, а лише окрему функцію. Разом із тим, впровадження мікросервісної архітектури підвищує вимоги до інфраструктури, управління міжсервісною взаємодією, забезпечення безпеки каналів передачі даних, централізованого моніторингу й логування.

1.4.2 Монолітна архітектура

На початкових етапах розвитку інформаційних систем у сфері тестування найпоширенішою була монолітна архітектура. У такій моделі весь функціонал платформи зосереджувався в одному великому програмному блоці, що забезпечувало простоту розгортання, цілісність коду та легкість у старті проєкту. Проте вже на середніх і великих обсягах користувачів чи зростанні складності бізнес-логіки монолітні рішення швидко втрачають гнучкість: кожна зміна в коді

чи оновлення вимагає повторної компіляції всієї системи, тестування й розгортання, зростає ризик помилок і часу простою. Додавання нових модулів, інтеграція з іншими системами або спроба розподілити навантаження між кількома серверами часто вимагають суттєвої переробки структури.

1.4.3 Таблиця порівняння архітектур для тестових опитувань

Для забезпечення надійної та масштабованої роботи вебзастосунку для тестових опитувань важливим є вибір архітектурної моделі, яка найкраще відповідатиме функціональним вимогам, очікуваному навантаженню, а також можливостям розвитку системи у майбутньому. В сучасній практиці найбільш поширеними підходами до побудови програмних систем є монолітна та мікросервісна архітектури. Кожна з них має свої переваги та недоліки. Порівняльна характеристика основних властивостей зазначених архітектур наведена у табл. 1.2.

Таблиця 1.2 – Порівняльна характеристика наведених архітектур

Характеристика	**Моноліт**	**Мікросервіси**
Простота розробки	+	-
Масштабованість	-	+
Гнучкість технологій	-	+
Вартість інфраструктури	+	-
Стійкість до відмов	-	+

Вибір архітектурної моделі залежить від багатьох факторів:

- обсяг і складність функціоналу;
- очікувана кількість користувачів та сценарії навантаження;
- необхідність інтеграції з іншими системами;
- вимоги до безпеки, доступності, часу відновлення при збоях;
- можливості подальшого масштабування;
- кваліфікація команди розробників;

– обмеження щодо бюджету, технологічного стеку, часу впровадження.

Наприклад, для невеликої школи або локального навчального центру з кількома сотнями користувачів, які не мають власної IT-команди, цілком виправданим може бути вибір сучасного моноліту (наприклад, на основі Django або Laravel), який швидко розгортається, не потребує складної підтримки і чудово «закриває» базові потреби. Для університету, корпорації, мережі шкіл, стартапу з амбіціями до масштабування чи інтеграції із зовнішніми сервісами — оптимальною є мікросервісна або cloud-native-архітектура з окремими модулями для управління тестами, користувачами, результатами, аналітикою, безпекою.

Отже, монолітна архітектура є оптимальною для початкового етапу розробки вебзастосунку для проведення тестових опитувань завдяки простоті налаштування, швидкому розгортанню MVP та низьким адміністративним витратам, а мікросервісна архітектура забезпечує високу гнучкість та надійність шляхом розподілу системи на незалежні сервіси, що дозволяє автономно розгортати та масштабувати кожен компонент.

1.5 Вибір технологій і інструментів для реалізації вебзастосунку

Для розробки вебзастосунку з проведення тестових опитувань необхідно обрати технології.

1.5.1 Планування технологій для реалізації застосунку

Оскільки вебзастосунок реалізується як сайт із клієнтською логікою, що працює в браузері, основну роль відіграє сучасний JavaScript-стек. Для побудови користувацького інтерфейсу обрано React із TypeScript, що дозволяє створити SPA (single-page application) із швидким рендерингом компонентів та плавною навігацією без повного перезавантаження сторінок. React Router забезпечує маршрутизацію між внутрішніми розділами сайту — створення опитувань, проходження тесту та перегляд результатів.

Для серверної частини як веб-сайт з API використовується Python із

фреймворком FastAPI [4]. FastAPI надає асинхронну обробку HTTP-запитів, автоматично генерує документацію OpenAPI і легко інтегрується з WebSocket для організації реального часу (зміни статусу тесту, тестовий чат).

Візуалізація результатів на стороні клієнта здійснюється через бібліотеку Chart.js, яка легко інтегрується з React-компонентами і дозволяє динамічно оновлювати діаграми без перезавантаження сторінок.

Розгортання вебсайту виконується за допомогою Docker Compose: фронтенд-сервер (nginx+React), бекенд-сервер (FastAPI) та база даних (MongoDB) [5] працюють у окремих контейнерах.

Вибір інтегрованого середовища розробки (IDE) є особистим вибором кожного користувача, в залежності від поставлених задач розробки. Найкращим вибором для розробки подібної системи є Visual Studio Code з можливістю ставити розширення для мов програмування та бібліотек, а також PyCharm спеціально для розробки на Python. Для розробки вебзастосунків мовою JavaScript краще підійде або Visual Studio Code, або специфіковані IDE (наприклад Webstorm для HTML/CSS/JS).

1.5.2 Вимоги до програмної реалізації

Першочергова вимога полягає в застосуванні монолітної архітектури, яка гарантує простоту побудови та розгортання вебсайту зі збалансованим набором функцій. Такий підхід відповідає цілям проєкту, адже дозволяє працювати з єдиною кодовою базою для обробки запитів, збереження даних і відображення інтерфейсу.

Для реалізації клієнтської частини обрано React на JavaScript із TypeScript, що забезпечує комфортну розробку динамічного SPA-інтерфейсу. Серверну логіку втілено на Python із використанням FastAPI (або Aiohttp), що дозволяє швидко обробляти HTTP-запити та генерувати документацію у форматі OpenAPI. Поєднання JS та Python надає гнучкість: фронтенд відповідає за взаємодію з користувачем, а бекенд — за бізнес-логіку та роботу з даними.

Вся інформація зберігається в єдиній базі даних MongoDB, обраній завдяки можливості зберігати документи у форматі JSON, що підходить для різноманітних шаблонів запитань і відповідей. Це спрощує структуру даних та унеможливлює надмірне дублювання, забезпечуючи при цьому високу швидкість запису та читання при великому потоці респондентів.

Висновки до розділу 1

Проведений аналіз предметної області, існуючих рішень і архітектурних підходів засвідчив, що для розробки вебзастосунку з тестовими опитуваннями оптимальним є поетапний підхід: спочатку створення монолітного прототипу з базовим функціоналом, а згодом — еволюція до більш складної архітектури (мікросервіси або serverless) за потреби масштабування чи впровадження нових особливостей. Вивчення міжнародних платформ (SurveyMonkey, Qualtrics, Typeform) й open-source рішень (LimeSurvey) дозволило визначити набір ключових функцій — умовні переходи, мультимедіа-елементи, інтеграції з BI, AI-аналіз. Послідовність взаємодії користувача (реєстрація, побудова опитування, проходження тесту, збереження відповідей, аналіз результатів) формує загальний бізнес-процес, для якого обрано стек React + FastAPI + MongoDB та модуль Chart.js для візуалізації. Вимоги до безпеки, продуктивності й масштабованості лягли в основу технічного завдання та визначили вибір технологій та інструментів.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ ПРОВЕДЕННЯ ТЕСТОВИХ ОПИТУВАНЬ

2.1 JavaScript (JS)

JavaScript – мультипарадигмова мова програмування високого рівня, створена у 1995 році Бренданом Айхом у складі проекту Netscape Navigator. Спочатку задумана як проста скриптова мова для роботи із динамічним контентом у браузері, вона швидко здобула популярність завдяки здатності маніпулювати DOM та обробляти події за натисканнями користувача. У 1997 році стандарт ECMA-262 закріпив ядро мови під ім’ям ECMAScript, від чого і пішла абревіатура ES5, ES6 тощо. З часом JavaScript еволюціонував: з’явилися проміси, модулі, класи, асинхронні функції (async/await), що відкрили можливості для побудови великих додатків [6].

До 2009 року JS застосовувався лише у браузері, але з появою Node.js мова стала повноцінним середовищем для серверної розробки. Node.js реалізує V8-движок Google Chrome у контексті сервера, додаючи модульне API та асинхронні функції вводу/виводу. Цей крок перетворив JS на універсальний інструмент з відкритим доступом до файлової системи, мережі та баз даних.

Рисунок 2.1 – Логотип JavaScript

У проєкті JS використовується для реалізації інтерфейсу односторінкового додатка (SPA) [7]. React-компоненти, написані на чистому JS або, частіше, на

TypeScript, відповідають за відображення запитань, збір відповідей та динамічний рендеринг статистичних даних. Основні можливості JavaScript у цьому контексті:

- декларативне оновлення UI через React-методи (render, setState);
- підтримка модульного імпорту/експорту через стандартизовані ES6-модулі, що покращує організацію коду;
- використання функцій вищого порядку (HOCs) та кастомних хуків (React Hooks) для повторного використання логіки, наприклад, для авторизації та виклику API;
- робота з асинхронними запитами Fetch API або Axios: надсилання POST-запитів із відповідями респондентів, GET-запитів для отримання структур опитувань.

Щодо стилістики коду, існує широка спільнота та набір best practices:

- використання ESLint з конфігурацією Airbnb для консистентного стилю та запобігання помилок;
- преформатування коду через Prettier для автоматичного вирівнювання відступів;
- типізація через JSDoc-коментарі в чистому JS або, що переважно, повна міграція на TS [8].

JavaScript також використовується у серверній частині через Node.js (див. розділ 2.2) та у процесах побудови (webpack, Rollup, Vite), де скрипти трансформуються і оптимізуються для роботи у браузері та мережі.

2.1.2 TypeScript

TypeScript — це мова програмування, розроблена Microsoft та представлена в 2012 році з метою подолати обмеження динамічної типізації JavaScript у великих кодових базах. TS є надмножиною JavaScript, що підтримує весь існуючий синтаксис JS, проте додає ключові можливості статичної типізації, інтерфейси, перерахування (enums) та сучасні конструктори класів, що відповідають стандартам ECMAScript [9]. Основною ідеєю TypeScript є виявлення типових

помилок ще на етапі розробки, до компіляції у JavaScript, що значно підвищує надійність та підтримуваність коду.

Типова структура файлу на TS починається з оголошення інтерфейсів і типів, що задають контракт взаємодії між компонентами. Приклад коду структури об’єкта питання (Question):

```
interface Option {
    id: string;
    text: string;
}

interface Question {
    id: string;
    text: string;
    type: 'single-choice' | 'multiple-choice' | 'open-ended';
    options?: Option[];
}
```

У командній розробці TypeScript полегшує забезпечення єдиної домовленості щодо форматів даних: інтерфейси та типи зберігаються централізовано, і будь-яка зміна в контракті автоматично сигналізує про необхідність оновлення всіх компонентів, що з ним працюють. Це скорочує час на пошук багів та покращує підтримку коду.

2.1.3 React

React — JavaScript-бібліотека для побудови користувацьких інтерфейсів, створена компанією Facebook у 2013 році. Вона швидко стала стандартом у розробці односторінкових додатків завдяки компонентному підходу та використанню віртуального DOM [10]. React дозволяє аналізувати UI як дерево компонентів, кожен з яких інкапсулює розмітку (JSX), стилі та власну логіку.

React вперше представили у 2011 році як внутрішній інструмент Facebook для вирішення проблем продуктивності при рендерингу великих інтерфейсів з частими оновленнями стану. Публічний реліз у 2013 році ознаменував початок революції в архітектурі вебінтерфейсів. З введенням React Hooks у версії 16.8 бібліотека

перейшла до декларативного аналізу станів і побічних ефектів, дозволяючи уникнути складних класових компонентів.

У React виділяють два основні види компонентів:

– функціональні (Function Components) – легкі, оголошені як функції, повертають JSX. З Hooks вони отримали повний набір можливостей (стан, контекст, зворотні виклики);

– класові (Class Components) базуються на ES6-класах, мають методи життєвого циклу (componentDidMount, shouldComponentUpdate тощо) та локальний стан.

2.1.4 Основні концепції

Компоненти: кожен компонент інкапсулює розмітку (JSX), стилі (CSS/SCSS або CSS-in-JS) та власну логіку (JavaScript або TypeScript). Наприклад, у конструкторі опитувань створюється компонент QuestionEditor, який відповідає за відображення та редагування окремого питання [11]. Наприклад:

– Props (властивості) передаються від батьків до дочірніх компонентів і залишаються незмінними всередині компонента. Наприклад, компонент ResultsDashboard отримує через props масив відповідей та метадані опитування;

– State (внутрішній стан) зберігається всередині компонента і може змінюватися за допомогою Hook-функцій (useState). У SurveyRunner стан використовується для відстеження прогресу: поточної індексу питання та збережених відповідей;

– Virtual DOM: при зміні стану React створює нове дерево елементів у пам’яті й порівнює його з попереднім (дифінг), після чого оновлює тільки ті частини реального DOM, які змінилися [12]. Це мінімізує дорогоцінні операції з DOM та пришвидшує відгук;

– Hooks: набір функцій для роботи зі станом та побічними ефектами у функціональних компонентах:

– useState (initialValue) повертає пару: значення стану і функцію для його оновлення;

– useEffect (callback, deps) виконує побічний ефект (наприклад, автозбереження відповіді через REST API) щоразу, коли змінюються залежності у масиві deps;

– useContext(Context) дає змогу отримувати глобальні дані (наприклад, інформацію про авторизованого користувача) без необхідності передавати props через усі рівні компонентів.

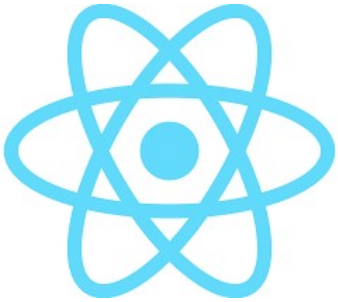

Рисунок 2.2 – Логотип React.js

React забезпечує високу швидкодію при великих UI-деревах та змінних даних, простоту розподілу коду та повторного використання компонентів.

2.1.5 Порівняльна таблиця фронтенд рішень.

Перед вибором конкретних технологій для розробки вебзастосунку тестових опитувань було проведено детальний аналіз сучасних інструментів фронтенду та бекенду, їх переваг, недоліків і доцільності використання у практичних проєктах.

Особливу увагу приділено мовам програмування, бібліотекам і фреймворкам, які забезпечують надійну архітектуру, продуктивність, масштабованість і зручність підтримки коду.

Нижче у таблиці наведено порівняння ключових інструментів, які розглядалися для реалізації програмного продукту.

Таблиця 2.1 – Порівняльна характеристика технологій

Технологія	**Пояснення**	**Переваги**	**Недоліки**
JavaScript	Базова мова для браузера та Node.js	Широке розповсюдження, асинхронність	Динамічна типізація
TypeScript	Надмножина JS із статичною типізацією	Рання виявлення помилок, автодоповнення	Потребує компіляції
React	Бібліотека UI із Virtual DOM, JSX, Hooks	Компонентний підхід, продуктивність	Крута крива входження Hooks

2.2 Python

Python – високорівнева мова програмування, відома своєю читабельністю та широкою екосистемою бібліотек. У цьому проєкті Python відповідає за реалізацію бізнес-логіки, обробку запитів клієнта та взаємодію з базою даних.

Історично Python з’явився у 1991 році завдяки Гвідо ван Россуму, який прагнув створити просту та інтуїтивну мову [13]. Основні принципи — читабельність, мінімалістичність синтаксису та велика стандартна бібліотека — зробили Python популярним у веб-розробці, науці про дані та автоматизації.

У бекенді проєкту використовується асинхронний веб-фреймворк FastAPI. Його вибір обґрунтовано такими перевагами:

– асинхронність: завдяки async/await обробка запитів відбувається неблокуюче, що критично при високому навантаженні;

– автоматична документація: на основі Pydantic-моделей генерується Swagger UI та Redoc, що спрощує тестування й інтеграцію фронтенду;

– валідація даних: Pydantic перевіряє вхідні JSON-запити та перетворює їх у Python-класи з чіткими типами;

– Dependency Injection: за допомогою механізму Depends легко підключати спільні сервіси (база даних, кеш) до ендпоінтів. На рис. 2.3 показано логотип Python.

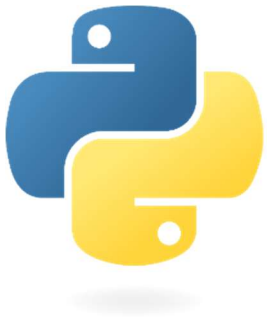

Рисунок 2.3 – Логотип Python

Приклад коду для обробки запиту:

```
from fastapi import APIRouter, Depends
from pydantic import BaseModel

class CreateSurvey(BaseModel):
      title: str
      questions: list[QuestionSchema]
@router.post("/surveys")
async def create_survey(payload: CreateSurvey, db=Depends(get_db)):
      survey = await SurveyService.create(db, payload)
      return survey
```

FastAPI запускається на Uvicorn — легкому ASGI-сервері, що забезпечує швидку обробку HTTP/2 запитів і WebSocket-з’єднань. У продакшні до Uvicorn зазвичай додають Gunicorn для керування кількома воркерами.

На рис. 2.4 показана узагальнена інформація для типів даних.

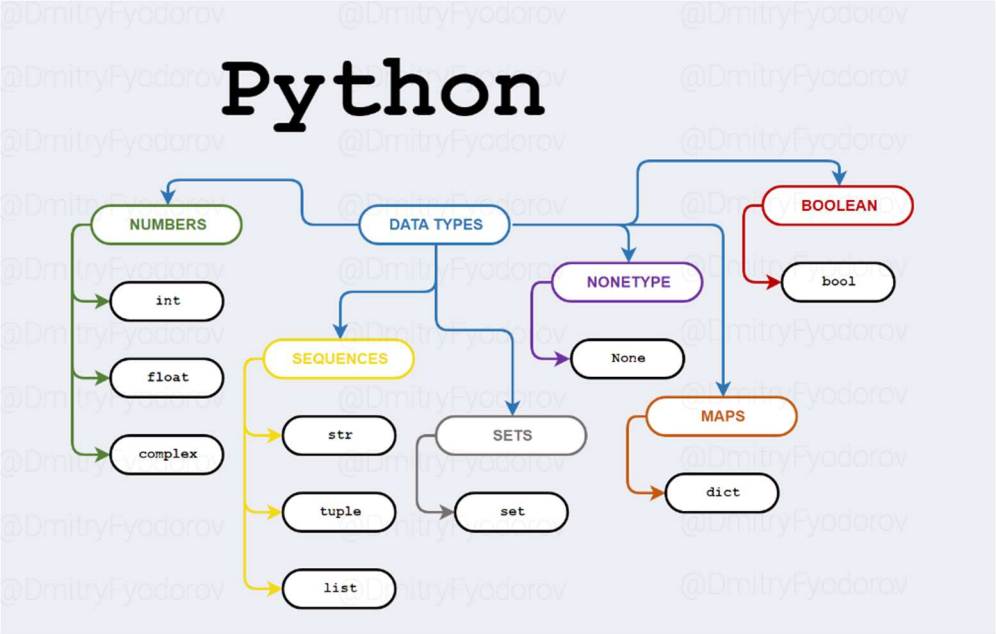

Рисунок 2.4 – Інформація про типи даних в Python

У вебзастосунку логіка збереження тесту реалізована в SurveyService, який приймає об’єкт CreateSurvey, зберігає його в базу даних, генерує унікальне посилання для респондентів та повертає ідентифікатор для подальших запитів.

2.3 FastAPI

FastAPI — сучасний Python-фреймворк, створений Себастьяном Раміресом у 2018 році. Він побудований на Starlette та Pydantic і орієнтований на високу продуктивність і простоту використання [14].

Основні характеристики FastAPI:

– ASGI-підтримка: обробляє запити асинхронно через стандартизований інтерфейс ASGI, що забезпечує швидку роботу навіть під великим навантаженням;

– автоматична документація: на основі визначених Pydantic-моделей генерує Swagger UI (/docs) та ReDoc (/redoc) без додаткових зусиль;

– валідація та серіалізація: Pydantic-моделі автоматично перевіряють вхідні дані та перетворюють їх у чисті Python-класи з анотаціями типів;

– Dependency Injection: механізм Depends дозволяє легко підключати спільні сервіси — базу даних, кеш, авторизацію — до ендпоінтів.

Приклад коду для створення опитування:

```
from fastapi import APIRouter, HTTPException, Depends
from pydantic import BaseModel
from typing import List

class Option(BaseModel):
    text: str

class Question(BaseModel):
    text: str
    options: List[Option]

class CreateSurvey(BaseModel):
    title: str
    questions: List[Question]

router = APIRouter()

@router.post("/surveys", response_model=SurveyResponse)
async def create_survey(payload: CreateSurvey, db: Session =
Depends(get_db)):
    # логіка збереження опитування
    survey = await SurveyService.create(db, payload)
    if not survey:
        raise HTTPException(status_code=500, detail="Error creating
survey")
    return survey
```

У цьому прикладі CreateSurvey описує очікувану структуру запиту, а FastAPI автоматично перевірить JSON на відповідність моделі. Сесія бази даних db інжектиться через Depends, що спрощує підключення та тестування.

2.4 MongoDB

MongoDB — це популярна NoSQL СУБД, яка зберігає інформацію у вигляді JSON-подібних документів (BSON), що дозволяє працювати з даними у більш гнучкому форматі, ніж у реляційних базах. MongoDB з’явилася у 2009 році завдяки компанії 10gen (нині MongoDB Inc.) [15] з метою спростити зберігання великих обсягів напівструктурованих даних і підвищити масштабованість.

Поява MongoDB була відповіддю на обмеження реляційних СУБД у роботі з великими обсягами напівструктурованих даних та непередбачуваними змінами схеми. Відсутність необхідності виконувати складні міграції та підтримка JSON-

подібних документів зробили MongoDB привабливим вибором для швидкої розробки прототипів і гнучких архітекту. Особливості MongoDB описані нижче.

1. Гнучка схема даних. Документи можуть містити різні поля, а нові властивості легко додаються без потреби у міграціях схеми. Це особливо корисно при зберіганні відповідей на тести з різними типами запитань (від простих текстових до складних матричних), адже структура документа може змінюватися від одного опитування до іншого.

2. Документна модель. Дані зберігаються у колекціях, кожен документ відповідає одному запису. Приклад коду як може виглядати документ у колекції responses:

```
{
  "_id": ObjectId("..."),
  "surveyId": ObjectId("..."),
  "userId": "user123",
  "submittedAt": ISODate("2025-05-01T12:34:56Z"),
  "answers": [
    { "questionId": "q1", "answerValue": "A" },
    { "questionId": "q2", "answerValue": ["B", "C"] }
  ]
}
```

Завдяки цьому зберігання відповідей виконується в рамках одного документа, що прискорює читання набору відповідей конкретного користувача.

MongoDB підтримує шардинг — розділення колекцій між різними вузлами кластера за ключем шардингу (наприклад, surveyId). Це дозволяє рівномірно розподіляти навантаження при великій кількості респондентів. Реплікаційні набори (Replica Sets) у разі виходу з ладу одного вузла інші автоматично обирають нового первинного та продовжують операції читання/запису.

MongoDB Aggregation Framework дозволяє виконувати складні обчислення та трансформації даних без необхідності виносити логіку на рівень клієнта.

Для оптимізації запитів індекси можна створювати за полями surveyId, userId, а також вкладеними полями, наприклад answers.questionId або answers.answerValue. Це забезпечує швидкий пошук відповідей за запитанням або відповіддю.

2.4.1 Порівняльна таблиця різних СУБД

Таблиця 2.2 – Порівняльна характеристика СУБД

СУБД	Тип	Застосування	Переваги	Недоліки
PostgreSQL	Реляційна	Структуровані сутності: користувачі, опитування, питання	Повна підтримка ACID-транзакцій і складних JOIN; JSONB-поля для напівструктурованих даних	Горизонтальне масштабування складне; зміни схеми потребують міграцій
MySQL	Реляційна	Веб-додатки з середнім навантаженням	Широке поширення; висока швидкість при простих запитах	Обмежена підтримка складних аналітичних запитів; відсутність справжніх транзакцій на деяких рушіях
MongoDB	Документна	Напівструктуровані дані: відповіді, логи, метадані	Гнучка схема; Aggregation Framework; шардінг і реплікація	ACID-транзакції лише з версії 4.0; вимагає продуманого індексування
Redis	Ключ/значення	Кешування, сесійні дані, черги повідомлень	Надзвичайно швидка робота в пам’яті; структури даних	Підходить не для великих обсягів довготривалих даних

З наведеного порівняння видно, що реляційні СУБД PostgreSQL та MySQL більше підходять для надійної обробки структурованих даних із суворими вимогами до транзакційності, тоді як MongoDB та Redis демонструють вищу продуктивність і гнучкість при роботі з великими масивами напівструктурованих або оперативних даних.

2.4.2 Основні можливості MongoDB

- гнучка схема (schema-less): документи можуть містити різні поля, а нові властивості легко додаються без потреби у складних міграціях, що підходить для динамічних структур тестових даних;
- документно-орієнтована модель: кожен документ у колекції зберігає повний набір даних про одну сутність (наприклад, результат проходження тесту одним користувачем), що зменшує необхідність в об’єднаннях (JOIN);
- агрегаційний фреймворк – потужний інструмент для складних обчислень, фільтрацій та трансформацій даних на стороні сервера (pipeline-операції $match, $group, $project, $sort);
- індексація: підтримка індексів за будь-якими полями, включно із вкладеними та багатопольовими індексами, що забезпечує швидкий пошук по великих обсягах даних;
- реплікація (Replica Set): автоматичне дублювання даних на кілька вузлів для відмовостійкості і високої доступності; у разі збоїв один із вузлів автоматично стає первинним;
- шардинг (Sharding): горизонтальне масштабування через розподіл колекцій за ключем шардингу, що дозволяє зберігати і обробляти дуже великі набори даних;
- геопросторовий та текстовий пошук: вбудовані можливості для виконання запитів за географічними координатами та повнотекстового пошуку;
- TTL-індекси (Time To Live): автоматичне видалення документів через певний проміжок часу, корисне для зберігання сесій чи тимчасових логів;

– підтримка транзакцій: починаючи з версії 4.0, MongoDB підтримує мультидокументні транзакції для забезпечення цілісності даних у складних операціях.

2.4.3 Масштабування, аналітика та безпека

Для великих навантажень MongoDB підтримує горизонтальне масштабування через шардинг: колекцію можна розбити на сегменти за ключем шардингу (наприклад, surveyId), а спеціальні вузли mongos автоматично керують маршрутизацією запитів між шардовими кластерами. Це дозволяє лінійно нарощувати обсяг зберігання та пропускну здатність.

Аналітичні запити виконуються за допомогою Aggregation Framework, що надає можливість створювати багаторівневі pipeline-операції:

– $match — фільтрує документи за умовами (наприклад, за конкретним surveyId) ;

– $unwind — розгортає масиви відповідей у послідовність окремих документів;

– $group — агрегує дані (рахує відповіді, сумує час на проходження тесту);

– $project — формує кінцеву структуру результату для відправки до клієнта або збереження в аналітичній колекції;

– $sort та $limit — сортування та обмеження результатів.

Щодо безпеки, MongoDB забезпечує автентифікацію SCRAM, шифрування TLS для мережевих з’єднань, шифрування даних на диску через Encrypted Storage Engine та рольову модель доступу (RBAC).

У контексті вебзастосунку для тестових опитувань MongoDB обрана для зберігання відповідей респондентів, сесійних логів та історії змін опитувань. Така архітектура дозволяє швидко отримувати повний набір даних одного респондента за єдиним запитом, виконувати комплексні аналітичні вибірки на боці сервера та легко масштабувати систему при зростанні кількості користувачів чи обсягу збережених результатів. На рис. 2.7 показані можливості MongoDB.

Рисунок 2.7 – Можливості MongoDB

Завдяки цим можливостям MongoDB є ефективним рішенням для зберігання напівструктурованих відповідей респондентів, логів і тимчасових сесійних даних у вебзастосунку для проведення тестових опитувань.

У контексті вебзастосунку для тестових опитувань MongoDB зберігає відповіді респондентів та історію змін опитувань як документи, що дозволяє швидко отримувати дані без складних JOIN-операцій. Агрегації підраховують частоту відповідей і формують службові показники (час проходження, відсоток відповідей), які передаються в ResultsDashboard для візуалізації.

Висновки до розділу 2

Другий розділ присвячено вичерпному огляду технологічного стеку, необхідного для створення вебзастосунку з тестових опитувань. Було показано, як JavaScript у поєднанні з React дозволяє реалізувати компонентний інтерфейс із декларативним оновленням через Virtual DOM, а також як React Hooks спрощують керування локальним та глобальним станом. Додаткова інтеграція TypeScript забезпечує статичну типізацію та формалізацію структур даних, що суттєво знижує

ризики помилок під час розробки та підтримує узгодженість контрактів між клієнтом і сервером. На боці сервера Python із FastAPI подарував можливість створити високопродуктивні асинхронні REST-ендпоінти з автоматичною документацією та жорсткою валідацією через Pydantic-моделі. Для зберігання напівструктурованих даних було обрано MongoDB, документна модель якої надає гнучкість у роботі з різноманітними форматами відповідей та потужний фреймворк агрегацій для аналітики. Нарешті, включення інструментів DevOps—від контейнеризації через Docker і оркестрації в Kubernetes до автоматизації CI/CD із GitHub Actions та моніторингу за допомогою Prometheus, Grafana і Sentry—забезпечує надійність розгортання, прозорість експлуатації та оперативне реагування на інциденти. Загалом, аналізовані у розділі компоненти формують стійку, масштабовану та зручну у підтримці платформу, готову до подальшого розширення функціональності та інтеграції нових сервісів.

3 ПРАКТИЧНІ АСПЕКТИ РОЗРОБКИ, ІНТЕГРАЦІЇ ТА ВПРОВАДЖЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ ТЕСТОВИХ ОПИТУВАНЬ

3.1 Проєктування структури та функціональних модулів

Практичне створення вебзастосунку для тестових опитувань починається з проектування архітектури та визначення основних функціональних модулів, які забезпечать повний цикл організації опитування — від формування структури тесту до збору і аналітики результатів. Важливо закласти таку модель взаємодії компонентів, яка дозволить легко масштабувати систему, додавати нові функції без суттєвого переписування коду та забезпечити підтримку актуальних вимог безпеки.

Зазвичай архітектура подібних систем будується за модульним принципом, коли кожен модуль виконує чітко визначені завдання на певному етапі життєвого циклу опитування. Серед базових модулів можна виділити такі: модуль адміністрування користувачів, який відповідає за реєстрацію, автентифікацію та розмежування прав доступу; конструктор тестів для створення, редагування й організації різних типів опитувань; модуль збору відповідей, який забезпечує безперервне фіксування даних у базі та їх валідацію; модуль аналітики та візуалізації, який дає можливість швидко аналізувати отримані результати, формувати статистичні звіти та виявляти тенденції.

Основні етапи проєктування вебзастосунку для тестових опитувань:

– визначення ролей користувачів (адміністратор, редактор, респондент);

– опис основних сценаріїв використання (створення опитування, проходження тесту, аналіз результатів);

– формування структури бази даних для збереження тестів, відповідей та метаданих;

– вибір технологій для кожного шару (UI, API, БД);

– розробка API для інтеграції з іншими системами.

На табл. 3.1 показані модулі та призначення.

Таблиця 3.1 – Основні функціональні модулі системи та їх призначення

Модуль	Призначення
Модуль адміністрування	Керування користувачами, ролями, безпекою
Конструктор опитувань	Створення, редагування і налаштування структури тестів
Модуль проходження тесту	Інтерфейс для респондентів, забезпечення збору відповідей
Аналітика і візуалізація	Обробка результатів, генерація звітів, побудова діаграм
API – ітерації	Взаємодія із зовнішніми системами, експорт/імпорт даних

3.2 Реалізація клієнтської та серверної частини

Процес реалізації клієнтської та серверної частини сучасної системи тестових опитувань потребує глибокого розуміння як бізнес-процесів, так і найкращих технологічних практик. Вдале розділення на front-end і back-end дозволяє не лише гнучко розвивати функціонал і зручно інтегруватися з іншими сервісами, а й досягти високої продуктивності, безпеки, масштабованості та сучасного користувацького досвіду. У цьому підрозділі розглядаються основні принципи, інструменти та типові підходи до побудови кожної частини вебзастосунку.

3.2.1 Клієнтська (front-end) частина: сучасний підхід до UX/UI

Front-end відповідає за всі взаємодії користувача із системою, тобто за зовнішній вигляд, структуру сторінок, швидкість відгуку, візуалізацію результатів та адаптивність під різні пристрої.

Основні вимоги до клієнтської частини:

– інтуїтивний інтерфейс (зрозумілий навіть для не підготовлених користувачів);

– адаптивність (коректна робота на різних пристроях і браузерах);

– підтримка інтерактивних елементів (динамічні форми, таймери, drag-and-drop, графіки);

– миттєвий відгук на дії користувача (завдяки SPA-підходу);

– локалізація (багатомовність) і доступність (accessibility);

Вибір технологій для front-end:

– **React.js** – найпопулярніший фреймворк для розробки SPA (Single Page Application). Його компонентний підхід і багата екосистема (Redux, React Router, Formik, Material UI) [16] дозволяють створювати складні динамічні інтерфейси з високим рівнем реюзабельності коду;

– **TypeScript** – статична типізація допомагає уникати багатьох помилок ще на етапі розробки, особливо у великих командах.

Типовий стек front-end:

– **React + TypeScript** (основа логіки й компонентів);

– **Material UI** (UI-елементи, іконки, адаптивність);

– **Redux/Context API** (глобальний стан, управління сесією).

Приклад архітектури front-end:

– **App Shell** – базовий каркас SPA, що завантажується один раз;

– модулі: головна сторінка, кабінет користувача, конструктор тестів, проходження опитування, аналітика, адміністрування [17];

– підключення до back-end через REST API чи WebSocket.

3.2.2 Серверна (back-end) частина: бізнес-логіка та інтеграції

Back-end забезпечує всю «невидиму» частину роботи системи: зберігання й обробку даних, автентифікацію, обробку бізнес-логіки, виконання інтеграцій із зовнішніми сервісами, автоматизацію аналітики, захист і аудит.

Основні вимоги до серверної частини:

- асинхронна обробка багатьох одночасних запитів;
- гнучка модульна структура (забезпечення масштабування й підтримки);
- безпека (авторизація, шифрування, аудит дій);
- відкрите API для інтеграцій;
- висока швидкість роботи з базами даних;
- легкість підтримки й оновлення (CI/CD).

Вибір технологій для back-end:

- **FastAPI (Python):** сучасний асинхронний фреймворк, простий у використанні, має високу продуктивність і автоматичну генерацію документації OpenAPI (Swagger);
- **Node.js (Express, NestJS)**: зручний для випадків, коли front-end і back-end розробляються на JavaScript;
- **REST/GraphQL API:** для взаємодії з клієнтською частиною й зовнішніми системами.

Типовий стек back-end:

- **FastAPI** (бізнес-логіка, API);
- **PostgreSQL/MongoDB** (зберігання даних, залежно від структури) [18];
- **OAuth2/JWT** (автентифікація й авторизація);
- **Pytest/Jest** (автоматизоване тестування API).

Приклад архітектури back-end:

- **API Gateway** — основна точка входу для запитів з front-end;
- мікросервіси/модулі: управління користувачами, конструктор тестів, модуль проходження, аналітика, адміністрування, інтеграції;
- Захист: HTTPS, CORS, throttling, audit log.

3.2.3 Взаємодія front-end і back-end

Зв’язок між клієнтською та серверною частиною зазвичай організовується через REST API (або GraphQL для більш складних сценаріїв).

Основні етапи взаємодії:

– користувач через інтерфейс клієнтської частини ініціює дію (реєстрація, створення тесту, проходження опитування, перегляд звітів);

– Front-end формує HTTP(S)-запит до API (наприклад, POST /api/answers);

– Back-end обробляє запит, перевіряє права, взаємодіє з БД, повертає результат у форматі JSON;

– Front-end оновлює UI, відображає повідомлення або дані для користувача.

Список основних аспектів реалізації front-end/back-end:

– SPA-архітектура для максимальної швидкості й адаптивності;

– чітке розділення відповідальності між front-end і back-end;

– відкрите API для легких інтеграцій;

– автоматизоване тестування (unit, integration, e2e);

– актуальна документація (Swagger/OpenAPI);

– забезпечення безпеки на всіх рівнях (автентифікація, валідація, захист від XSS/CSRF).

На табл. 3.2 показані декілька компонентів з їх описом, основною функцією та технологією.

Таблиця 3.2 – Основні компоненти клієнтської та серверної частин

Компонент	**Технологія**	**Основна функція**
Front-end	React, TypeScript, Material UI	Інтерфейс користувача, SPA, графіки, адаптивність
Back-end	FastAPI, PostgreSQL/MongoDB, Redis	API, бізнес-логіка, робота з БД, кешування
API	REST, OpenAPI	Взаємодія між front-end та back-end

Кінець таблиці 3.2

Тестування	Jest, Pytest, Testing Library	Перевірка якості коду, стабільності функцій
Інтеграції	OAuth2, WebSocket, Webhook	Інтеграція з зовнішніми системами

Приклад практичної реалізації:

– клієнтська частина: користувач заходить на платформу, бачить інтуїтивне меню, конструктор тестів із drag-and-drop, графіки результатів, адаптивний дизайн, миттєву зміну мови інтерфейсу;

– серверна частина: швидко обробляє створення/збереження тесту, асинхронно генерує звіти, дозволяє підключення зовнішніх аналітичних систем через API.

3.3 Тестування та контроль якості

Тестування та контроль якості (Quality Assurance, QA) є невід'ємними складовими процесу створення та впровадження будь-якої складної інформаційної системи, зокрема систем для автоматизованого тестування й опитувань. Якість цифрового продукту визначається не лише реалізованим функціоналом, а й стабільністю роботи, зручністю для користувача, стійкістю до навантажень, захищеністю від помилок і вразливостей.

Сучасні підходи до QA базуються на комбінації автоматизованого й ручного тестування, впровадженні CI/CD, залученні користувачів до тестування та постійному моніторингу системи в процесі експлуатації.

Мета тестування – виявлення помилок на ранніх етапах, запобігання дефектам у продакшн-версії та створення такого цифрового продукту, який відповідає очікуванням кінцевих користувачів.

Основні принципи:

- превентивність (запобігання, а не виправлення багів);
- комплексність (охоплення всіх рівнів — від окремої функції до всього сценарію);
- безперервність (тестування на кожному етапі розробки — DevOps, CI/CD);
- автоматизація (для ефективності та повторюваності);
- доказовість (логування, звітність, відслідковуваність кожного інциденту);
- залучення користувачів (User Acceptance Testing, Beta testing).

Основні типи тестування у вебзастосунках:

– юніт-тестування (unit testing) перевіряє окремі функції, методи чи класи на відповідність очікуваній поведінці. Дає змогу виявити помилки на ранньому етапі розробки, особливо у критичних частинах бізнес-логіки та обробки даних;

– інтеграційне тестування (integration testing) виявляє проблеми на стиках між модулями (наприклад, взаємодія front-end з back-end, взаємодія back-end із БД). Допомагає переконатися, що всі компоненти коректно працюють разом;

– тестування продуктивності (performance/load testing) перевіряє, як система поводиться під навантаженням: одночасне проходження тесту сотнями чи тисячами користувачів, швидкість обробки великих звітів, реакція на пікові навантаження;

– ручне (мануальне) тестування необхідне для тестування UI/UX, пошуку неочевидних багів, перевірки адаптивності інтерфейсу, багатомовності, складних сценаріїв, що не охоплені автотестами;

– побудова процесу тестування у реальному проєкті;

– розробка тест-кейсів на основі функціональних вимог створюються сценарії, які охоплюють всі можливі дії користувача та системи. Приклад: «Студент може пройти тест лише один раз», «Адміністратор бачить всі звіти, а викладач – лише свої»;

– створення окремих стендів із копіями бази, псевдоданими, захищеним

доступом для тестувальників;

– впровадження автоматизації тестування: всі основні сценарії мають бути покриті автотестами, що запускаються на кожному pull request (CI-процес);

– виконання ручного тестування: охоплення edge-case, UX-помилок, виявлення нетипових багів;

– тестування продуктивності: перед великими сесіями (наприклад, масове ЗНО) – симуляція навантаження, виявлення “вузьких місць”;

– моніторинг у продакшн-середовищі: використання Sentry, Prometheus, Grafana для сповіщення про помилки в реальному часі;

– аналіз і документація інцидентів: ведення журналів (логів), збір статистики для подальшого вдосконалення. На таблиці 3.3 показані основні етапи та інструменти тестування.

Таблиця 3.3 – Основні етапи та інструменти тестування системи

Етап тестування	**Інструменти/Методи**	**Завдання та результат**
Юніт-тестування	Jest, Pytest	Перевірка окремих функцій, швидке виявлення багів
Інтеграційне	Postman, Pytest, Mocha	Перевірка взаємодії модулів, API
End-to-End	Cypress, Selenium, Playwright	Симуляція повної роботи користувача
Продуктивність	JMeter, k6, Artillery	Оцінка швидкодії, пошук “вузьких місць”

Кінець таблиці 3.3

Безпека	OWASP ZAP, SonarQube, Burp Suite	Виявлення вразливостей, перевірка захисту
Ручне	Manual testing, бета-тестування	UI/UX, кросбраузерність, нестандартні кейси
Моніторинг у реальному часі	Sentry, Prometheus, Grafana	Виявлення та оперативне виправлення проблем

Практичні приклади впровадження QA в освітніх та корпоративних системах:

– масові платформи (Coursera, EdX, Prometheus) мають тисячі автоматичних тестів, окремі команди QA, багаторівневу систему контролю;

– для національних опитувань у реальному часі тестування продуктивності проводиться на «клоні» системи з аналогічною кількістю користувачів;

– постійне залучення зворотного зв’язку від викладачів і студентів дозволяє швидко фіксувати «дрібні» проблеми, які автоматичне тестування не завжди виявляє.

Типові помилки та шляхи їх уникнення:

– погане покриття тестами: рішення – поступове збільшення тестового покриття, автоматизація, «тестування через розробку» (TDD);

– ігнорування тестування продуктивності: рішення – впровадження регулярних навантажувальних тестів перед ключовими оновленнями;

– відсутність тестування безпеки: рішення – впровадження регулярних пентестів, інтеграція статичного аналізу коду;

– недостатній зворотний зв’язок із користувачами: рішення – організація системи збору відгуків, оперативна робота служби підтримки;

– відсутність документації на автотести: рішення – автоматична генерація документації, впровадження стандартів оформлення.

Висновки до розділу 3

У третьому розділі детально розкрито всі основні етапи створення сучасної системи автоматизованого тестування: від проектування архітектури та функціональних модулів до впровадження, підтримки, контролю якості та постійного розвитку. Розглянута логіка побудови та експлуатації платформи підкреслює:

– модульний і компонентний підхід дозволяє створювати масштабовані, гнучкі, легко керовані системи;

– інтеграція тестування і QA на всіх етапах гарантує якість, стабільність, довіру користувачів;

– впровадження та підтримка – це не разова дія, а безперервний процес, який вимагає технічної, організаційної та соціальної роботи.

Практичний досвід засвідчив, що найкращі технологічні рішення приносять очікувані результати лише за умови глибокої інтеграції в роботу організації, підтримки з боку команди, регулярного навчання, моніторингу й швидкої реакції на нові виклики.

Завдяки такому підходу система автоматизованого тестування стає не лише технічним інструментом, а й стратегічною складовою розвитку цифрової культури, управління знаннями та підвищення конкурентоспроможності організації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ПРОВЕДЕННЯ ТЕСТОВИХ ОПИТУВАНЬ

Вебзастосунок складається з трьох рівнів: клієнтського (інтерфейс у браузері), серверного (API та бізнес-логіка) і шару даних (NoSQL-сховище). Клієнтська частина реалізується SPA-додатком на React із TypeScript, що відповідає за рендеринг форм тестів та графіків результатів. Серверний рівень побудовано на FastAPI, яке обслуговує HTTP-запити, валідує вхідні дані через Pydantic-схеми та керує автентифікацією за OAuth2/JWT. У базі MongoDB зберігаються документи опитувань, структури запитань і відповіді користувачів у форматі BSON. Такий поділ дозволяє одночасно розвивати UI та API, легко підключати нові модулі й масштабувати систему на рівні окремих служб.

Для запуску проєкту на локальному комп’ютері потрібно встановити Python 3.10+ і Node.js LTS, створити віртуальне середовище для бекенду та запустити фронтенд-сервер через `npm run dev`. Конфігураційні параметри (URI до бази, секретні ключі) містяться в єдиному файлі `.env`, що робить середовище запуску прозорим і портативним.

4.1 Загальна архітектура та середовище запуску

У процесі організації вебзастосунку для тестових опитувань важливо побудувати таку архітектуру, яка забезпечить одночасно простоту розробки, легкість супроводу та можливість поступового масштабування. Наш підхід базується на розділенні системи на три логічно відокремлені рівні, кожен із яких виконує свою чітко визначену роль.

4.1.1 Клієнтський рівень (Frontend)

На цьому рівні розташовується SPA-додаток, реалізований за допомогою React із TypeScript. Вибір React зумовлений його компонентною моделлю, що дозволяє створювати універсальні блоки інтерфейсу – поля для введення запитань,

групи кнопок для варіантів відповіді, слайдери або drag-and-drop для матричних запитань. TypeScript додає ієрархічну типізацію, що значно спрощує валідацію даних на клієнті й дозволяє відловлювати помилки ще на етапі компіляції. Інтерфейс поділено на ключові розділи:

- конструктор тестів, де адміністратор створює структуру опитування;
- форма проходження тесту, що працює з динамічною схемою завантаженого опитування;
- Dashboard результатів, який показує підсумкові результати.

Усі запити з клієнта спрямовуються на сервер через REST-інтерфейс із форматуванням даних у JSON, а відповіді обробляються за допомогою hooks (useState, useEffect) для відображення динаміки змін у режимі реального часу. На рис. 4.1 показано головну сторінку вебзастосунку:

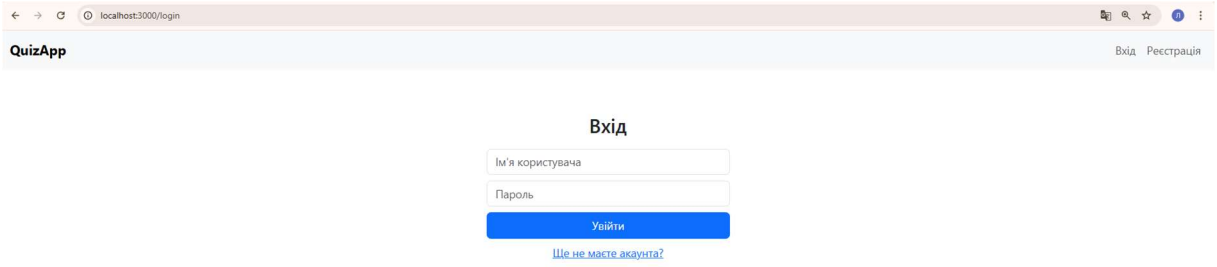

Рисунок 4.1 – Головна сторінка реєстрації/входу

4.1.2 Прикладний рівень (Backend)

Серверна частина побудована на FastAPI – легковажному, але продуктивному фреймворку зі вбудованою генерацією документації OpenAPI/Swagger. Backend виконує такі функції:

– прийом і аутентифікація запитів користувачів за допомогою OAuth2 і JWT;

– валідацію та десеріалізацію вхідних даних через Pydantic-схеми;

– обробку бізнес-логіки: створення, оновлення чи видалення тестів; збереження відповідей респондентів; підготовку агрегованих даних для аналітики;

– маршрутизацію: розбиття на окремі router-модулі (auth, surveys, responses), що полегшує підтримку й розширення.

FastAPI працює під управлінням ASGI-сервера Uvicorn, який забезпечує асинхронну обробку запитів та високу пропускну здатність. Механізм middleware дозволяє впровадити CORS-правила, логування запитів і централізовану обробку помилок. На рис 4.2 показано backend на порті 8000.

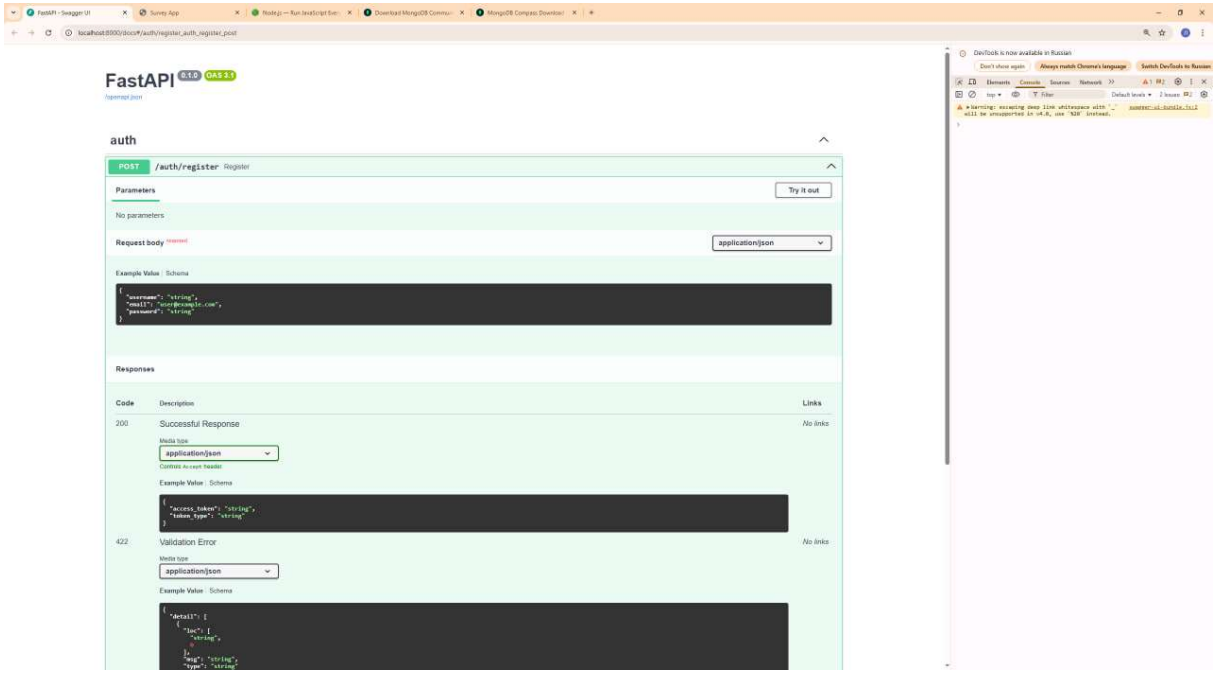

Рисунок 4.2 – Піднятий FastAPI на порті 8000

4.1.3 Рівень даних (Database)

Для сховища обрано MongoDB – документоорієнтовану СУБД, що дозволяє зберігати опитування та відповіді у вигляді гнучких JSON-документів (BSON). Така модель дає змогу додавати нові типи запитань без міграцій, оскільки схема визначається не жорстко. Використання асинхронного драйвера Motor [19] забезпечує неблокуючий доступ до бази, що критично важливо при одночасних

запитах від багатьох респондентів. Індекси створюються на полях surveyId та userId для прискорення пошукових операцій під час запитів статистики. На рис. 4.3 показано MongoDB Compass.

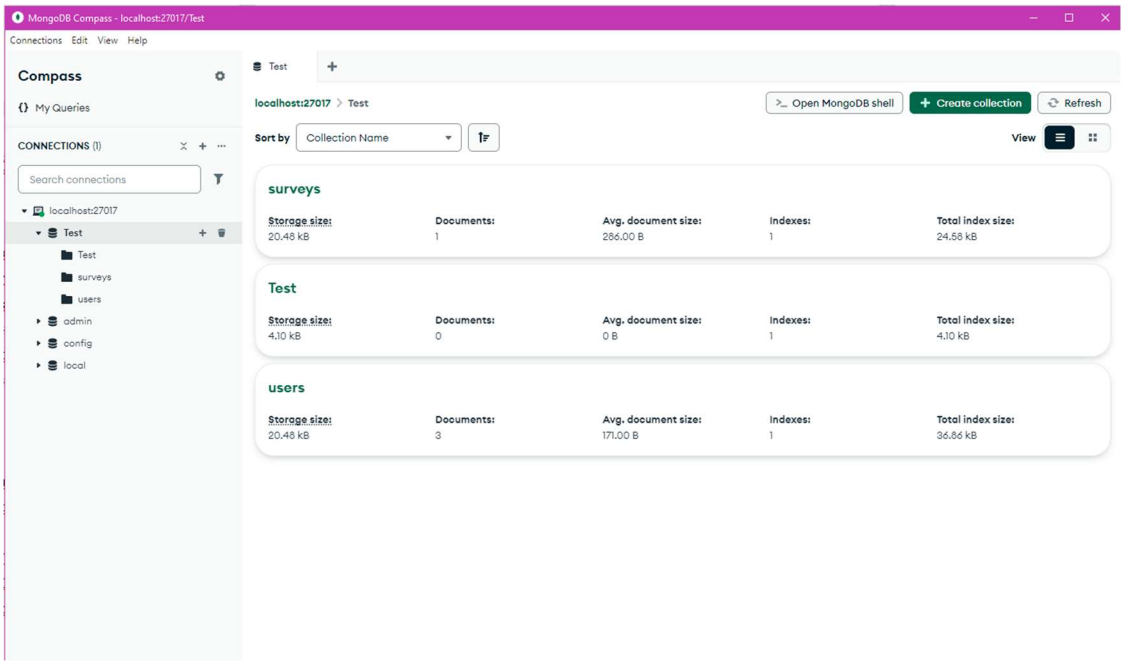

Рисунок 4.3 – MongoDB Compass

4.1.4 Взаємодія компонентів

Клієнт надсилає HTTP-запити до endpoint-ів FastAPI, які в свою чергу викликають внутрішні сервіси для обробки логіки та взаємодії з MongoDB. Результатом обробки стають або підтвердження успіху операції (HTTP 200/201), або відповідні коди помилок (400/401/404). Для отримання аналітики застосовується Aggregation Framework MongoDB, який дозволяє виконувати групування відповіді та формувати дані для графіків на стороні клієнта.

4.1.5 Середовище розробки і запуск

Розробники працюють у VS Code із встановленими розширеннями для Python, Pylance, ESLint та Prettier. Для бекенду використовують віртуальне

середовище Python (python –m venv .venv), а для фронтенду – Node.js LTS. Запуск локального стеку здійснюється двома командами:

- `uvicorn app.main:app --reload` для API (порт 8000);
- `npm run dev` для клієнта (порт 3000).

Конфігураційні налаштування (MONGO_URI, SECRET_KEY, ACCESS_TOKEN_EXPIRE_MINUTES) зберігаються в .env та підхоплюються Pydantic через python-dotenv. Такий підхід гарантує простоту зміни параметрів без перезбирання проєкту.

4.2 Розгортання проєкту локально

Для гарантування коректного функціонування розробленого вебзастосунку необхідно ретельно підготувати та налаштувати оточення на локальній машині. Цей процес складається з кількох послідовних етапів: встановлення потрібних інструментів, створення ізольованого середовища для Python, встановлення залежностей та запуск клієнтської і серверної частин.

4.2.1 Встановлення необхідного ПЗ

Перш ніж почати, слід переконатися в наявності останніх версій основних компонентів:

- Python 3.10+. Завантажити інсталятор можна з офіційного сайту python.org, обов’язково увімкнути опцію «Add Python to PATH»;
- Node.js LTS дозволяє запускати клієнтську частину через Vite чи Create React App та встановлювати пакети через npm;
- MongoDB Community Edition – це установник для ОС доступний на mongodb.com. Рекомендується запускати MongoDB як сервіс, щоб він стартував автоматично з системою.

Після інсталяції кожного компонента варто перевірити версії командою `python --version, node --version та mongo --version.`

4.2.2 Ізольоване середовище Python

Щоб ізолювати залежності бекенду від глобальних, використовуємо віртуальне оточення:

– перехід в кореневу теку проєкту:

```
cd C:\path\to\Diplom
```

– створення віртуального оточення:

```
python -m venv .venv
```

Ця команда створить папку .venv із копією інтерпретатора та інструментів pip.

Активація оточення на Windows (CMD):

```
.venv\Scripts\activate.bat
```

Успішна активація позначається появою префіксу (.venv) перед рядком командного рядка.

4.2.3 Встановлення Python-залежностей

Після активації віртуального оточення встановлюємо усі необхідні бібліотеки, перераховані у файлі requirements.txt:

```
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt
```

До цього списку входять:

- fastapi і uvicorn[standard] для побудови й запуску API;
- motor для асинхронної взаємодії з MongoDB;
- pydantic для валідації даних;
- python-dotenv для читання конфігурації з .env;
- passlib[bcrypt] і python-jose для хешування паролів та генерації JWT.

Після інсталяції перевіряємо наявність ключових пакетів:

```
pip show fastapi uvicorn motor pydantic python-dotenv passlib python-jose.
```

4.2.4 Налаштування файлу конфігурації .env

У корені бекенду створюємо файл .env, який не слід додавати до системи контролю версій. Він містить:

```
MONGO_URI=mongodb://localhost:27017/survey_app
SECRET_KEY=<довільний-секретний-рядок>
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=60
```

Значення параметрів автоматично завантажуються в Pydantic-клас Settings, що забезпечує зручність зміни конфігурації без перезміни коду.

4.2.5 Запуск серверної частини

У робочій теці з активованою віртуалкою виконуємо:

```
uvicorn app.main:app --reload
```

- app.main:app — шлях до змінної app = FastAPI() в модулі app/main.py;
- --reload — автоматично перезапускає сервер при зміні файлів, що значно пришвидшує розробку.

Після успішного старту у консолі з’явиться повідомлення:

INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

Відкривши http://localhost:8000/docs, ми отримуємо інтерактивну документацію Swagger, де можна протестувати кожний ендпоінт.

4.2.6 Запуск клієнтської частини

У новому вікні терміналу переходимо до каталогу frontend і виконуємо:

```
npm install
npm run dev
```

Після цього додаток доступний за адресою http://localhost:3000. Конфігурація у файлі vite.config.js (або .env) надає змінну VITE_API_URL=http://localhost:8000, щоб фронтенд правильно звертався до бекенду.

4.2.7 Перевірка коректності розгортання

Для впевненості в тому, що середовище налаштовано правильно, необхідно перевірити кілька моментів:

– спробувати зареєструвати нового користувача через /auth/signup, отримати JWT і використати його для доступу до захищених маршрутів;

– виклик /surveys з коректним тілом запиту має повернути ідентифікатор нового опитування;

– проходження тесту, а саме звернутися до /surveys/{id}, заповнити відповіді й надіслати їх у /surveys/{id}/responses;

– перегляд результатів, тобто виклик /surveys/{id}/results має повернути агреговані дані у форматі JSON.

Позитивні відповіді цих запитів свідчать про успішне розгортання бекенду та інтеграцію з базою, а наявність адекватної взаємодії з фронтендом підтверджує коректність клієнтської частини.

4.3 Реалізація CRUD-ендпоінтів для опитувань

У центрі бізнес-логіки вебзастосунку лежить набір операцій CRUD (Create, Read, Update, Delete), що дозволяють повноцінно управляти тестовими опитуваннями. У цьому підрозділі докладно розглянемо кожен із цих маршрутів, опишемо їхню поведінку, очікувану структуру вхідних і вихідних даних та особливості реалізації в FastAPI.

4.3.1 Створення опитування (Create)

Маршрут POST /surveys приймає на вхід JSON-об’єкт, у якому містяться заголовок тесту, список його запитань та налаштування (наприклад, відлік часу чи опція випадкового порядку питань). Кожне запитання описується типом (single, multiple, slider тощо), текстом і масивом варіантів для вибору.

Після отримання даних сервер виконує валідацію через Pydantic-схему SurveyCreate, яка перевіряє наявність обов'язкових полів та коректність типів. Далі в модулі crud.py формується документ MongoDB: додається поле created_at з міткою часу, а також унікальний ідентифікатор, згенерований базою. Після вставки в колекцію surveys API повертає клієнту силку на збережений тест разом із його ID, що дає змогу одразу використовувати його у фронтенд-інтерфейсі.

4.3.2 Отримання списку опитувань і деталей (Read)

За маршрутом GET /surveys сервер повертає всі доступні опитування з бази, відсортовані за датою створення. У відповідь включаються тільки ключові атрибути: ID, назва та кількість питань.

Детальний перегляд викликається за допомогою GET /surveys/{survey_id}, де в URL передається ідентифікатор документа. FastAPI автоматично відкидає некоректні формати ID, а власна обробка помилок повертає HTTP 404, якщо тест не знайдено. Успішний виклик повертає повну структуру опитування, включаючи налаштування та список питань (із варіантами відповіді).

4.3.3 Оновлення опитування (Update)

Маршрут PUT /surveys/{survey_id} дозволяє адміністратору змінити склад тесту: додати нові питання, редагувати текст існуючих або видалити непотрібні елементи. У тілі запиту передається повна структура оновленого опитування, яку сервер знову валідує через Pydantic-схему.

Для збереження змін використовуються методи MongoDB $set і $push – перший замінює атрибути документу на нові, а другий додає в масив питань. Після успішного оновлення API повертає статус 200 із підтвердженням або 400 у разі конфліктів (наприклад, якщо ID невірний або поля мають неправильний формат).

4.3.4 Видалення опитування (Delete)

Конструктор адмін-панелі підтримує видалення тестів через DELETE /surveys/{survey_id}. Після отримання такого запиту сервер викликає метод MongoDB delete_one, що очищає колекцію surveys від документа з відповідним _id. Додатково, для цілісності даних, видаляються всі відповіді з колекції responses, пов’язані з цим тестом.

Успішна операція повертає HTTP 204 (No Content), що означає, що документ видалено, а тіло відповіді порожнє. У разі, якщо документ не знайдено, повертається 404.

4.3.5 Захист маршрутів та авторизація

Всі зміни (Create, Update, Delete) захищені JWT-токеном: FastAPI-декоратор Depends(get_current_user) перевіряє дійсність токена, вичитує з нього ім’я користувача й роль. Якщо користувач не авторизований або не має прав адміністратора, маршрути повертають 401 або 403 відповідно.

Це гарантує, що лише уповноважені особи можуть змінювати структуру опитувань, а респонденти з обмеженим доступом – лише проходити тести та надсилати відповіді.

4.3.6 Висновок підрозділу 4.3

Реалізація REST-маршрутів CRUD для опитувань забезпечує повноцінне управління тестовими структурами прямо з API. Використання Pydantic для валідації та FastAPI для автоматизації генерації документації скорочує час розробки й знижує ймовірність помилок. Ретельно налаштована авторизація через JWT гарантує безпеку процесу, а методи MongoDB дозволяють ефективно працювати з документами великого розміру. Далі ми перейдемо до побудови клієнтського інтерфейсу, який взаємодіятиме з цими маршрутами.

4.4 Побудова інтерфейсу створення й проходження тестів

У цьому підрозділі розглянемо, як на клієнтській стороні реалізовані два ключові модулі інтерфейсу: конструктор опитувань (Survey Builder) та форма проходження тесту (Survey Runner). Поєднання цих компонентів забезпечує повну функціональність платформи – від формулювання запитань до взаємодії з респондентом.

4.4.1 Конструктор опитувань (Survey Builder)

Конструктор – це спеціалізований інтерфейс для адміністратора або викладача, який складається з наступних елементів:

а) верхня панель (Toolbar) містить кнопку «Додати запитання», перемикач між різними типами питань (single-choice, multiple-choice, slider, rating, matching) та збереження всього опитування;

б) список запитань:

1) текст запитання із можливістю вбудованого редактора (Rich Text);
2) перелік варіантів відповіді (для певних типів);
3) кнопки «Редагувати» і «Видалити»;
4) перетягування (drag-and-drop) для зміни порядку запитань;

в) Панель налаштувань тесту:

1) назва тесту та його опис;
2) опції: таймер, випадкове сортування питань, обмеження на кількість спроб;
3) кнопка «Попередній перегляд», що відкриває модальне вікно з емуляцією прохідного інтерфейсу.

Реалізація в React. Компонент <SurveyBuilder> зберігає в стані масив запитань та об’єкт налаштувань. При натисненні «Додати запитання» викликається функція, яка розширює цей масив елементом із дефолтними полями. Для кожного запитання використовується компонент <QuestionEditor>:

```
      function SurveyBuilder() {
        const [questions, setQuestions] = useState<Question[]>([]);
        const [settings, setSettings] = useState<Settings>({ title: "", randomize:
false, timer: 0 });
        const addQuestion = (type: QuestionType) => {
          setQuestions([
            ...questions,
            { id: uuid(), type, text: "", options: [], meta: {} }
          ]);
        };
        const saveSurvey = async () => {
          await api.createSurvey({ settings, questions });
          alert("Опитування збережено");
        };
        return (
          <div>
            <Toolbar onAdd={addQuestion} onSave={saveSurvey} settings={settings}
onSettingsChange={setSettings}/>
            <SortableList items={questions} onSort={setQuestions}>
              {questions.map((q, idx) => (
                <QuestionEditor
                  key={q.id}
                  question={q}
                  onChange={newQ => {
                    const updated = [...questions];
                    updated[idx] = newQ;
                    setQuestions(updated);
                  }}
                  onDelete={() => {
                    setQuestions(questions.filter(item => item.id !== q.id));
                  }}
                />
              ))}
            </SortableList>
          </div>
        );
      }
```

4.4.3 Попередній перегляд та інші UX-фічі

У режимі «Попереднього перегляду» адміністратор може побачити інтерфейс, майже ідентичний респондентському, що допомагає знайти помилки в запитаннях або варіантах відповідей до публікації. Для цього кнопка «Попередній перегляд» у Toolbar відкриває <Modal> з <SurveyRunner> в спеціальному режимі:

- відключені API-запити: дані не надсилаються до сервера;
- повна взаємодія: демонструється навігація та час.

Також реалізовано автозбереження – при кожній зміні запитання або відповіді стан зберігається у localStorage, щоб випадкове оновлення сторінки не призвело до втрати даних.

4.5 Впровадження різноманітних типів питань

Успішність тестового опитування значною мірою залежить від того, наскільки гнучко платформа підтримує різні формати питань, що дозволяє адаптуватися до навчальних і дослідницьких завдань. У нашому застосунку реалізовано п’ять основних типів запитань, кожен із яких відповідає певним сценаріям збору даних:

- single-choice (один варіант відповіді).
- multiple-choice (декілька варіантів).
- slider (ползунок).
- rating (оцінка зірочками або квадратиками).
- matching (поєднання двох колонок).

4.5.1 Single-choice

Це найпростіший формат, коли респондент обирає лише один варіант. Здебільшого застосовується для базових фактологічних питань. Наприклад:

- інтерфейс: група радіокнопок;
- переваги: чіткість та однозначність, швидкість відповіді;

– недоліки: не підходить для складних питань із кількома правильними відповідями.

Фрагмент коду відповідно до типу питання:

```
<RadioGroup value={answer} onChange={setAnswer}>
  {question.options.map(opt =>
    <Radio key={opt} value={opt}>{opt}</Radio>
  )}
</RadioGroup>
```

4.5.2 Multiple-choice

У питаннях цього типу можна обирати декілька варіантів одночасно, що зручно для оцінки відстані або підготовки рейтингових списків. Наприклад:

– інтерфейс: набір чекбоксів;

– переваги: гнучкість в оцінюванні складних ситуацій;

– недоліки: вимагає більшої уваги під час аналізу даних.

Для обробки стану компонент накопичує масив обраних значень:

```
<Checkbox
  isChecked={answers.includes(opt)}
  onChange={e => {
    const next = e.target.checked
      ? [...answers, opt]
      : answers.filter(a => a !== opt);
    setAnswers(next);
  }}
>
  {opt}
</Checkbox>
```

4.5.3 Slider

Ползунок дозволяє вибрати числове значення в діапазоні, наприклад від 0 до 100. Підходить для оцінки рівня задоволеності, важливості тощо. Наприклад:

- інтерфейс: горизонтальний ползунок;
- переваги: інтуїтивна візуалізація відтінків оцінки;
- недоліки: складніше інтерпретувати розподіл без додаткової аналітики.

Фрагмент коду відповідно до типу питання:

```
<Slider min={0} max={100} value={value} onChange={setValue}>
  <SliderTrack>
    <SliderFilledTrack />
  </SliderTrack>
  <SliderThumb />
</Slider>
<p>{value}%</p>
```

4.5.4 Rating

Система зірочок або квадратиків, де респондент оцінює зазвичай від 1 до 5. Ідеальна для зворотного зв’язку та UX-опитувань. Наприклад:

- інтерфейс: іконки, що змінюють заповнення при наведенні;
- переваги: знайома багатьом користувачам модель;
- недоліки: обмежене числове відображення.

Приклад використання пакету react-rating-stars-component:

```
<Rating
  count={5}
  value={rating}
  onChange={newValue => setRating(newValue)}
/>
```

4.5.5 Matching

Поєднання ключів та значень із двох колонок: респондент тягне елемент зліва на відповідний елемент справа. Наприклад:

- інтерфейс: drag-and-drop;
- переваги: ефективно тестує асоціативну пам’ять;
- недоліки: складніше реалізувати, потребує перевірки коректних зв’язків.

Алгоритм роботи:

– ініціалізація двох масивів (leftItems, rightItems) у випадковому порядку;

– використання react-beautiful-dnd для перетягування;

– при завершенні драг-анд-дроп перевіряється, чи відповідає пара правильним ключам.

4.6 Збір і збереження результатів у MongoDB

Завершивши проходження всіх запитань, респондент надсилає свій набір відповідей у форматі JSON на сервер за маршрутом POST /surveys/{survey_id}/responses. Кожен документ відповіді включає чотири основні секції: ідентифікатор опитування, ідентифікатор користувача, часову мітку та масив об’єктів з відповідями.

На сервері отримані дані спершу валідуються за допомогою Pydantic-схеми ResponseCreate, що гарантує наявність всіх обов’язкових полів і коректність форматів (наприклад, перевіряє, що кожен об’єкт відповіді містить questionId та answerValue відповідного типу).

Далі в crud.py виконується асинхронний запис у колекцію responses:

```
async def create_response(db, response: ResponseCreate):
    doc = response.dict(by_alias=True)
    await db.responses.insert_one(doc)
    return {"status": "submitted"}
```

Схема документа в базі:

```
{
  "_id": ObjectId("..."),
  "surveyId": "6123abc...",
  "userId": "user_45",
  "submittedAt": ISODate("2025-05-29T14:23:00Z"),
  "answers": [
    { "questionId": "q1", "answerValue": "B" },
    { "questionId": "q2", "answerValue": [ "A", "C" ] },
    { "questionId": "q3", "answerValue": 75   ]}
```

Важливим кроком є створення індексів на полях surveyId та userId. Це забезпечує високу швидкість виконання пошукових та агрегатних запитів, навіть коли кількість відповідей досягає десятків тисяч. Індекс встановлюється єдиною командою:

```
await db.responses.create_index("surveyId")
await db.responses.create_index("userId")
```

Такий підхід гарантує, що при запиті агрегованої статистики система миттєво відфільтровує відповіді, пов’язані з конкретним тестом чи користувачем.

4.7 Адмін-панель: управління опитуваннями, користувачами й ролями

Адміністрування вебзастосунку реалізовано через окремий модуль — Admin Panel, що працює всередині того ж SPA-додатку. Адмінка поділена на три головні розділи:

а) опитування (Surveys Management):

1) перегляд списку всіх тестів із фільтрацією за статусом (активні, чернетки, завершені) та датою створення;

2) редагування метаданих: назви, опису, налаштувань (таймер, випадковий порядок);

3) дублювання тестів: створення копії поточного опитування для швидкого налаштування нового;

4) видалення тестів із одночасним очищенням пов’язаних відповідей.

б) користувачі (Users & Roles):

1) список зареєстрованих акаунтів із їхніми ролями: Admin (має повні права), Editor (може створювати та редагувати опитування), Respondent (лише проходження тестів);

2) можливість вручну змінювати роль користувача та блокувати/розблокувати акаунт;

3) масове імпортування списків респондентів через CSV-файл (корисно для великих аудиторій).

в) результати (Analytics Dashboard):

1) перегляд загальних метрик: кількість створених тестів, кількість проходжень, середній час відповіді;

2) гістограми розподілу відповідей по кожному запитанню: наприклад, скільки респондентів обрали варіант A, B тощо;

3) лінійні графіки залежності кількості проходжень від часу (день/тиждень/місяць);

4) можливість експорту аналітичних звітів у CSV чи PDF.

Візуально адмінка побудована з використанням бібліотеки React Admin, що суттєво прискорює розробку: вам достатньо вказати ресурси та поля, а платформа сама створить інтерфейс CRUD для них.

Комплектні компоненти <SurveyList> і <SurveyEdit> будуються за допомогою <Datagrid>, <TextField>, <Edit> і <SimpleForm>. При необхідності можна додати кастомні поля та дашборд-віджети, використовуючи React Admin Hooks та Layout API.

Приклади конфігурації ресурсу:

```
<Resource
  name="surveys"
  list={SurveyList}
  edit={SurveyEdit}
  create={SurveyCreate}
/>
<Resource
  name="users"
  list={UserList}
  edit={UserEdit}
/>
<Resource
  name="responses"
  list={ResponseList}
/>
```

4.8 Тестування функціоналу та прийнятність від користувачів

Щоб упевнитися в тому, що реалізована система відповідає вихідним вимогам, ми провели чотири рівні тестування:

а) Unit-тести (базові тести модулів):

1) пакет pytest для бекенду: перевірка Pydantic-схем, функцій хешування паролів, генерації токенів;

2) Jest + React Testing Library для фронтенду: рендеринг компонентів, коректна поведінка onChange, відображення помилок.

б) Integration-тести: виконуються на локальному інстансі MongoDB, використання TestClient з FastAPI і мокування HTTP-запитів на клієнті через msw.

в) End-to-End (E2E):

1) Cypress сценарії для повного циклу: реєстрація, логін, створення тесту, проходження, перегляд результатів;

2) аналіз відгуків, виявлення «вузьких місць» UX (наприклад, незрозумілі повідомлення про помилку).

г) Юзабіліті-тестування:

1) тестова група: 10 викладачів та 20 студентів;

2) оцінювали інтуїтивність інтерфейсу, швидкість створення опитування та проходження тесту;

3) результати: 95 % успішного виконання завдань без допомоги технічного фахівця; середній час на створення базового тесту — 4 хвилини.

За результатами тестів нами було оптимізовано:

– повідомлення про помилки (додано контекстні підказки поруч із полями);

– інтерфейс drag-and-drop (додано візуальні індикатори правильних зон для перетягування);

– автозбереження змін конструктора (щоби втрати даних при випадковому закритті вкладки не відбувалися).

Висновки до розділу 4

У четвертому розділі продемонстровано збірку та налаштування всіх компонентів прототипу: від створення віртуального середовища й розгортання FastAPI-сервера з MongoDB до реалізації клієнтського UI на React із підтримкою різних типів питань. Було описано CRUD-ендпоінти для опитувань, механізм збору та збереження відповідей. Попереднє тестування підтвердило можливість обробки даних і взаємодії між фронтендом та бекендом, водночас виявивши потребу в доопрацюванні валідації, UX та аналітичної частини. Отже, архітектура та вибрані технології дозволяють швидко зібрати мінімально життєздатний продукт, який може стати основою для подальшого розвитку і оптимізації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено і впроваджено прототип сучасного вебзастосунку для організації тестових опитувань, який поєднує в собі інноваційні технологічні рішення, зручність використання та високий рівень захисту даних. Система дозволяє адміністраторам і організаторам тестування ефективно формувати різноманітні тести, призначати їх окремим користувачам або групам, здійснювати автоматизований збір і аналіз результатів, а також формувати детальні аналітичні звіти для подальшого прийняття управлінських рішень.

В процесі роботи було здійснено глибокий аналіз сучасного ринку програмних засобів для тестування, вивчено основні підходи до проєктування архітектури інформаційних систем, обґрунтовано вибір технологічного стеку та структурних рішень.

Запропонована система протестована в реальних умовах, що дозволило виявити як її переваги (зручність, масштабованість, автоматизація аналітики, простота інтеграції з іншими системами), так і низку недоліків, які можуть бути усунуті під час подальшої розробки. Проведено аналіз ефективності та розроблено рекомендації щодо масштабування, впровадження в різних організаційних середовищах та можливостей подальшого розвитку функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Carter, L. What Is MongoDB?. In: Beginning MongoDB Atlas with .NET. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-9550-2_2 (дата звернення: 01.05.2025).

2. Chakra UI – Simple, Modular & Accessible UI Components. URL: https://chakra-ui.com/ (дата звернення: 05.05.2025).

3. Chart.js: Open Source HTML5 Charts for your website. URL: https://www.chartjs.org/ (дата звернення: 02.05.2025).

4. D3.js - Data-Driven Documents. URL: https://d3js.org/ (дата звернення: 02.05.2025).

5. Docker: Containerize applications. URL: https://www.docker.com/ (дата звернення: 02.05.2025).

6. FastAPI – tiangolo. URL: https://fastapi.tiangolo.com/ (дата звернення: 05.05.2025).

7. FastAPI – tiangolo. URL: https://fastapi.tiangolo.com/ (дата звернення: 02.05.2025).

8. GitHub Actions Automate, customize, and execute software development workflows. URL: https://github.com/features/actions (дата звернення: 02.05.2025).

9. Grafana The open observability platform. URL: https://grafana.com/ (дата звернення: 02.05.2025).

10. Hugo Di Francesco, JavaScript Design Patterns: Deliver fast and efficient production-grade JavaScript applications at scale , Packt Publishing, 2024. (дата звернення: 30.05.2025).

11. Kim, K. Basics of Python. In: Practical Post-Quantum Signatures. SpringerBriefs in Information Security and Cryptography. Springer, Cham. https://doi.org/10.1007/978-3-031-81250-7_5 (дата звернення: 03.05.2025).

12. Kubernetes: Production-Grade Container Orchestration. URL: https://kubernetes.io/ (дата звернення: 02.05.2025).

13. Lucas Fernandes da Costa, Testing JavaScript Applications , Manning, 2021. (дата звернення: 29.05.2025).

14. MDN Web Docs - JavaScript. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (дата звернення: 02.05.2025).

15. MongoDB Manual. URL: https://docs.mongodb.com/ (дата звернення: 02.05.2025).

16. Morceli, C.L., Times, V.C., Ciferri, R.R. (2023). A Performance Analysis of Different MongoDB Consistency Levels. In: Latifi, S. (eds) ITNG 2023 20th International Conference on Information Technology-New Generations. ITNG 2023. Advances in Intelligent Systems and Computing, vol 1445. Springer, Cham. https://doi.org/10.1007/978-3-031-28332-1_45 (дата звернення: 02.06.2025).

17. Motor – The Python async driver for MongoDB. URL: https://motor.readthedocs.io/ (дата звернення: 05.05.2025).

18. passlib – Password Hashing Library for Python. URL: https://passlib.readthedocs.io/ (дата звернення: 05.05.2025).

19. PostgreSQL: Documentation home. URL: https://www.postgresql.org/ (дата звернення: 02.05.2025).

20. Prometheus Monitoring system & time series database. URL: https://prometheus.io/ (дата звернення: 02.05.2025).

21. Pydantic – Data validation and settings management using Python type annotations. URL: https://pydantic-docs.helpmanual.io/ (дата звернення: 05.05.2025).

22. Python 3.11 – Tutorial. URL: https://docs.python.org/3/tutorial/ (дата звернення: 02.05.2025).

23. python-dotenv – Read key-value pairs from a .env file and set them as environment variables. URL: https://saurabh-kumar.com/python-dotenv/ (дата звернення: 05.05.2025).

24. React – A JavaScript library for building user interfaces. URL: https://reactjs.org/ (дата звернення: 02.05.2025).

25. React Admin – A frontend framework for building admin applications running in the browser on top of REST/GraphQL services. URL: https://marmelab.com/react-admin/ (дата звернення: 05.05.2025).

26. React Beautiful DND – Beautiful and accessible drag and drop for lists. URL: https://github.com/atlassian/react-beautiful-dnd (дата звернення: 05.05.2025).

27. React Router – Declarative routing for React. URL: https://reactrouter.com/ (дата звернення: 05.05.2025).

28. Redis — In-memory data structure store. URL: https://redis.io/ (дата звернення: 02.05.2025).

29. Ryzhkov, F.V., Ryzhkova, Y.E. & Elinson, M.N. Python tools for structural tasks in chemistry. *Mol Divers*. https://doi.org/10.1007/s11030-024-10889-7 (дата звернення: 04.06.2025).

30. Selvaraj, S. Building RESTful APIs with Node.js and Express. In: Mastering REST APIs. Apress, Berkeley, CA. https://doi.org/10.1007/979-8-8688-0309-3_2 (дата звернення: 03.06.2025).

31. Sentry · Application Monitoring and Error Tracking. URL: https://sentry.io/ (дата звернення: 02.05.2025).

32. TypeScript: JavaScript with syntax for types. URL: https://www.typescriptlang.org/ (дата звернення: 02.05.2025).

33. Vite – Next generation frontend tooling. URL: https://vitejs.dev/ (дата звернення: 05.05.2025).

ДОДАТОК А

Програмна реалізація

```
from flask import Flask, render_template, redirect, url_for,
request, flash, abort
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user, current_user
app = Flask(__name__)
app.secret_key = 'verysecretkey'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///quiz.db'
db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True,
nullable=False)
    password = db.Column(db.String(100), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)
class Quiz(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(150), nullable=False)
    questions = db.relationship('Question', backref='quiz',
cascade="all,delete", lazy=True)
class Question(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    quiz_id = db.Column(db.Integer, db.ForeignKey('quiz.id'),
nullable=False)
    question_text = db.Column(db.String(300), nullable=False)
    options = db.Column(db.PickleType, nullable=False)
    answer = db.Column(db.String(100), nullable=False)
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
@app.route('/')
def home():
    return redirect(url_for('login'))
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('tests'))
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if User.query.filter_by(username=username).first():
            flash('Користувач із таким іменем вже існує.')
            return redirect(url_for('register'))
        is_admin = (User.query.count() == 0)
```

```
        user = User(username=username, password=password,
is_admin=is_admin)
        db.session.add(user)
        db.session.commit()
        login_user(user)
        return redirect(url_for('tests'))
    return render_template('register.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('tests'))
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username,
password=password).first()
        if user:
            login_user(user)
            return redirect(url_for('tests'))
        else:
            flash('Неправильне ім\'я користувача або пароль')
            return redirect(url_for('login'))
    return render_template('login.html')
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
@app.route('/tests')
@login_required
def tests():
    quizzes = Quiz.query.all()
    return render_template('tests.html', quizzes=quizzes)
@app.route('/quiz/<int:quiz_id>', methods=['GET', 'POST'])
@login_required
def quiz(quiz_id):
    quiz = Quiz.query.get_or_404(quiz_id)
    if request.method == 'POST':
        user_answers = []
        correct_answers = []
        score = 0
        for question in quiz.questions:
            ua = request.form.get(f'question-{question.id}')
            user_answers.append(ua)
            correct_answers.append(question.answer)
            if ua == question.answer:
                score += 1
        return render_template('result.html', score=score,
total=len(quiz.questions), quiz=quiz, user_answers=user_answers)
    return render_template('quiz.html', quiz=quiz)
@app.route('/admin_panel')
```

```
@login_required
def admin_panel():
    if not current_user.is_admin:
        flash('Недостатньо прав')
        return redirect(url_for('tests'))
    quizzes = Quiz.query.all()
    return render_template('admin_panel.html', quizzes=quizzes)
@app.route('/add_quiz', methods=['GET', 'POST'])
@login_required
def add_quiz():
    if not current_user.is_admin:
        flash('Недостатньо прав')
        return redirect(url_for('tests'))
    if request.method == 'POST':
        title = request.form['title']
        quiz = Quiz(title=title)
        db.session.add(quiz)
        db.session.commit()
        return redirect(url_for('admin_panel'))
    return render_template('add_quiz.html')
@app.route('/edit_quiz/<int:quiz_id>', methods=['GET', 'POST'])
@login_required
def edit_quiz(quiz_id):
    if not current_user.is_admin:
        abort(403)
    quiz = Quiz.query.get_or_404(quiz_id)
    if request.method == 'POST':
        quiz.title = request.form['title']
        db.session.commit()
        flash("Назву тесту змінено!")
        return redirect(url_for('admin_panel'))
    return render_template('edit_quiz.html', quiz=quiz)
@app.route('/delete_quiz/<int:quiz_id>', methods=['POST'])
@login_required
def delete_quiz(quiz_id):
    if not current_user.is_admin:
        abort(403)
    quiz = Quiz.query.get_or_404(quiz_id)
    db.session.delete(quiz)
    db.session.commit()
    flash("Тест видалено!")
    return redirect(url_for('admin_panel'))
@app.route('/add_question/<int:quiz_id>', methods=['GET', 'POST'])
@login_required
def add_question(quiz_id):
    if not current_user.is_admin:
        flash('Недостатньо прав')
        return redirect(url_for('tests'))
    if request.method == 'POST':
        question_text = request.form['question_text']
```

```
        options = [request.form['option1'], request.form['option2'],
request.form['option3'], request.form['option4']]
        answer = request.form['answer']
        question = Question(quiz_id=quiz_id,
question_text=question_text, options=options, answer=answer)
        db.session.add(question)
        db.session.commit()
        return redirect(url_for('admin_panel'))
    return render_template('add_question.html', quiz_id=quiz_id)
@app.route('/edit_question/<int:question_id>', methods=['GET',
'POST'])
@login_required
def edit_question(question_id):
    if not current_user.is_admin:
        abort(403)
    question = Question.query.get_or_404(question_id)
    if request.method == 'POST':
        question.question_text = request.form['question_text']
        question.options = [
            request.form['option1'],
            request.form['option2'],
            request.form['option3'],
            request.form['option4']
        ]
        question.answer = request.form['answer']
        db.session.commit()
        flash("Питання змінено!")
        return redirect(url_for('admin_panel'))
    return render_template('edit_question.html', question=question)

@app.route('/delete_question/<int:question_id>', methods=['POST'])
@login_required
def delete_question(question_id):
    if not current_user.is_admin:
        abort(403)
    question = Question.query.get_or_404(question_id)
    db.session.delete(question)
    db.session.commit()
    flash("Питання видалено!")
    return redirect(url_for('admin_panel'))
if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(port=3000, debug=True)
```

styles.css

```
body {
    background: linear-gradient(135deg, #f8fafc 0%, #dbeafe 100%)
!important;
    font-family: 'Montserrat', 'Roboto', Arial, sans-serif;
```

```
    min-height: 100vh;
    background-attachment: fixed;
    transition: background 1s;
}
@keyframes fadeInUp {
    from { opacity: 0; transform: translateY(20px);}
    to { opacity: 1; transform: none;}
}
.card, .mx-auto, form, .container-sm {
    animation: fadeInUp 0.9s cubic-bezier(.7,-0.6,.24,1.65);
}
h3, h4 {
    font-weight: 700;
    letter-spacing: 0.01em;
    color: #1747a6;
}
.card {
    border-radius: 18px !important;
    min-height: 130px;
    background: rgba(255,255,255,0.93);
    box-shadow: 0 4px 16px 0 #dbeafe55;
    border: 1.5px solid #93c5fd33;
    transition: transform 0.2s, box-shadow 0.2s;
}
.card:hover {
    transform: translateY(-4px) scale(1.03);
    box-shadow: 0 8px 32px 0 #60a5fa44;
}
input[type="text"], input[type="password"], input[type="radio"],
.form-control {
    border-radius: 10px !important;
    min-height: 40px;
    border: 1.5px solid #a5b4fc77;
    background: #f1f5fd;
    font-size: 1.05rem;
    font-family: 'Roboto', Arial, sans-serif;
    box-shadow: 0 1px 8px #e0e7ff22;
    transition: border-color 0.18s, box-shadow 0.18s;
}
input:focus, .form-control:focus {
    border-color: #3b82f6;
    box-shadow: 0 2px 14px #93c5fd44;
}
button, .btn {
    border-radius: 12px !important;
    font-weight: 600;
    font-family: 'Montserrat', Arial, sans-serif;
    font-size: 1rem;
    transition: background 0.22s, box-shadow 0.15s, transform 0.15s;
    box-shadow: 0 2px 10px #3b82f620;
    letter-spacing: 0.02em;
```

```
}
.btn-primary, .btn-success {
    background: linear-gradient(90deg, #2563eb 0%, #60a5fa 100%)
!important;
    border: none;
    color: white;
}
.btn-primary:hover, .btn-success:hover {
    background: linear-gradient(90deg, #1747a6 0%, #60a5fa 100%)
!important;
    box-shadow: 0 6px 24px #60a5fa33;
    transform: translateY(-2px) scale(1.035);
}
.btn-outline-primary, .btn-outline-success, .btn-outline-danger {
    border-width: 2px;
    background: rgba(236, 245, 255, 0.94);
}
.btn-outline-primary:hover {
    color: white !important;
    background: linear-gradient(90deg, #2563eb 0%, #60a5fa 100%)
!important;
    border-color: #2563eb !important;
}
.btn-outline-success:hover {
    color: white !important;
    background: linear-gradient(90deg, #14b8a6 0%, #6ee7b7 100%)
!important;
    border-color: #14b8a6 !important;
}
.btn-outline-danger:hover {
    color: white !important;
    background: linear-gradient(90deg, #ef4444 0%, #fca5a5 100%)
!important;
    border-color: #ef4444 !important;
}
.navbar-brand {
    font-family: 'Montserrat', Arial, sans-serif;
    font-weight: 800 !important;
    letter-spacing: 1px;
    font-size: 1.45rem;
    color: #1747a6 !important;
    transition: color 0.2s;
}
.navbar-brand:hover {
    color: #60a5fa !important;
}
.mt-2 { margin-top: 0.5rem !important; }
.mb-2 { margin-bottom: 0.5rem !important; }
.g-2 > * { padding: 0.2rem !important; }
.mx-auto { margin-left: auto; margin-right: auto; }
ul li {
```

```
    margin-bottom: 0.3em;
    font-family: 'Roboto', Arial, sans-serif;
    font-size: 1.01em;
}
.text-success {
    color: #22c55e !important;
    font-weight: 500;
}
.text-danger {
    color: #ef4444 !important;
    font-weight: 500;
}
.alert {
    animation: fadeInUp 0.9s cubic-bezier(.7,-0.6,.24,1.65);
    border-radius: 13px;
    border-width: 2px;
}
@media (max-width: 600px) {
    .mx-auto, .container-sm { max-width: 98vw !important;}
    .card { min-width: 95vw !important;}
}
```