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

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»__202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
Проєктування та реалізація веб-сервісу для створення опитувань, збору відповідей та базової аналітики на основі технології Node.js

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

Виконавець роботи Микитенко Віталій Іванович
_____«____» _____202_ р.
(підпис)

Науковий керівник доц., к.ф.-м.н. Парфьонова Тетяна Олександрівна
_____«___»_____202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2025

АНОТАЦІЯ

Записка: 71 сторінка, 15 рисунків, 4 таблиці, 1 додаток, 20 джерел.

Мета роботи – розробка веб-сервісу для автоматизованого створення, проходження та аналітичної обробки опитувань із використанням технологій штучного інтелекту.

Об'єкт розробки – процес організації, проведення та аналізу онлайн-опитувань.

Методи дослідження та інформаційне забезпечення – принципи клієнт–серверної архітектури, REST-підхід, технології JavaScript/TypeScript, хмарні та web-фреймворки (Node.js, Express.js, React), методи машинного навчання для генерації текстів, документоорієнтована база даних MongoDB, інструменти забезпечення автентифікації та безпеки (JWT, OAuth 2.0, bcrypt).

Результати дослідження. У роботі проведено аналіз сучасних систем для створення онлайн-опитувань, виявлено їхні обмеження та визначено вимоги до власного веб-сервісу. Спроєктовано та реалізовано архітектуру MERN-застосунку (MongoDB, Express, React, Node.js) з розподілом на фронтенд та бекенд. Розроблено конструктор опитувань із шістьма видами запитань, підтримкою логічних переходів, обмежень доступу, аналітичного перегляду результатів та експорту даних.

Інтегровано модулі штучного інтелекту Google Gemini для генерації опитувань, покращення запитань і аналізу відповідей. Реалізовано двотокенну систему автентифікації (Access/Refresh), захист даних, валідацію та механізми безпеки відповідно до сучасних вимог. У результаті отримано повністю функціональний, масштабований та продуктивний веб-сервіс, придатний до використання у практичній діяльності.

Наукова новизна полягає у застосуванні інтегрованого підходу до автоматизації створення й аналізу опитувань на основі веб-технологій та генеративних моделей ШІ, що забезпечує адаптивність і змістову оптимізацію анкет у режимі реального часу.

Рекомендації щодо використання результатів дослідження. Розроблений веб-сервіс може бути використаний у навчальному процесі, соціологічних

дослідженнях, HR-аналізі, маркетингових опитуваннях, а також у будь-яких організаціях, які потребують зручного та швидкого збору й обробки даних. Система може бути розширена новими модулями аналітики, інтелектуальними підсистемами або адаптована під корпоративні потреби.

Наукова апробація. Основні положення та результати дослідження доповідалися та обговорювалися на XLVIII Міжнародній науковій студентській конференції «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті» (Полтава, 2025) та XI Міжнародній науково-практичній конференції «Наука і молодь в XXI сторіччі» (Полтава, 2025).

КЛЮЧОВІ СЛОВА: ВЕБ-СЕРВІС, ОНЛАЙН-ОПИТУВАННЯ, MERN STACK, ШТУЧНИЙ ІНТЕЛЕКТ, GEMINI API, REST.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
AI	Штучний інтелект (Artificial Intelligence)
API	Програмний інтерфейс взаємодії між клієнтом і сервером (Application Programming Interface)
CORS	Технологія спільного використання ресурсів між різними джерелами (Cross-Origin Resource Sharing)
CSRF	Міжсайтова підробка запиту (Cross-Site Request Forgery)
Frontend	Клієнтська частина веб-додатку, що відповідає за інтерфейс та взаємодію з користувачем
HTTP	Протокол передачі гіпертексту (HyperText Transfer Protocol)
HTTPS	Захищений протокол передачі даних (HyperText Transfer Protocol Secure)
IP	Мережевий ідентифікатор пристрою (Internet Protocol address)
JSON	Формат обміну даними на основі JavaScript (JavaScript Object Notation)
JWT	Веб-токен для безпечної передачі даних (JSON Web Token)
MVC	Архітектурний патерн проектування (Model–View–Controller)

OAuth	Відкритий протокол авторизації (Open Authorization)
REST	Архітектурний стиль веб-сервісів (Representational State Transfer)
TLS	Криптографічний протокол захисту транспортного рівня (Transport Layer Security)
XSS	Міжсайтовий скриптінг (Cross-Site Scripting)
bcrypt	Алгоритм адаптивного криптографічного хешування паролів

ВСТУП

Актуальність теми. Ефективний збір та аналіз зворотного зв'язку від користувачів неможливий без застосування спеціалізованих веб-систем, що забезпечують автоматизацію процесів опитування та обробки результатів. Оперативне отримання структурованої інформації від респондентів і зменшення витрат часу на її аналіз вимагають раціональної організації процесів збору даних та обов'язкового використання сучасних веб-технологій з елементами штучного інтелекту. У даному дипломному проекті розглядаються етапи проектування архітектури веб-сервісу для проведення опитувань та його програмна реалізація на базі сучасного стеку технологій. Дана розробка є актуальною, адже використання власних систем опитувань у сучасному бізнес-середовищі – це вже не альтернатива, а критична необхідність для організацій, які прагнуть контролювати свої дані, оптимізувати процеси дослідження ринку та робити збір інформації ефективним для прийняття стратегічних рішень. Застосування подібних веб-сервісів дозволяє підвищити швидкість збору даних, знизити вартість проведення досліджень та забезпечити високу точність результатів завдяки автоматичній валідації та AI-аналізу.

Метою роботи є розробка та впровадження веб-сервісу для створення, проведення та аналізу результатів опитувань з використанням технологій MERN stack та штучного інтелекту.

Для досягнення мети поставлено такі завдання: – проаналізувати існуючі рішення для онлайн-опитувань; – спроєктувати архітектуру веб-застосунку та структуру бази даних; – реалізувати функціонал створення опитувань з використанням AI; – розробити механізми збору, валідації та аналізу відповідей; – протестувати розроблену систему.

Об'єктом розробки є процес організації, проведення та аналізу онлайн-опитувань.

Предметом розробки є методи та програмні засоби створення опитувань, збору даних та їх інтелектуальної обробки з використанням генеративного штучного інтелекту.

Методи дослідження. У роботі використано методи об’єктно-орієнтованого програмування, принципи REST-архітектури для взаємодії клієнта і сервера, методи системного аналізу для проєктування бази даних MongoDB, а також методи машинного навчання (через Google Gemini API) для генерації та аналізу текстового контенту.

Наукова новизна одержаних результатів полягає у застосуванні інтегрованого підходу до побудови систем опитувань, який, на відміну від існуючих аналогів, поєднує гнучку логіку переходів (Skip Logic) з генеративними можливостями штучного інтелекту для автоматичного створення анкет та інтерпретації відповідей у реальному часі.

Практичне значення одержаних результатів. Розроблений веб-сервіс дозволяє автоматизувати процес соціологічних та маркетингових досліджень, зменшити час на підготовку анкет та підвищити якість аналізу відкритих відповідей. Система може бути впроваджена у навчальних закладах, HR-відділах та маркетингових агенціях.

Апробація результатів. Основні положення роботи оприлюднено на наукових конференціях:

- XLVIII Міжнародна наукова студентська конференція «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті» (м. Полтава, 10 квітня 2025 р.).
- XI Міжнародна науково-практична конференція «Наука і молодь в XXI сторіччі» (м. Полтава, 10 листопада 2025 р.).

За результатами виконаних досліджень опубліковано 2 тези доповідей.

Структура роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 71 сторінку.

1. ПОСТАНОВКА ЗАДАЧІ

Основною задачею дипломної роботи є проєктування та програмна реалізація веб-сервісу для автоматизації процесів створення, проведення та аналізу опитувань. Актуальність розробки зумовлена необхідністю створення системи, яка поєднує гнучкість налаштувань, сучасні засоби аналітики та доступність, оскільки існуючі аналоги часто мають суттєві обмеження у безкоштовних версіях або характеризуються високим порогом входу. Метою роботи визначено створення програмного комплексу у вигляді веб-додатку, що дозволяє користувачам конструювати опитування зі складною логікою, збирати відповіді респондентів та отримувати автоматизовані аналітичні звіти з використанням технологій штучного інтелекту.

Вхідними даними для функціонування системи є структура опитування, яку задає автор, включаючи перелік запитань, типи відповідей та логіку переходів, а також параметри доступу, такі як часові межі проведення та обмеження кількості респондентів. Окрім того, система приймає текстовий опис теми для генерації структури опитування засобами штучного інтелекту та безпосередньо відповіді респондентів у різних форматах. Результатом роботи системи, або вихідними даними, є згенероване активне посилання на веб-форму для проходження опитування, сформована база даних зібраних відповідей, візуалізована статистика у вигляді графіків та діаграм на дашборді користувача, а також аналітичні висновки та інсайти, створені AI-модулем, із можливістю експорту звітів у форматі PDF.

Для досягнення поставленої мети система повинна забезпечувати виконання низки функціональних вимог, серед яких ключовим є конструювання опитувань із підтримкою різних типів питань, таких як текстове та числове поля, одиночний і множинний вибір, а також шкала рейтингу. Критично важливою є реалізація механізму умовної логіки (Skip Logic), що дозволяє налаштовувати динамічне відображення питань залежно від попередніх відповідей респондента. Система також має передбачати інтеграцію з Google Generative AI (Gemini) для автоматичної генерації питань на основі заданої теми, аналізу текстових відповідей та надання

інсайтів через інтерактивний чат. Окрім цього, функціонал повинен включати можливості для командної роботи через систему співавторства, валідацію відповідей у реальному часі, побудову графіків за допомогою бібліотеки Recharts та керування процесом опитування через налаштування обмежень і моніторинг активності.

З архітектурної точки зору система проєктується за клієнт-серверною моделлю з використанням технологічного стеку MERN. Серверна частина реалізується на базі Node.js та Express.js для забезпечення ефективної асинхронної обробки запитів, а як сховище даних використовується NoSQL база даних MongoDB, що дозволяє гнучко зберігати різноманітні структури опитувань. Клієнтська частина будується на основі бібліотеки React 19 із застосуванням мови TypeScript для забезпечення типобезпеки, інструменту Vite для швидкої збірки проєкту, Tailwind CSS для адаптивної стилізації інтерфейсу та менеджера стану Zustand. Взаємодія компонентів системи організовується згідно з архітектурним патерном MVC на стороні сервера та компонентним підходом на клієнті.

Вимоги до безпеки та надійності системи передбачають реалізацію механізму автентифікації на основі dual-token підходу з використанням JWT та httpOnly cookies, де Access Token має короткий термін життя, а Refresh Token забезпечує тривалу сесію. Захист даних забезпечується шляхом хешування паролів за алгоритмом bcrypt, інтеграції з Google OAuth 2.0 для спрощеного входу, використання бібліотеки Helmet.js для налаштування заголовків безпеки, а також конфігурації політик CORS та обмеження частоти запитів (rate-limiting). Система повинна демонструвати високу продуктивність із часом відгуку інтерфейсу до 100 мілісекунд та здатністю витримувати навантаження щонайменше 1000 одночасних користувачів без деградації якості обслуговування.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд існуючих рішень з аналогічними завданнями

У сучасній практиці створення онлайн-опитувань найбільш поширеними інструментами є Google Forms, SurveyMonkey, Typeform та LimeSurvey. Кожен із цих сервісів пропонує власний підхід до формування запитань, збору відповідей та аналізу результатів.

Google Forms є одним із найпоширеніших рішень для створення онлайн-опитувань, що входить до екосистеми Google Workspace. Завдяки простоті використання, доступності та інтеграції з іншими сервісами Google, цей інструмент набув великої популярності серед викладачів, студентів, дослідників та організацій, які потребують швидко створювати опитувальні форми без спеціальної технічної підготовки.

Сервіс пропонує базовий, але достатньо функціональний набір інструментів для створення анкет, включаючи текстові поля різних типів, запитання з варіантами вибору, шкали оцінювання, сітки, завантаження файлів та можливість додавання зображень або відеоматеріалів. Структура редактора інтуїтивна та не вимагає складних налаштувань: користувач може швидко додати нове запитання, змінити порядок пунктів або налаштувати просту логіку переходів між секціями.

Однією з ключових переваг Google Forms є швидкість створення опитування: навіть складну форму можна сформувати за кілька хвилин, використовуючи готові шаблони або додаючи блоки вручну. Крім того, сервіс забезпечує повну інтеграцію з Google Drive, що спрощує спільну роботу над формами, а також дозволяє керувати доступом і контролювати перегляд або редагування опитування іншими користувачами. Важливим елементом є автоматичне збереження відповідей у Google Sheets. Це дає змогу одразу переглядати результати в табличному вигляді, використовувати формули, будувати графіки або експортувати дані у зручні формати для подальшого аналізу. Подібна інтеграція робить Google Forms зручним

інструментом для швидкого збору базових даних без необхідності застосування зовнішніх систем.

Водночас сервіс має низку суттєвих обмежень. Можливості аналітики залишаються мінімальними: доступні лише прості кругові діаграми, гістограми та підрахунок відповідей. Налаштування дизайну також досить скромні — користувач може змінити кілька кольорів та вибрати стандартну тему, але не може гнучко керувати версткою або зовнішнім виглядом форми. Для багатьох професійних досліджень цього рівня кастомізації недостатньо. Експерти також зазначають, що Google Forms не має вбудованих функцій для експорту даних у спеціалізовані статистичні формати (наприклад, SPSS), що ускладнює обробку результатів у масштабних наукових дослідженнях [1]. Також Google Forms не підтримує складні сценарії логіки, багаторівневі умови переходів чи персоналізовані опитування. Це обмежує його використання у великих проєктах, які потребують точного налаштування потоку питань або адаптивних сценаріїв.

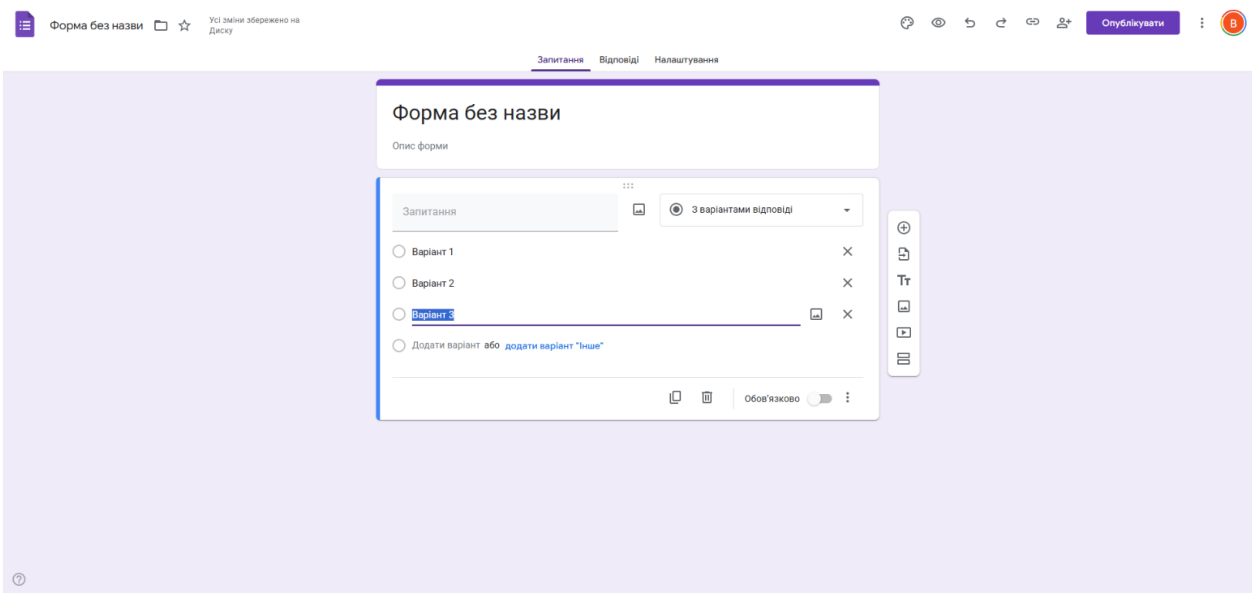

Рисунок 2.1 – Інтерфейс створення опитування у Google Forms.

SurveyMonkey орієнтований на професійні та комерційні опитування і широко використовується у бізнес-аналітиці, HR, маркетингових дослідженнях та корпоративних опитуваннях. Платформа пропонує великий набір типів запитань, готових шаблонів і гнучких інструментів для побудови структури анкети. Окрім

стандартних варіантів відповідей, сервіс підтримує матричні запитання, ранжування, оцінювання, логічні переходи між блоками та персоналізовані сценарії опитування. Однією з ключових особливостей SurveyMonkey є його аналітичний модуль. Користувач може здійснювати сегментацію результатів, застосовувати фільтри, порівнювати вибірки між собою та експортувати дані у різні формати. Ці можливості роблять сервіс зручним для комплексних досліджень, у яких важливо не лише зібрати інформацію, а й отримати глибокий статистичний аналіз. Разом із цим більшість професійних функцій, включаючи розширену аналітику, логіку умов, брендовані теми та обмеження доступу, доступні лише у платних тарифних планах. Зокрема, базовий безкоштовний тариф дозволяє створити опитування лише з 10 запитань та переглянути обмежену кількість відповідей (до 25 на анкету), при цьому функції експорту даних у зручні формати (PDF, Excel) залишаються заблокованими [2]. Це може бути суттєвим обмеженням для невеликих команд, студентських робіт або організацій з обмеженим бюджетом, що підтверджують і незалежні огляди профільних видань [1].

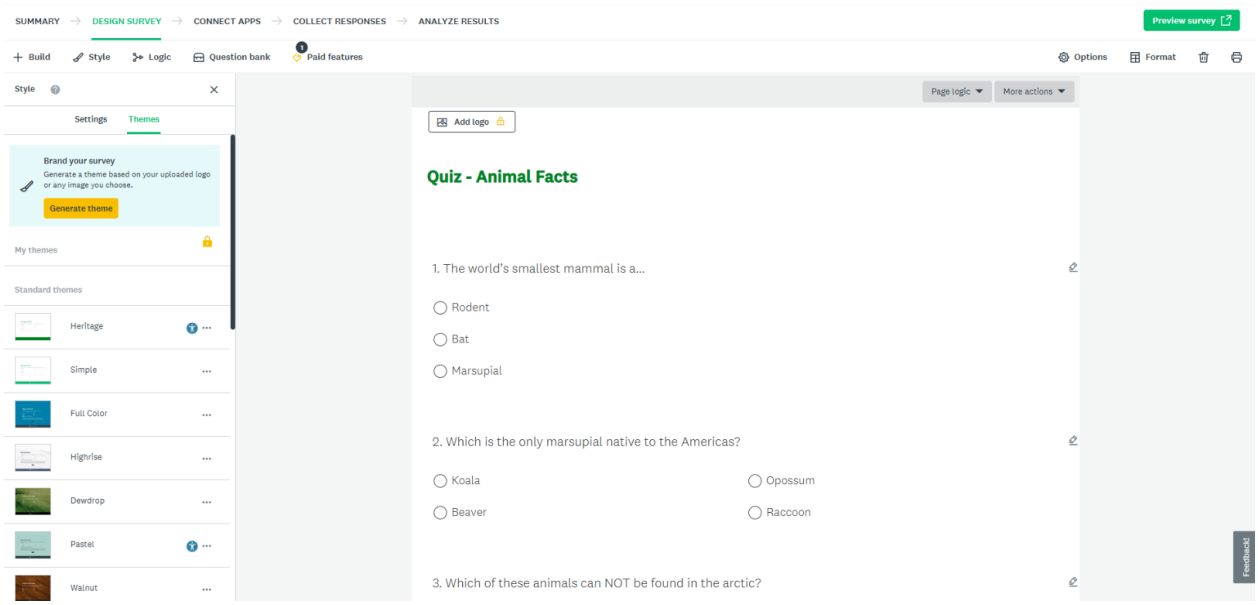

Рисунок 2.2 – Інтерфейс конструктора опитувань у SurveyMonkey

Typeform вирізняється сучасним підходом до взаємодії з користувачем, орієнтованим на зручність та емоційне сприйняття. Основна ідея сервісу полягає у поданні запитань у форматі послідовного діалогу, коли респондент бачить одне

запитання на екрані та переходить до наступного лише після відповіді. Такий підхід створює ефект природної розмови, зменшує когнітивне навантаження та сприяє більш плавному проходженню анкети. Завдяки цьому Typeform часто використовують у дослідженнях, де важлива залученість користувача та якість отриманих даних [3]. Водночас сервіс поступається професійним платформам у гнучкості налаштування логіки опитування та глибині аналітичних інструментів, тому найчастіше застосовується у невеликих маркетингових, освітніх чи дослідницьких проєктах.

LimeSurvey є відкритою open-source платформою, яка дає змогу створювати складні опитування з гнучкою логікою переходів, налаштовуваними шаблонами та розширеними параметрами експорту даних. Завдяки відкритому коду систему можна вільно розгортати на власному сервері, що забезпечує повний контроль над зберіганням даних та їх конфіденційністю, а також дозволяє адаптувати функціональність під специфічні потреби організації чи дослідницького проєкту. Платформа підтримує великий набір типів запитань, багаторівневі сценарії логічних умов, динамічні текстові елементи та можливість створення складних анкет з багатьма секціями. Користувач може самостійно налаштовувати зовнішній вигляд форми, використовуючи шаблони або власні CSS-стилі, а також підключати додаткові модулі та плагіни. Завдяки цьому LimeSurvey часто застосовується у наукових дослідженнях, де потрібна гнучкість та точний контроль над процесом опитування. Разом із перевагами система має й певні недоліки. Найсуттєвішим із них є складніший поріг входу: інтерфейс менш інтуїтивний порівняно з комерційними платформами, такими як Google Forms або Typeform. Налаштування сервера, управління базою даних та технічне адміністрування також вимагають додаткових навичок. Через це LimeSurvey краще підходить для організацій та команд, які мають технічні ресурси для підтримки системи.

2.2. Переваги та недоліки розглянутих систем

Аналіз популярних сервісів для створення онлайн-опитувань показує, що кожна платформа має власні сильні сторони та обмеження, зумовлені її цільовою

аудиторією та архітектурними рішеннями. Google Forms, SurveyMonkey, Typeform та LimeSurvey забезпечують різний рівень функціональності, безпеки, кастомізації та можливостей аналітики, що впливає на їх придатність для конкретних завдань.

З точки зору функціональності Google Forms пропонує простий інтерфейс і базовий набір типів запитань, що робить сервіс зручним для швидкого створення анкет. Проте можливості налаштування форми, логіки переходів і розширеної аналітики є досить обмеженими. Безпека залежить від інфраструктури Google, проте користувач не має контролю над розгортанням системи чи зберіганням даних.

SurveyMonkey демонструє значно ширший функціонал, включаючи логічні оператори, сегментацію відповідей, A/B-тестування та інструменти для корпоративного використання. Однак більшість цих можливостей доступні лише у платних планах. Платформа забезпечує високий рівень захисту даних, але залишає користувача залежним від стороннього сервісу та його тарифів.

Typeform відрізняється інноваційним користувацьким досвідом, орієнтованим на покрокову взаємодію з респондентом. Він забезпечує високий рівень залученості, але поступається SurveyMonkey у гнучкості логіки та глибокій аналітиці. Продукт добре підходить для простих опитувань і маркетингових форм, однак обмежений у масштабуванні великих дослідницьких проєктів.

LimeSurvey вирізняється своєю відкритою архітектурою: сервіс можна розгорнути на власному сервері, забезпечивши повний контроль над даними, чого не пропонують інші платформи. Це робить систему придатною для використання у закладах з підвищеними вимогами до конфіденційності. Водночас інтерфейс менш інтуїтивний, а налаштування потребує технічної підготовки.

Зведений аналіз узагальнено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз сервісів для створення опитувань

Характеристика	Google Forms	SurveyMonkey	Typeform	LimeSurvey
Призначення	Базові опитування, освіта	Професійні опитування для бізнесу	Маркетинг	Наукові дослідження
Простота використання	Дуже проста	Середня, потребує адаптації	Дуже проста, інтуїтивна	Низька, висока складність для новачків
Типи запитань	Базові	Дуже різноманітні	Середні	Розширені та спеціалізовані
Можливості аналітики	Базові графіки	Професійні звіти, сегментація	Базові	Розширені, можливість експорту

Продовження таблиці 2.1

Безпека	Висока, але без власного контролю	Висока (корпоративні стандарти)	Висока	Залежить від сервера (повний контроль)
Вартість	Безкоштовно	Платні тарифи	Безкоштовний обмежений план та платні функції	Безкоштовно

Аналіз даних, наведених у таблиці 2.1, показує, що жоден із розглянутих сервісів не є універсальним рішенням для всіх типів опитувань. Google Forms вирізняється простотою використання та повною безкоштовністю, проте має обмежені можливості кастомізації й аналітики. SurveyMonkey пропонує найбільш професійний набір інструментів, включаючи гнучку логіку та розширені аналітичні функції, однак більшість із них доступні лише у платних тарифах. Typeform забезпечує сучасний, інтуїтивний та залучаючий користувацький досвід, але поступається конкурентам у глибині налаштувань і масштабованості. LimeSurvey, як open-source рішення, надає максимальний контроль над даними та найширшу гнучкість у створенні складних опитувань, однак вимагає технічних навичок і є менш інтуїтивним для користувачів без досвіду.

Отже, основні недоліки існуючих платформ полягають у поєднанні таких факторів, як обмежена аналітика у безкоштовних сервісах, залежність від платних підписок у професійних інструментів, а також складність розгортання та адміністрування open-source рішень. Це створює передумови для розробки нового веб-сервісу, який би об’єднав простоту використання, достатню гнучкість логіки, базову аналітику та доступність без суттєвих фінансових чи технічних бар’єрів.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура веб-додатків

Архітектура веб-додатків визначає логічну організацію компонентів системи, способи їх взаємодії та розподіл відповідальності між окремими частинами програмного забезпечення. Від коректності обраної архітектури залежить стабільність роботи застосунку, його здатність обробляти значні обсяги даних, адаптуватися до збільшення кількості користувачів, а також можливість подальшого розвитку та інтеграції нових функціональних модулів. У сучасних інформаційних системах, що працюють у мережі інтернет, найпоширенішим підходом є клієнт–серверна модель із застосуванням RESTful API, яка забезпечує стандартизовану взаємодію між компонентами через мережеві протоколи. Додатково в таких системах широко використовується багаторівнева структурна організація (Layered Architecture), що дає змогу розділити програмний код за функціональними рівнями та досягти високої модульності. Комбінація цих підходів дозволяє розробляти програмні системи, здатні гнучко реагувати на зміни вимог, розширювати функціональність без суттєвого втручання у наявний код та забезпечувати ефективне управління даними. Для веб-сервісів, орієнтованих на роботу з великим обсягом інформації, наприклад систем опитувань, така архітектура створює умови для швидкого оброблення запитів, підтримки багатокористувацької роботи та гарантованої цілісності даних. Саме тому описані архітектурні підходи стали стандартом у розробці сучасних веб-додатків різних масштабів — від невеликих SaaS-рішень до великих корпоративних систем.

Клієнт–серверна модель передбачає розподіл системи на дві автономні частини: клієнтську та серверну. Такий поділ дозволяє забезпечити логічне й функціональне розмежування між процесами, що відповідають за представлення даних користувачу, та компонентами, які займаються їхньою обробкою. Клієнтська частина виконується у веб-браузері та відповідає за відображення інформації, формування інтерфейсу й забезпечення інтерактивної взаємодії з користувачем у режимі реального часу. Вона отримує дані із сервера у вигляді структурованих

відповідей і на їх основі формує вигляд сторінок, переходи між елементами та виконання користувацьких дій. Відповідно до визначення, наведеного у книзі Кулуріса (G. Coulouris) «Distributed Systems: Concepts and Design» [4], клієнт–серверна архітектура є фундаментальною моделлю для побудови розподілених застосунків, оскільки забезпечує незалежний розвиток клієнтської та серверної частин, підтримує масштабованість та дозволяє ефективно розподіляти навантаження між вузлами системи. Завдяки такій структурі веб-додатки зберігають високу продуктивність і стабільність навіть за умов зростання кількості користувачів або обсягу даних.

У контексті веб-сервісів, що надають можливість створювати та проходити опитування, клієнтська частина відіграє ключову роль, оскільки забезпечує зручність користувацької роботи з формами, питаннями та результатами. Використання фреймворків на основі JavaScript або TypeScript дозволяє реалізувати гнучкі й інтерактивні UI-компоненти, які швидко реагують на дії користувача та забезпечують комфортну роботу з опитуваннями будь-якої складності.

Серверна частина виконується на стороні backend та є центральним елементом системи, який відповідає за опрацювання запитів від клієнта, реалізацію бізнес-логіки, валідацію даних, автентифікацію та управління всіма ресурсами веб-додатка. Сервер приймає, аналізує та обробляє вхідні дані, виконує роботу з базою даних та формує відповіді у стандартизованому форматі. У веб-сервісах, пов’язаних із опитуваннями, сервер відповідає за створення структур опитувань, збереження відповідей респондентів, керування доступом користувачів та формування аналітичних даних.

Взаємодія між клієнтом і сервером здійснюється за допомогою HTTP/HTTPS протоколів у форматі JSON, що дає можливість забезпечити легкість інтеграції, порівняльну простоту обробки даних та високу сумісність із різними клієнтськими платформами. Дотримання архітектурних принципів REST дозволяє забезпечити стандартизовані правила обміну інформацією та спрощує масштабування системи, особливо коли вона працює з великою кількістю одночасних запитів. Такий підхід є

ефективним і надійним для реалізації веб-платформ, що обробляють значні масиви структурованих даних, включаючи опитування, відповіді та статистичні результати.

REST (Representational State Transfer) є домінуючим архітектурним стилем побудови веб-сервісів, який ґрунтується на моделі взаємодії клієнта і сервера та принципах, притаманних розподіленим системам. Як зазначає Кулуріс (G. Coulouris) у своїй праці [4], архітектури розподілених застосунків базуються на чіткому визначенні ресурсів, їх ідентифікації та stateless-комунікації між вузлами — саме ці принципи становлять основу REST. Популярність REST зумовлена його простотою, високою сумісністю з різними платформами, легкістю інтеграції сторонніх застосунків та можливістю ефективної обробки великої кількості запитів. Завдяки відсутності прив’язки до конкретних технологій REST є універсальним способом організації API для веб-додатків різного масштабу — від невеликих сервісів до складних розподілених систем.

У контексті веб-сервісів, що працюють із динамічними даними, такими як опитування, питання та відповіді користувачів, RESTful API дозволяє організувати логічно структурований і передбачуваний спосіб доступу до ресурсів. Кожен об’єкт (опитування, запитання, відповідь, профіль користувача) розглядається як окремий ресурс, який має власний унікальний ідентифікатор та адресу в системі. Це спрощує розробку, тестування та розширення логіки застосунку, а також забезпечує зрозумілість API для інших розробників.

Основні принципи REST, реалізовані у веб-додатках, включають:

Унікальність URL-адрес ресурсів. Кожен тип даних має власний endpoint, наприклад: /api/surveys — робота з опитуваннями, /api/auth/login — автентифікація користувачів. Це забезпечує логічну структуру й передбачувану навігацію API.

Використання стандартних HTTP-методів. Для виконання операцій над ресурсами застосовуються уніфіковані методи протоколу HTTP, опис яких наведено у таблиці 3.1.

Таблиця 3.1 – Стандартні HTTP-методи та їх призначення в REST-архітектурі

Метод	Призначення (операція)	Опис дії
GET	Отримання даних (Read)	Запит на отримання ресурсу або колекції ресурсів без їх зміни.
POST	Створення ресурсу (Create)	Відправлення даних для створення нового підпорядкованого ресурсу.
PUT / PATCH	Оновлення даних (Update)	PUT замінює ресурс повністю, PATCH використовується для часткових змін.
DELETE	Видалення (Delete)	Запит на видалення зазначеного ресурсу з сервера.

Завдяки використанню цих методів API стає інтуїтивно зрозумілим і легко інтегрується з будь-яким клієнтським застосунком.

- Передача даних у форматі JSON. JSON є компактним, людиночитабельним і широко підтримується усіма сучасними мовами програмування, що робить його оптимальним форматом обміну даними у веб-середовищі.
- Stateless-взаємодія. Кожен запит містить всю інформацію, необхідну для його обробки. Сервер не зберігає стану між запитами, що значно спрощує масштабування — нові серверні інстанси можуть підключатися без синхронізації попереднього контексту.
- Кешування відповідей. REST рекомендує використовувати кешування для підвищення швидкодії, особливо для ресурсів, що запитуються багаторазово або рідко змінюються.
- Розділення клієнта та сервера. Це дозволяє оновлювати інтерфейс без зміни серверної логіки, а сервер — без впливу на роботу клієнтської частини.

Таке структурування відповідає рекомендаціям, наведеним у Massé [5], і забезпечує передбачуваність та масштабованість RESTful API.

REST-архітектура є оптимальним рішенням для системи опитувань, оскільки вона забезпечує можливість легко додавати нові функціональні модулі — наприклад, аналітичні інструменти, механізми AI-аналізу, експорту даних або розширення типів питань. Крім того, REST забезпечує високу продуктивність навіть за великої кількості паралельних запитів, що є важливим для веб-сервісів, де одночасно працюють створювачі опитувань і респонденти. Завдяки стандартизованій структурі API система залишається гнучкою, розширюваною та придатною для інтеграцій з іншими платформами та мобільними застосунками.

Для організації внутрішньої логіки backend у веб-додатках широко застосовується багаторівнева архітектура (Layered Architecture), яка передбачає суворе розмежування відповідальностей між різними частинами системи. Такий підхід дозволяє чітко розподілити функції, спростити підтримку програмного забезпечення, підвищити його гнучкість та забезпечити незалежний розвиток окремих компонентів. Як зазначає Coulouris та ін. [4], поділ системи на взаємопов’язані, але автономні рівні є одним з ключових принципів побудови розподілених програмних систем, оскільки це підвищує передбачуваність та структурованість архітектури. У типовому веб-додатку на основі Layered Architecture виділяють такі основні рівні:

Presentation Layer — рівень представлення, який включає маршрути та контролери. Вони приймають запити клієнта, виконують первинну валідацію даних, координують подальшу обробку запиту та повертають відповідь у стандартизованому форматі. Контролери не повинні містити складної логіки, оскільки їхня основна роль — виклик відповідних сервісів та формування коректної відповіді.

Business Logic Layer — рівень бізнес-логіки, який реалізує алгоритми роботи системи. Саме тут опрацьовуються правила побудови опитувань, перевірка прав

доступу, специфічні сценарії взаємодії респондентів з формою, логіка підрахунку статистики, інтеграція зі сторонніми сервісами або інструментами аналізу даних. Розміщення такої логіки в окремому шарі забезпечує чіткість структури та дозволяє масштабувати систему без зміни її основних модулів.

Data Access Layer — рівень доступу до даних, який відповідає за взаємодію з базою даних. На цьому рівні використовуються моделі та репозиторії, що визначають структуру таблиць або документів і забезпечують виконання CRUD-операцій. Розділення доступу до даних у межах окремого шару спрощує зміну або оптимізацію схеми бази даних без втручання у бізнес-логіку чи логіку контролерів.

Використання такого підходу на практиці забезпечує важливі переваги. По-перше, досягається висока модульність системи, оскільки логіка, пов’язана з різними аспектами роботи застосунку, чітко структурована за рівнями. По-друге, підвищується можливість повторного використання коду. По-третє, спрощується тестування: кожен шар можна перевіряти окремо, що підвищує якість і надійність програмного рішення. По-четверте, система стає більш гнучкою до змін, оскільки оновлення одного рівня не потребує модифікацій у інших.

Паралельно з багаторівневою архітектурою в веб-додатках також використовується патерн MVC (Model–View–Controller). У серверних REST-сервісах його реалізація має свої особливості.

- Model відповідає за структуру та зберігання даних, використовуючи схеми та об’єкти доступу до бази;
- View у класичних веб-додатках формує HTML-відповідь, однак у REST-орієнтованих системах його роль виконує JSON-представлення, яке надсилається клієнту;
- Controller обробляє HTTP-запити, викликає відповідні сервіси бізнес-логіки та формує стандартизовану відповідь.

Як підкреслює Coulouris та співавтори [4], поділ системи на рівні обробки даних та логіки взаємодії є ключовим принципом побудови розподілених застосунків, що забезпечує стійкість і прогнозованість їх поведінки.

MVC органічно вписується в Layered Architecture і доповнює її, створюючи чіткий шаблон організації серверної логіки. Використання клієнт-серверної структури, REST-підходу та багаторівневої архітектури дозволяє сформувати продуктивний і масштабований веб-сервіс. Такий підхід є оптимальним для систем опитувань, у яких важливо забезпечити стабільну роботу API, гнучкість побудови опитувань та можливість безперервного розширення функціональності, включно з інтеграцією інтелектуальних модулів, аналітичних механізмів, адаптивної логіки та засобів автоматичної обробки відповідей.

3.2. Основи систем автентифікації

Автентифікація є одним із ключових елементів безпеки будь-якого веб-додатку, оскільки саме вона визначає, чи має користувач право доступу до певних ресурсів і функцій системи. У процесі автентифікації відбувається перевірка достовірності наданих користувачем облікових даних, після чого сервер приймає рішення про надання доступу. Це робить механізм автентифікації фундаментально важливим для захисту конфіденційної інформації, забезпечення цілісності даних та запобігання несанкціонованим діям у межах системи. З огляду на те, що переважна більшість сучасних веб-додатків працює в мережі інтернет і доступна широкому колу користувачів, автентифікація має враховувати ризики зовнішніх атак, зловмисних спроб проникнення та перехоплення даних.

У розподілених веб-системах, які обслуговують велике навантаження або складаються з кількох серверних компонентів, вимоги до механізмів автентифікації значно зростають. Вона має бути не лише безпечною, але й масштабованою, гнучкою до розширення функціоналу, стійкою до поширених кіберзагроз та придатною для інтеграції з мобільними, десктопними чи сторонніми клієнтськими застосунками. В умовах, коли системи можуть працювати у хмарному середовищі або використовувати балансування навантаження між декількома серверами,

класичні методи автентифікації, які передбачають збереження стану сесії на сервері, втрачають актуальність.

Саме тому у веб-сервісах, побудованих за клієнт–серверною архітектурою та RESTful-підходом, ключовим рішенням є використання маркерів (токенів), які передаються між клієнтом і сервером і підтверджують особу користувача під час кожного запиту. Такий підхід забезпечує незалежність сервера від конкретної сесії, а клієнта — від конкретного серверного вузла. Токени не потребують збереження на стороні сервера, що дозволяє створювати високонавантажені системи з горизонтальним масштабуванням.

Найбільш поширеним форматом токенів є JSON Web Token (JWT) — стандарт відкритого формату, який забезпечує компактність, безпеку та універсальність передачі інформації про користувача. JWT вже багато років є де-факто стандартом у розробці RESTful API завдяки своїм властивостям: він легко передається через HTTP-заголовки, може бути збережений у cookies або локальному сховищі клієнта, містить підпис для перевірки цілісності та може бути перевірений на будь-якому сервері, що має ключ підпису. Це дозволяє будувати stateless-системи, в яких кожен запит самодостатній, а обробка автентифікації не залежить від стану попередньої взаємодії.

Таким чином, автентифікація на основі JWT забезпечує не лише високий рівень безпеки, але й гнучкість при розгортанні веб-додатку, можливість легкої інтеграції зі сторонніми сервісами, підтримку авторизації на різних клієнтських платформах та оптимальні умови для масштабування системи опитувань, у тому числі в хмарному середовищі.

JSON Web Token (JWT) є сучасним стандартом передачі даних про автентифікованого користувача у розподілених веб-системах. Він являє собою компактний, URL-безпечний текстовий маркер, який кодує інформацію у структурованому вигляді та може бути перевірений сервером без необхідності зберігання стану. Згідно зі стандартом IETF [6], JWT складається з трьох частин —

заголовка, корисного навантаження та криптографічного підпису, що забезпечує цілісність і автентичність токена. Завдяки своїй універсальності та простоті інтеграції, JWT широко застосовується в RESTful API, мобільних застосунках, односторінкових інтерфейсах (SPA) та системах з мікросервісною архітектурою.

JWT складається з трьох частин, які розділені крапками і кодуються у форматі Base64Url. Перший компонент — це заголовок (Header), де міститься інформація про тип токена та алгоритм, за допомогою якого він буде підписаний. Зазвичай використовується HMAC SHA256 (HS256), однак у складніших системах можуть застосовуватись асиметричні алгоритми типу RS256, що передбачають наявність пари ключів — приватного і публічного. Заголовок визначає спосіб підпису, структуру та правила обробки токена на сервері.

Другий компонент — корисне навантаження (Payload), яке містить набір тверджень про користувача та параметри дії токена. Саме тут розміщується ідентифікатор користувача, його роль або рівень доступу, час створення токена, строк його життя, а також інші довільні дані, які можуть знадобитися для логіки системи. Існує два типи тверджень: стандартні (наприклад, exp для визначення часу закінчення дії, iat для часу видачі, iss для вказання джерела токена) та кастомні, визначені розробником. Payload не шифрується за замовчуванням, але він захищений підписом, що унеможливлює його непомітну модифікацію.

Третій компонент — це криптографічний підпис (Signature). Він генерується шляхом об’єднання закодованих Header і Payload та застосування алгоритму підписання з використанням секретного ключа. Підпис гарантує цілісність токена: будь-яка змінена або підроблена інформація призведе до того, що сервер не зможе пройти перевірку підпису, і токен буде відхилено. Таким чином, навіть якщо зловмисник зможе прочитати вміст токена, він не зможе змінити його без володіння секретним ключем.

Однією з ключових переваг JWT є його самодостатність. Токен містить усю необхідну інформацію про користувача і не потребує зберігання стану на сервері,

що повністю відповідає принципу stateless-сервісів у REST-архітектурі. Як зазначено у стандарті IETF JSON Web Token (JWT) [6], структура токена дозволяє передавати корисні дані у компактному форматі й забезпечує можливість криптографічної перевірки на будь-якому серверному вузлі без використання централізованих сесій або спільної пам’яті. Подібний підхід ідеально підходить для систем, які масштабуються горизонтально: запити можуть оброблятися будь-яким сервером у кластері, не потребуючи синхронізації стану між інстансами.

JWT також зручний для передачі по мережі: він має компактний розмір, сумісний з URL, HTTP-заголовками та cookie-механізмом. Це робить його універсальним та придатним як для браузерних клієнтів, так і для мобільних застосунків. У комплексі ці властивості роблять JWT практичним, ефективним і безпечним рішенням для автентифікації у веб-сервісах, особливо тих, які потребують високої швидкодії та здатності обробляти велику кількість одночасних користувачів.Одним із найбільш надійних і сучасних методів організації автентифікації у веб-додатках є так званий dual-token підхід, який базується на використанні двох взаємодоповнюючих токенів — короткотривалого Access Token та довготривалого Refresh Token. Така схема дозволяє досягти балансу між безпекою, зручністю використання та продуктивністю системи, що особливо важливо у веб-сервісах, які працюють у розподіленому або хмарному середовищі.

Access Token відіграє роль основного маркера доступу та використовується для підтвердження прав користувача під час виконання запитів до захищених ресурсів. Його строк дії обмежується кількома хвилинами, що суттєво зменшує ризики у разі його викрадення або перехоплення. Токен містить лише ключову інформацію, необхідну для швидкої авторизації — ідентифікатор користувача, роль, набір дозволів та метадані. Завдяки своїй компактності Access Token легко передається в HTTP-запитах і практично не створює навантаження на сервер.

Refresh Token, на відміну від Access Token, має значно більший строк життя і не використовується для безпосереднього доступу до ресурсів. Його єдина функція — отримання нового Access Token після завершення строку його дії. Це дозволяє

уникнути повторного введення логіну та пароля, забезпечуючи так звану «довгу сесію». З міркувань безпеки Refresh Token зазвичай зберігається у httpOnly cookie, що унеможливлює доступ до нього через JavaScript і значно знижує ризик крадіжки. Якщо існує підозра на компрометацію, Refresh Token може бути відкликаний сервером — наприклад, через список недійсних токенів або зберігання активних сесій у базі даних.

Особливість такого підходу полягає у відмові сервера від зберігання стану сесії. Як зазначають українські дослідники: «Токен-орієнтовані механізми автентифікації забезпечують високу безпеку, оскільки дозволяють перевіряти користувача без необхідності зберігання стану сесії на сервері» [6]. Це робить dual-token механізм особливо ефективним у системах, які працюють у хмарному середовищі або використовують балансування навантаження.

Dual-token механізм має низку значущих переваг. По-перше, він забезпечує високий рівень безпеки: навіть якщо Access Token потрапить у руки зловмисника, його використання буде обмежене у часі. По-друге, цей підхід створює комфорт для користувачів, оскільки дозволяє залишатися у системі тривалий час без повторної автентифікації. По-третє, механізм легко підтримує завершення всіх активних сесій, що є важливим для корпоративних систем або у випадку втрати пристрою. По-четверте, dual-token модель повністю сумісна з безстанною архітектурою REST і дозволяє масштабувати систему без потреби у централізованому зберіганні сесій. Для кращого розуміння технічних відмінностей між типами токенів, їх порівняльна характеристика наведена у таблиці 3.2.

Таблиця 3.2 – Поріняльна таблиця Access та Refresh токенів

Характеристика	Access Token (Токен доступу)	Refresh Token (Токен оновлення)
Призначення	Доступ до захищених ресурсів API	Отримання нової пари токенів

Продовження таблиці 3.2

Термін дії	Короткий (15–30 хвилин)	Довгий (7–30 днів)
Місце зберігання	Пам’ять клієнта (RAM) або JSON-відповідь	HttpOnly Cookie (захищене сховище)
Формат	JWT (містить payload з даними)	JWT або випадковий рядок (Opaque)
Безпека	Вразливий до XSS (якщо не в Cookie)	Захищений від XSS, вразливий до CSRF (потребує захисту)
Дії при витоку	Зловмисник має доступ, доки не спливе час	Може бути відкликаний сервером (Revoked)

Ефективність і надійність токен-орієнтованих моделей підтверджується і в сучасних наукових дослідженнях. Зокрема, українські дослідники наголошують, що подібні механізми дають змогу перевіряти користувача без збереження стану сесії на сервері, що підвищує безпеку та спрощує масштабування веб-застосунків [6]. Саме тому dual-token підхід активно використовується у великій кількості сучасних цифрових платформ, таких як Google, Facebook, GitHub, Microsoft Azure та інші сервіси, які працюють у багатокомпонентних розподілених інфраструктурах. Така модель зарекомендувала себе як один із найбільш ефективних способів забезпечення безпеки, стійкості та стабільності автентифікації у веб-системах.

Оскільки dual-token механізм має чітко визначений життєвий цикл, доцільно подати його у вигляді блок-схеми, яка наочно демонструє взаємодію клієнта та серверної частини, процес оновлення токенів та логіку обробки помилок. На блок-схемі 3.1 представлено узагальнену модель роботи dual-token автентифікації у веб-додатку.

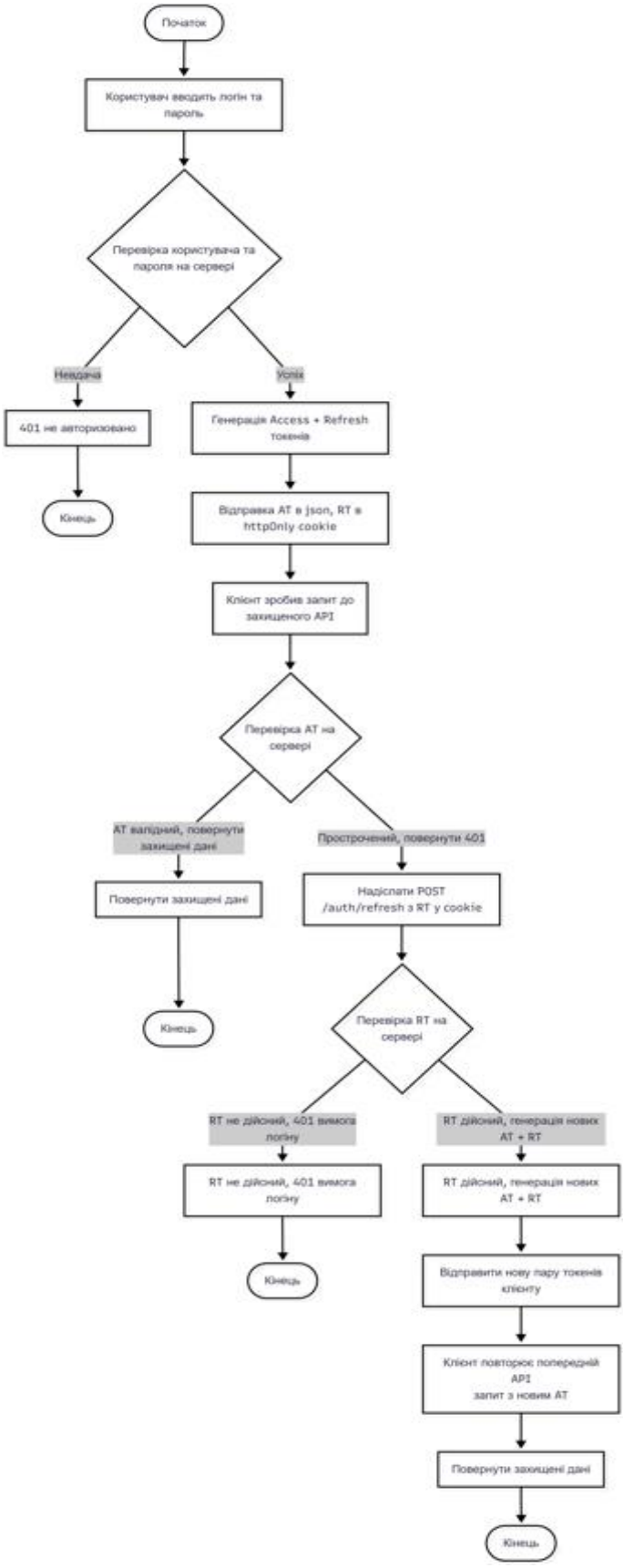

Рисунок 3.1 – Блок-схема процесу dual-token автентифікації

Безпека системи автентифікації значною мірою залежить від правильного застосування криптографічних методів, які забезпечують цілісність, конфіденційність та автентичність даних, що передаються між клієнтом і сервером. У сучасних веб-додатках криптографія відіграє фундаментальну роль, оскільки саме вона захищає критично важливу інформацію, таку як облікові дані користувачів, маркери доступу, персональні дані та інші конфіденційні відомості. Вбудовані криптографічні механізми дозволяють створювати системи, стійкі до найпоширеніших атак — перехоплення трафіку, підміни даних, підбору паролів та маніпуляцій з токенами [8].

Одним з основних напрямів криптографічного захисту є хешування паролів. Під час реєстрації або зміни пароля користувача сервер не зберігає пароль у відкритому вигляді. Натомість застосовується односпрямоване криптографічне хешування із використанням алгоритму bcrypt, який генерує стійкий хеш і включає в себе механізм адаптивної складності. Це означає, що зі збільшенням обчислювальних можливостей сучасних систем значення параметрів bcrypt можна підвищувати, ускладнюючи атаку brute-force. Додатково використовується salt — випадкове криптографічне значення, яке додається до пароля перед хешуванням. Впровадження salt унеможливлює використання попередньо згенерованих таблиць (rainbow tables), тим самим суттєво підвищуючи безпеку збережених паролів, що є обов'язковою вимогою сучасних стандартів безпеки, таких як NIST [9].

Важливою складовою криптографічного захисту є підпис токенів, що використовуються у системі автентифікації. JWT містить криптографічний підпис, який гарантує, що вміст токена не був змінений після його створення. У найпоширеніших системах застосовується алгоритм HMAC SHA256 — симетричний метод підпису, де один секретний ключ використовується як для створення, так і для перевірки підпису. У більш складних або розподілених системах перевагу можуть надавати асиметричним алгоритмам, таким як RS256 (RSA), де підпис створюється приватним ключем, що зберігається на сервері, а його перевірка може виконуватися за допомогою публічного ключа. Такий підхід

дозволяє масштабувати систему та забезпечує можливість верифікації токенів великим набором незалежних сервісів без розкриття секретної інформації [10].

Окремим важливим напрямом є шифрування передачі даних між клієнтом і сервером. Для цього використовується протокол HTTPS, який запобігає перехопленню даних зловмисником завдяки створенню захищеного каналу на основі TLS (Transport Layer Security). Сучасні версії TLS 1.2 та TLS 1.3 застосовують стійкі криптографічні алгоритми, що забезпечують неможливість дешифрування трафіку без володіння приватним ключем сервера. Шифрування каналу є обов'язковим елементом безпеки, оскільки навіть найнадійніша логіка автентифікації може бути скомпрометована через атаки типу «людина посередині» (Man-in-the-Middle), якщо дані передаються у відкритому вигляді.

Ще одним рівнем захисту є використання web-безпечних cookie, які зберігають токени або інші критичні дані. Встановлення атрибутів HttpOnly, Secure та SameSite дозволяє мінімізувати ризики атак на клієнтському боці. HttpOnly забороняє доступ JavaScript до cookie, тим самим захищаючи їх від XSS-атак; Secure гарантує передачу cookie лише за зашифрованим HTTPS-з'єднанням; режим SameSite=Strict запобігає автоматичній передачі cookie при міжсайтових запитах, що суттєво знижує ризик CSRF-атак.

Поєднання цих криптографічних методів формує комплексний механізм захисту, який робить систему автентифікації стійкою до значної частини сучасних кіберзагроз і дозволяє забезпечити надійний захист даних користувачів у веб-додатку.

3.3. Алгоритм роботи системи опитувань

Алгоритм роботи веб-системи опитувань охоплює повний цикл взаємодії між автором опитування, респондентами та серверною інфраструктурою. Цей процес починається зі створення опитування та завершується формуванням підсумкових аналітичних звітів, що дозволяє авторам отримати якісну й структуровану інформацію для подальшого аналізу. Архітектура системи організована таким

чином, щоб забезпечити узгодженість та цілісність даних, стабільну роботу при збільшенні навантаження, а також оперативне реагування на дії користувачів у режимі реального часу.

Етап створення опитування є одним із ключових у загальному алгоритмі роботи системи. Автор формує структуру майбутнього опитування, задаючи його основні параметри: назву, опис, перелік запитань та формат відповідей. Система дозволяє використовувати різні типи полів — текстові відповіді, числові значення, вибір одного або декількох варіантів, рейтинги чи дати, що робить опитування гнучким та придатним для широкого спектра дослідницьких задач. Важливим аспектом є те, що всі параметри налаштовуються одразу в процесі створення, без необхідності виділяти окремий етап чернетки: автор відразу бачить, як виглядатиме форма, може змінювати порядок запитань, уточнювати логіку переходів та вимоги до обов’язкового заповнення.

Під час створення опитування також встановлюються такі важливі характеристики, як доступність форми для респондентів, дата та час автоматичного закриття, максимальна кількість дозволених відповідей або правила щодо повторного проходження. Ці параметри формують поведінку опитування після публікації та дозволяють автору контролювати його життєвий цикл. Особливу роль відіграє налаштування логічних переходів (умовної логіки), завдяки чому система може адаптувати послідовність запитань відповідно до попередніх відповідей респондента. Це суттєво покращує користувацький досвід та дозволяє формувати більш персоналізовані опитування.

Після завершення налаштування опитування автоматично отримує унікальне публічне посилання. Його можна поширювати через електронну пошту, соціальні мережі або інші канали комунікації. Веб-система одразу робить це опитування доступним для респондентів, без необхідності додаткової модерації або ручної активації. Завдяки цьому створення та запуск опитування відбуваються максимально швидко і не потребують від автора спеціальних технічних знань.

Під час проходження опитування респондент взаємодіє з інтерфейсом форми, яка послідовно відображає запитання та забезпечує коректність введення даних. Інтерфейс побудований таким чином, щоб зробити процес максимально зрозумілим: респондент бачить прогрес виконання, може повернутися до попередніх запитань (якщо це дозволено структурою опитування), а також отримує візуальні підказки щодо обов'язкових полів або формату введення. Кожне запитання обробляється окремо, що дозволяє системі оперативно реагувати на дії користувача — наприклад, приховувати або показувати наступні запитання залежно від вибраних відповідей, якщо використовується логічна структура переходів.

Перед відправленням результатів клієнтська частина виконує первинну перевірку коректності введених даних. Система контролює, чи не пропущено обов'язкові питання, чи відповідають введені значення допустимим типам (дата, число, текст), чи не порушено встановлених обмежень, таких як мінімальна та максимальна довжина текстових відповідей або діапазон числових значень. Додатково перевіряється, чи дозволене повторне проходження опитування з цього пристрою або для цього користувача, чи не досягнуто встановленого автором обмеження щодо максимальної кількості респондентів, а також чи не сплив термін дії опитування. Лише після успішного проходження всіх перевірок дані формуються в структуру відповіді та надсилаються на сервер.

На серверній стороні відбувається більш глибока перевірка отриманої інформації. Спочатку система переконується, що структура отриманих даних відповідає формату опитування, а всі запитання, що передані, належать до відповідного опитування і не були змінені вручну. Така перевірка захищає систему від підроблених або некоректно сформованих запитів. Далі сервер проводить валідацію значень відповідей згідно з правилами, заданими автором: перевіряє допустимість варіантів, правильність типів даних, відповідність логічним умовам та часовим обмеженням.

Після успішної валідації запис формується у стандартизованому вигляді, фіксується точний час його надсилання, а також метадані — наприклад, IP-адреса чи

інформація про клієнтський пристрій, якщо це дозволено налаштуваннями. Дані зберігаються в базі таким чином, щоб їх можна було швидко читати та аналізувати, не перевантажуючи систему навіть при значному потоці відповідей. Для цього використовуються оптимізовані структури зберігання та індексації, які дозволяють ефективно обробляти великі масиви інформації в режимі реального часу.

Архітектура серверної частини спроєктована так, щоб забезпечити стабільність роботи навіть у випадку різких піків навантаження — наприклад, коли велика кількість респондентів починає проходити опитування одночасно. Завдяки асинхронній моделі обробки запитів система може обслуговувати десятки чи сотні паралельних відповідей без затримок, що є критично важливим для масових або відкритих опитувань із широким колом учасників.

Після завершення активного етапу збору відповідей система переходить до аналітичної частини, яка є ключовою для отримання практичного результату від проведеного опитування. Усі зібрані дані проходять низку внутрішніх процедур обробки, що забезпечують їх узгодженість, структурування та підготовку до подальшого аналізу. На першому етапі система агрегує всі отримані відповіді, об’єднуючи їх у впорядкований масив даних, де кожна відповідь прив’язана до конкретного запитання та часу її надсилання. Це дозволяє коректно працювати з великими обсягами інформації, виконуючи подальші обчислення без втрати продуктивності.

Далі здійснюється статистична обробка отриманих результатів. Для кожного запитання система визначає кількісні показники, що відображають розподіл відповідей: загальну кількість відповідей, частотність вибору варіантів, відсоткове співвідношення, середні значення або діапазони, якщо це передбачено типом питання. Для запитань з вибором кількох варіантів проводиться додаткове групування та підрахунок комбінацій відповідей. У випадку числових даних система обчислює середнє значення, медіану або інші базові статистичні параметри, що дозволяє автору краще зрозуміти характер розподілу. Статистичні результати

подаються у вигляді структурованих таблиць, що забезпечує зручний огляд отриманих даних.

Паралельно відбувається формування графічних елементів звіту, які включають діаграми, гістограми, колові схеми або лінійні графіки залежно від типу питання. Візуалізація дозволяє наочно продемонструвати тенденції та закономірності, які можуть бути менш помітними у табличному вигляді. Саме графічне представлення значно спрощує аналіз для користувачів, які не мають досвіду роботи зі статистикою.

Окрему увагу приділено роботі з відкритими текстовими відповідями. У таких випадках система може проводити базовий синтаксичний аналіз або передавати дані до інтелектуального модуля, який виконує глибинну обробку тексту: визначає ключові концепції, виявляє повторювані теми, інтерпретує настрій респондентів або формує узагальнені висновки. Використання механізмів штучного інтелекту дозволяє значно підвищити інформативність текстових відповідей, перетворюючи їх із несистематизованого набору повідомлень на структурований аналітичний матеріал.

Після завершення аналітичної обробки формується підсумковий звіт, який надає автору опитування можливість комплексно оцінити результати. Звіт може містити статистичні таблиці, графічні елементи, текстові висновки, автоматично згенеровані інсайти та рекомендації. Для зручності подальшого використання звіт може бути експортований у кілька форматів — PDF для друку та офіційної документації та CSV для подальшого аналізу у табличних редакторах.

Таким чином, завершальний етап роботи системи опитувань забезпечує повний і логічно впорядкований аналіз зібраних даних, перетворюючи їх на зрозумілий та структурований аналітичний матеріал. Це дозволяє авторам опитувань швидко отримувати значущу інформацію, приймати обґрунтовані рішення та використовувати результати у дослідницькій, бізнесовій або навчальній діяльності.

3.4 Візуалізація алгоритму функціонування системи

Алгоритм роботи системи опитувань можна наочно представити у вигляді блок-схеми, яка відображає основні етапи взаємодії між автором, респондентом та серверною частиною додатку. На рисунку 3.2 показано повний життєвий цикл опитування — від моменту його створення до автоматичного формування аналітичних даних після завершення збору відповідей.

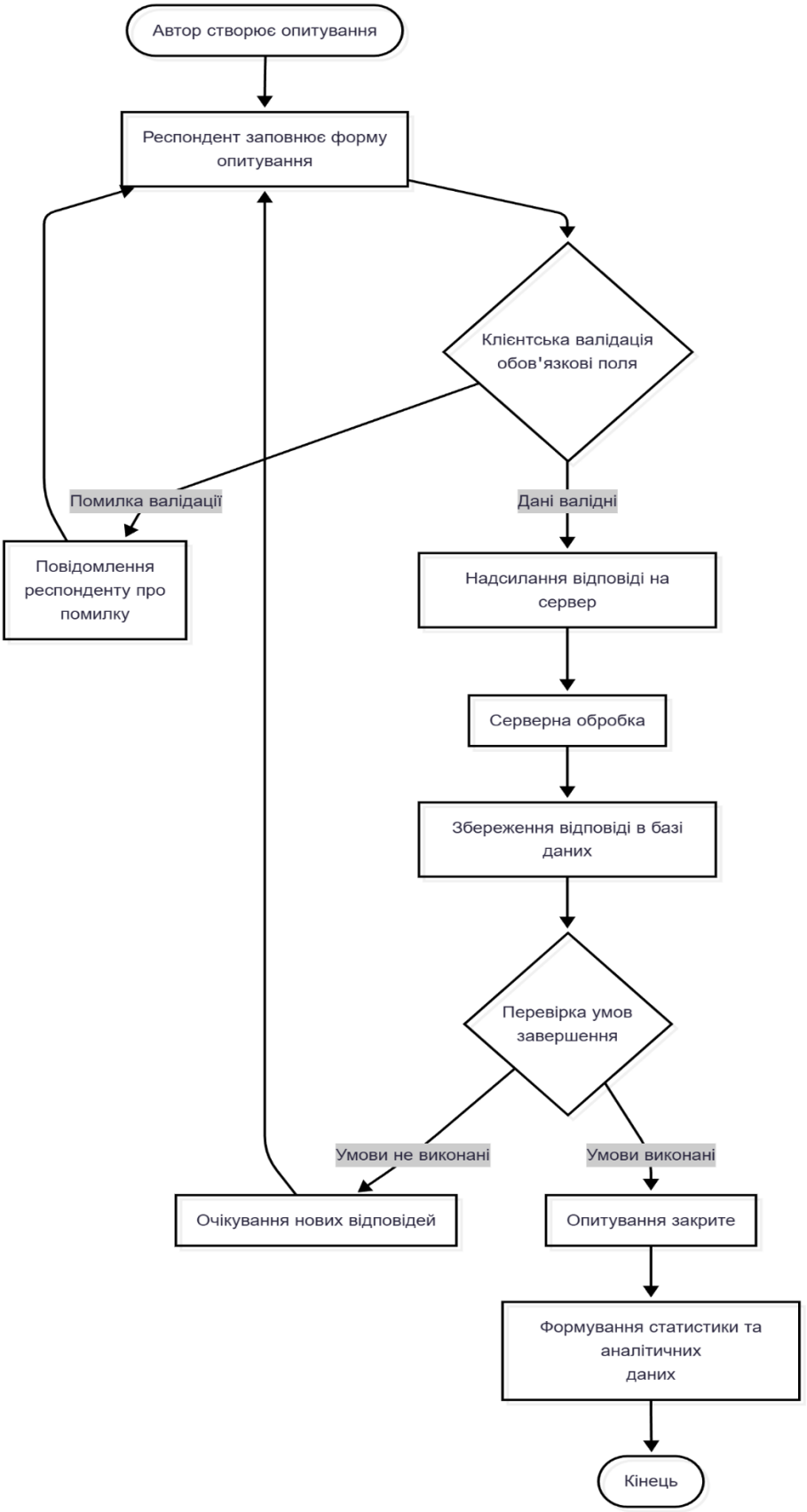

Рисунок 3.2 – Блок-схема життєвого циклу опитування у веб-системі

4.ПРАКТИЧНА ЧАСТИНА

4.1. Опис розробки веб-сервісу

Розробка веб-сервісу опитувань передбачала створення сучасної багатокомпонентної клієнт–серверної системи, здатної забезпечувати стабільну роботу при значному навантаженні, підтримувати гнучку логіку опитувань та інтегрувати інструменти штучного інтелекту для автоматизації аналітичних процесів. Архітектура системи формувалася з урахуванням вимог до масштабованості, продуктивності та можливості подальшого розширення функціональності без необхідності кардинальної перебудови внутрішніх модулів. Саме тому веб-додаток було спроєктовано за принципами модульності, розділення відповідальностей та використання стандартизованих протоколів обміну даними.

Процес розробки охоплював декілька послідовних етапів, кожен з яких мав критичне значення для формування цілісної та надійної системи. На першому етапі здійснювався вибір технологічного стеку, що відповідає сучасним вимогам до веб-розробки, включаючи можливість побудови продуктивного REST API, інтеграції сторонніх сервісів та забезпечення типобезпечності клієнтського коду. Наступним кроком стало проєктування структури бази даних, оптимізованої під зберігання динамічних структур опитувань, запитань різних типів та великої кількості відповідей, що надходять паралельно.

Після визначення архітектурних рішень було реалізовано серверну логіку, яка охоплює обробку HTTP-запитів, виконання бізнес-правил, управління опитуваннями, роботу з токенами автентифікації, обробку відповідей респондентів та формування статистичних даних. Окремий напрямок становила інтеграція модулів штучного інтелекту, яка дала змогу автоматизувати створення опитувань, генерувати варіанти запитань, здійснювати аналіз текстових відповідей та формувати аналітичні рекомендації.

Розробка клієнтського інтерфейсу включала проєктування та реалізацію інтерактивного та адаптивного UI на основі React, створення багаторівневої

навігації, побудову системи компонентів, організацію передачі даних між фронтендом і сервером, а також візуалізацію статистики за допомогою графіків та діаграм. Особлива увага приділялася зручності користувача, мінімізації часу відповіді та створенню інтуїтивних інструментів для управління опитуваннями.

Важливою частиною розробки стала побудова системи безпеки. Це включало механізми автентифікації на основі JWT та Google OAuth, захист каналів обміну даними, налаштування політик CORS, обмеження частоти запитів, перевірку коректності даних на рівні моделей, а також впровадження httpOnly cookies для безпечного зберігання refresh-токенів. Комплекс таких заходів забезпечив відповідність сучасним вимогам до захисту персональних даних та безпеки веб-сервісів.

Узагальнюючи, процес розробки веб-сервісу опитувань поєднував архітектурне планування, інженерні рішення та сучасні технологічні підходи, що в сукупності забезпечили створення гнучкої, масштабованої та функціонально насиченої системи.

Технологічний стек було сформовано відповідно до вимог щодо продуктивності, розширюваності та інтеграції сторонніх сервісів, а також із урахуванням перспектив подальшого розвитку системи. Ключовим елементом серверної логіки є платформа Node.js, що працює на основі подієвої моделі виконання та неблокуючого введення-виведення (non-blocking I/O) [17]. Така модель дає змогу ефективно обробляти тисячі одночасних підключень завдяки механізму event loop, який забезпечує асинхронність виконання операцій та мінімізує затримки, пов’язані з очікуванням доступу до зовнішніх ресурсів (бази даних, файлової системи, зовнішніх API).

Node.js використовує високопродуктивний рушій V8, який компілює JavaScript безпосередньо у машинний код, що значно пришвидшує обробку складних операцій [17]. Завдяки цьому сервер здатний витримувати пікові навантаження під час активного проходження опитувань великою кількістю

респондентів без необхідності вертикального масштабування або застосування дорогих апаратних ресурсів.

Фреймворк Express.js було обрано як інструмент для побудови REST-орієнтованого API завдяки його легковаговій архітектурі та широким можливостям розширення. Express надає гнучкі механізми маршрутизації, що дозволяє створювати ієрархію ендпоінтів, групувати їх за функціональними модулями та чітко розмежовувати логіку контролерів. До екосистеми Express входить велика кількість middleware-компонентів, які забезпечують:

• логування запитів (morgan);

• обмеження швидкості звернень (rate-limit) для запобігання brute-force атак;

• автоматичний парсинг тіла запитів у форматах JSON і URL-encoded;

• налаштування CORS;

• захист HTTP-заголовків через Helmet;

• централізовану обробку помилок.

Завдяки цьому серверна частина системи отримує багаторівневий захист, модульність і можливість розширення без порушення загальної архітектури.

Інтеграція Express із JWT та Passport.js забезпечує реалізацію декількох способів автентифікації, включно з локальною (email/password) та авторизацією через Google OAuth 2.0. Це створює передумови для майбутнього розширення набору зовнішніх провайдерів (GitHub, Microsoft, Facebook) без необхідності змінювати основні механізми роботи з токенами.

Важливою складовою технологічного стеку є підтримка модулів штучного інтелекту. Інтеграція з API Google Gemini здійснюється через окремий сервісний шар, що відповідає за формування промптів, передавання запитів, обробку відповідей та кешування результатів. Це дозволяє використовувати AI для генерації опитувань, покращення запитань, аналізу текстових відповідей та побудови

рекомендацій. Обраний стек забезпечує можливість заміни моделі або провайдера без необхідності переписувати основна логіку, що відповідає принципам слабкої зв’язності компонентів.

Під час формування клієнтської частини системи було обрано React 19 з підтримкою TypeScript, що дає змогу строго типізувати компоненти, зменшити ризик помилок та покращити читабельність коду. Інструмент збирання Vite забезпечує надзвичайно швидке завантаження проєкту завдяки механізму Hot Module Replacement і оптимізації модулів у процесі компіляції. Для керування станом застосунку застосовано Zustand, який реалізує концепцію Flux-подібної архітектури з мінімальними накладними витратами та високою продуктивністю. Синхронізацію даних між клієнтом і сервером забезпечує TanStack React Query, який виконує кешування, повторні запити, регідратацію та автоматичне оновлення даних у разі зміни їх стану на сервері.

Візуальна частина інтерфейсу реалізована за допомогою Tailwind CSS, що дозволяє швидко створювати адаптивні компоненти з використанням utility-класів. Графічні модулі для побудови статистики відповідей реалізовано засобами Recharts, які надають інтерактивні та анімовані візуалізації з мінімальними вимогами до ресурсів.

Таким чином, обраний технологічний стек формує надійну, масштабовану та гнучку основу для веб-сервісу опитувань, забезпечує швидку взаємодію між клієнтом і сервером, підтримку AI-інструментів та можливість подальшого розширення функціональності без критичних змін у архітектурі.

Узагальнений перелік обраних технологій та інструментів розробки, що формують архітектуру MERN-додатку, подано у таблиці 4.2.

Таблиця 4.1 – Обґрунтування вибору технологічного стеку

Компонента системи	Технологія / Інструмент	Причина вибору для проекту
Backend	Node.js + Express.js	Асинхронна обробка запитів, єдина мова (JS) з фронтендом, висока швидкість I/O операцій.
Frontend	React 19 + TypeScript	Компонентний підхід, віртуальний DOM, сувора типізація для уникнення помилок.
База даних	MongoDB (NoSQL)	Гнучка схема даних (Schema-less), ідеальна для зберігання динамічних структур опитувань.
State Management	Zustand	Легковагова альтернатива Redux, простіша в налаштуванні та підтримці.
AI Модуль	Google Gemini API	Висока точність генерації тексту, безкоштовний рівень доступу для розробки.
Збірка та UI	Vite + Tailwind CSS	Миттєва збірка (HMR) та швидка стилізація без написання окремих CSS-файлів.

Зберігання даних реалізовано на основі документоорієнтованої бази даних MongoDB, яка оптимально відповідає специфіці веб-сервісу опитувань. На відміну від класичних реляційних СУБД, MongoDB не потребує жорсткого визначення структури таблиць заздалегідь, що є суттєвою перевагою для систем, де структура опитувань, типи запитань та їхні атрибути можуть змінюватися залежно від функціональних вимог. Документоорієнтований підхід дозволяє зберігати запитання у вигляді вкладених масивів усередині документа опитування, що значно спрощує виконання операцій читання, підвищує ефективність отримання повної структури

опитування і усуває необхідність складних JOIN-операцій, характерних для реляційних моделей.

Для управління даними використовується ORM-рішення Mongoose, яке забезпечує об’єктно-документне відображення, надає типізацію, механізми попередньої валідації, хуки життєвого циклу документів і налаштування індексації. За допомогою Mongoose створено строго визначені схеми для кожної з колекцій, що дозволяє поєднувати гнучкість MongoDB із контролем структури даних, необхідним для забезпечення цілісності системи.

Структура бази даних включає чотири базові колекції, кожна з яких реалізує окремий функціональний модуль системи та має власну логіку зберігання та індексації.

1. Колекція User

Колекція відповідає за зберігання основної інформації про користувачів системи та реалізацію механізмів авторизації. Структура включає:

• email користувача з унікальним індексом;

• хешований пароль, згенерований за алгоритмом bcrypt;

• ім’я та роль (user / admin), що визначає рівень доступу;

• дані автентифікації через Google OAuth (id, email, аватар);

• дату створення та оновлення запису.

Індексація email та ідентифікатора OAuth дозволяє швидко виконувати пошук під час автентифікації та забезпечує стабільну продуктивність при великій кількості зареєстрованих користувачів.

2. Колекція Survey

Колекція опитувань має найбільш складну структуру, оскільки містить:

• назву та опис опитування;

• масив запитань, кожне з яких включає тип (radio, checkbox, text, number, rating), текст питання, список варіантів відповіді та правила обов’язковості;

• умовну логіку (skip-logic), яка визначає переходи між питаннями залежно від вибраних відповідей;

• налаштування доступності — дата завершення, максимальна кількість учасників;

• статус опитування (активне / завершене);

• ідентифікатор автора та список співавторів.

Лістинг 4.1 – Фрагмент Mongoose-схеми опитування та схеми питання у базі даних MongoDB

```
const surveySchema = new mongoose.Schema({
  // Основна інформація
  title: {
    type: String,
    required: [true, 'Назва опитування обов\'язкова'],
    trim: true,
    maxlength: [200, 'Назва не може бути довшою за 200 символів']
  },
  description: {
    type: String,
    trim: true,
    maxlength: [1000, 'Опис не може бути довшим за 1000 символів']
  },
  // Питання (вкладена схема)
  questions: [questionSchema],
    creator: {
```

```
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  collaborators: [{
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    addedAt: {
      type: Date,
      default: Date.now
    }
  }],
```

Повний лістинг схеми опитування можна переглянути в додатку А.1 - Схема опитування з вкладеними питаннями та skip logic.

Структура побудована з використанням вкладених документів, що дозволяє виконувати всі запити до опитування за одну операцію читання. Такий підхід значно оптимізує продуктивність, оскільки реляційна модель вимагала б численних JOIN або окремих звернень до таблиць виключно для отримання структури запитань.

3. Колекція Response

Колекція відповідей містить великі масиви записів, що надсилаються респондентами під час проходження опитування. Кожен документ включає:

• ідентифікатор опитування;

• масив відповідей респондента;

• час надсилання;

• userAgent та IP-адресу (для базового захисту від повторних проходжень);

Надзвичайно важливим елементом є індексація поля surveyId. Це дає змогу швидко виконувати агрегацію статистики, отримувати відповіді конкретного опитування та забезпечує масштабування колекції до сотень тисяч записів без втрати продуктивності.

Документоорієнтована модель дозволяє швидко обробляти записи та передавати їх агрегованими у звітність, що є критичним для побудови графіків, діаграм і аналітичних моделей.

4. Колекція RefreshToken

Колекція використовується для зберігання refresh-токенів у випадках, коли система передбачає можливість завершення всіх сесій користувача або ревокацію токенів. Структура містить:

• сам токен у зашифрованому або хешованому вигляді;

• userId власника токена;

• дату створення;

• автоматичний TTL-індекс, який видаляє токен після закінчення строку його дії.

TTL-індексація реалізується механізмом MongoDB expireAfterSeconds, що дозволяє автоматизувати очищення колекції без втручання з боку програмної логіки.

На рисунку 4.1 подано ER-діаграму, яка відображає основні сутності системи, їх атрибути та логічні зв’язки між ними.

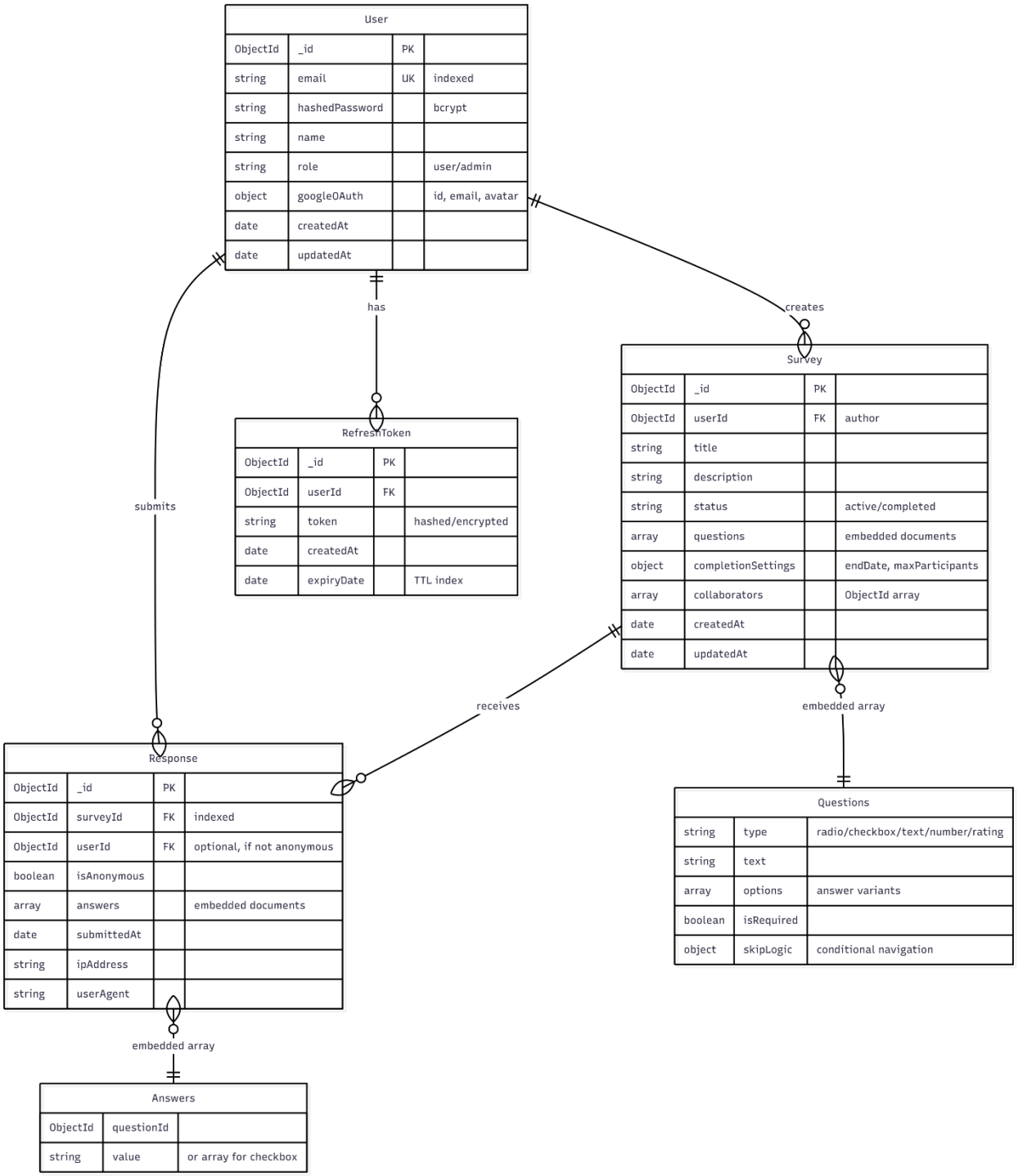

Рисунок 4.1 – ER-діаграма структури бази даних веб-сервісу опитувань

Діаграма демонструє взаємозв’язок між колекціями User, Survey, Response та RefreshToken, а також внутрішні вкладені структури документів Questions і Answers, характерні для документоорієнтованої моделі MongoDB.

На діаграмі видно, що користувач може створювати декілька опитувань та надсилати відповіді, а також мати декілька активних сесій, представлених у вигляді refresh-токенів. Опитування містить вкладений масив запитань різних типів, а

відповідь — масив окремих відповідей на кожне запитання. Така структура забезпечує оптимальну продуктивність під час читання великих обсягів даних та відповідає принципам денормалізації, яка рекомендована для високонавантажених NoSQL-систем.

Побудована структура забезпечує:

• високу продуктивність завдяки зберіганню запитань й відповідей у вкладених документах;

• масштабованість — колекції можуть розростатися без потреби у реорганізації структури;

• адаптивність — додавання нових типів запитань і параметрів не потребує зміни моделі даних;

• цілісність даних завдяки валідації на рівні Mongoose-схем;

• зменшення кількості запитів до бази — один документ містить повний набір даних для рендерингу опитування;

• стійкість до високих навантажень під час паралельного проходження опитування багатьма респондентами.

Обрана архітектура бази даних дозволяє системі опитувань функціонувати стабільно, швидко та без втрат продуктивності, забезпечуючи можливість подальшого масштабування та розширення функціональності.

Серверна частина веб-сервісу реалізована у вигляді RESTful API, яке забезпечує стандартизовану та уніфіковану взаємодію між клієнтським застосунком і сервером. API структуровано за модульним принципом і поділено на п’ять логічних груп маршрутів, кожна з яких відповідає за окремий функціональний аспект системи. Така побудова дозволяє чітко розмежувати відповідальність між компонентами backend-сервісу, спрощує тестування та полегшує подальший супровід і масштабування.

Перша група маршрутів /api/auth/ охоплює всі процеси, пов’язані з автентифікацією та управлінням доступом. Вона включає реєстрацію нових користувачів, авторизацію, вихід із системи, оновлення пари токенів за механізмом dual-token, а також процедури відновлення доступу через електронну пошту. Усі ці операції реалізуються із застосуванням Passport.js, JSON Web Token та криптографічних алгоритмів (bcrypt, HMAC/RS256), що гарантує відповідність сучасним вимогам безпеки.

Маршрути групи /api/surveys/ забезпечують повний життєвий цикл опитувань. Вони дозволяють створювати нові опитування з динамічною структурою запитань, редагувати їх, змінювати параметри доступу, видаляти або архівувати, а також отримувати статистичну інформацію. Окрім цього, у межах цього модуля реалізовані методи інтеграції зі штучним інтелектом — генерація структури опитування за допомогою моделей Google Gemini та покращення або переписування запитань для підвищення якості форм.

Група маршрутів /api/surveys/:id/responses забезпечує приймання відповідей від респондентів, первинну валідацію коректності даних, фіксацію метаданих (IP-адреса, userAgent, час проходження) та збереження у MongoDB. Також передбачено механізми експорту відповідей у JSON або CSV, що дозволяє інтегрувати отримані дані в інші інформаційні системи або проводити зовнішній аналіз.

Маршрути /api/ai/ реалізують логіку взаємодії з вбудованими AI-модулями. До їх функціоналу входить автоматична генерація нового опитування за описом теми, покращення вже створених запитань, аналіз відкритих відповідей за допомогою моделей природної мови, а також робота інтерактивного AI-чату, який допомагає авторам у налаштуванні структури опитування. Завдяки застосуванню AI Façade-pattern система може використовувати різні AI-провайдери — Google Gemini, Hugging Face або fallback-моделі. Логіка генерації опитування на основі теми та цілі реалізована в контролері aiController, який викликає відповідний сервіс взаємодії з Google Gemini. Фрагмент коду наведено в лістингу 4.2.

Лістинг 4.2 – Фрагмент контролера та сервісу формування запиту до AI-провайдера Google Gemini

```
// controllers/aiController.js
const generateSurvey = async (req, res) => {
  const { topic, goal, questionCount, questionTypes } = req.body;
  // Валідація вхідних даних
  if (!topic || topic.trim().length === 0) {
    return res.status(400).json({ message: 'Тема опитування обов\'язкова' });
  }
  // Генерація структури опитування через AI-сервіс
  const surveyData = await aiService.generateSurvey({
    topic: topic.trim(),
    goal: goal || 'збір відгуків',
    questionCount: questionCount || 7,
    questionTypes: questionTypes || ['radio', 'checkbox', 'text']
  });
  return res.status(200).json({ success: true, data: surveyData });
};
// config/aiService.js – виклик Google Gemini
async generate_gemini(topic, goal, questionCount, questionTypes) {
  const model = this.gemini.getGenerativeModel({ model: 'gemini-2.5-flash' });
  // Формування промпта
  const prompt = `Створи опитування українською мовою на тему "${topic}".
Ціль: ${goal}
Кількість питань: ${questionCount}
ВАЖЛИВО: Відповідай ТІЛЬКИ у форматі JSON.
Формат відповіді:
{
  "title": "Назва опитування",
  "description": "Короткий опис",
  "questions": [
    {
      "text": "Текст питання",
      "type": "radio",
```

```
      "options": [{"text": "Варіант 1", "value": "value1"}],
      "required": true
    }
  ]
}

Типи питань: ${questionTypes.join(', ')}
Створи ${questionCount} професійних питань українською мовою.`;
  // Виклик Gemini API
  const result = await model.generateContent(prompt);
const response = await result.response;
  const text = response.text();
  // Парсинг JSON-відповіді моделі
  return this.parseAIResponse(text, questionTypes, 'gemini');
}
```

Наведений фрагмент демонструє типовий сценарій виклику AI-провайдера: контролер отримує параметри опитування з HTTP-запиту, передає їх до сервісу aiService, який формує промпт українською мовою, викликає модель gemini-2.5-flash та повертає структуроване JSON-представлення опитування.

Окрема група /api/export/ відповідає за формування підсумкових документів. Вона підтримує генерацію PDF-звіту зі статистикою, а також підготовку агрегованих таблиць для подальшої аналітики. Для цього використовуються спеціалізовані бібліотеки формування документів та інструменти оптимізації для великих обсягів даних.

Робота всіх маршрутів координується відповідними контролерами: authController, surveyController, responseController та aiController. Кожен контролер відповідає за обробку запитів, виклик бізнес-логіки та формування стандартизованих відповідей. Така організація забезпечує суворе дотримання принципів розділення відповідальності, що значно спрощує тестування, підвищує безпомилковість коду та забезпечує можливість масштабування проєкту без порушення цілісності архітектури.

Система безпеки веб-додатку побудована як багаторівневий комплекс механізмів, спрямованих на захист персональних даних, запобігання поширеним веб-атакам та забезпечення контролю доступу до API. Використані рішення відповідають сучасним рекомендаціям OWASP та враховують специфіку розподілених REST-систем, які працюють із конфіденційною інформацією.

Одним з основних елементів системи безпеки є хешування паролів. Для цього використовується бібліотека bcryptjs з параметром обчислювальної складності (salt rounds) 12, що забезпечує стійкість до атак перебору та робить неможливим відновлення початкового пароля навіть у випадку компрометації бази даних. Використання сольових значень унеможливлює застосування rainbow-таблиць та інших передобчислених методів злому.

Критично важливу роль у механізмах автентифікації відіграє JWT-система, яка включає пари Access Token і Refresh Token. Access Token має строк дії 8 годин і використовується для виконання запитів до захищених маршрутів. Refresh Token зберігається у захищеному httpOnly cookie, недоступному для клієнтських скриптів, що значно знижує ризики XSS-атак. Термін дії refresh-токена встановлено на 7 днів, що забезпечує зручність тривалих сесій при збереженні високого рівня безпеки. Процедура оновлення токенів виконується сервером із валідацією підпису та стану токена, що унеможливлює несанкціоноване продовження сесії.

Для захисту від зловмисних звернень реалізовано механізм rate limiting. Зокрема, кількість запитів на вхід до системи обмежено десятком за 15 хвилин, а для операцій з відновленням пароля встановлено ще більш суворе обмеження — три спроби за той самий часовий проміжок. Це мінімізує ризики brute-force атак та знижує навантаження на систему при масових автоматизованих запитах.

Суттєвий внесок у загальний рівень безпеки забезпечують HTTP-заголовки, налаштовані за допомогою Helmet. Використовується політика Content Security Policy (CSP), що дозволяє контролювати джерела скриптів і запобігає виконанню шкідливих скриптів. Активовано механізми X-Frame-Options для захисту від

clickjacking-атак, X-Content-Type-Options для запобігання MIME-sniffing, а також інші рекомендації OWASP. Усі ці заходи роблять фронтенд-інтерфейс менш вразливим до XSS, CSRF і маніпуляцій з візуальними елементами.

Додатковий рівень захисту забезпечує коректно налаштований CORS-механізм. Система визначає білий список дозволених доменів, які можуть надсилати запити до API, а також дозволяє передавання cookie лише у захищеному режимі (credentials: true). Це запобігає несанкціонованим крос-доменним запитам та зменшує ризики CSRF.

Важливим аспектом є валідація даних. На рівні Mongoose-моделей реалізовано схеми з обмеженнями типів, допустимих значень та обов'язкових полів. Після цього додаткові перевірки виконуються на серверному рівні, що забезпечує подвійну фільтрацію даних і запобігає ін'єкційним атакам та некоректному збереженню даних.

Захист доступу до API забезпечується через систему middleware-перевірок. Перед виконанням кожного запиту токен проходить верифікацію, після чого система визначає рівень доступу користувача, що дозволяє обмежувати роботу лише авторизованим або адміністративним обліковим записам. Для публічних сторінок передбачено механізм optionalAuth, який дозволяє надавати доступ без авторизації, проте верифікувати токен при його наявності.

Застосований комплекс заходів дозволяє гарантувати відповідність веб-сервісу сучасним нормативам безпеки, забезпечує стійкість до основних типів атак (XSS, CSRF, SQL/NoSQL-injection, brute-force), а також підтримує захист персональних і статистичних даних користувачів.

Для наочного представлення архітектури безпеки веб-сервісу розроблено багаторівневу діаграму, що відображає комплексну взаємодію між механізмами захисту та потоками даних (рисунок 4.2). Діаграма ілюструє, як саме обробляється запит користувача — починаючи від моменту потрапляння до системи через захищений HTTPS-канал і завершуючи перевіркою даних у базі.

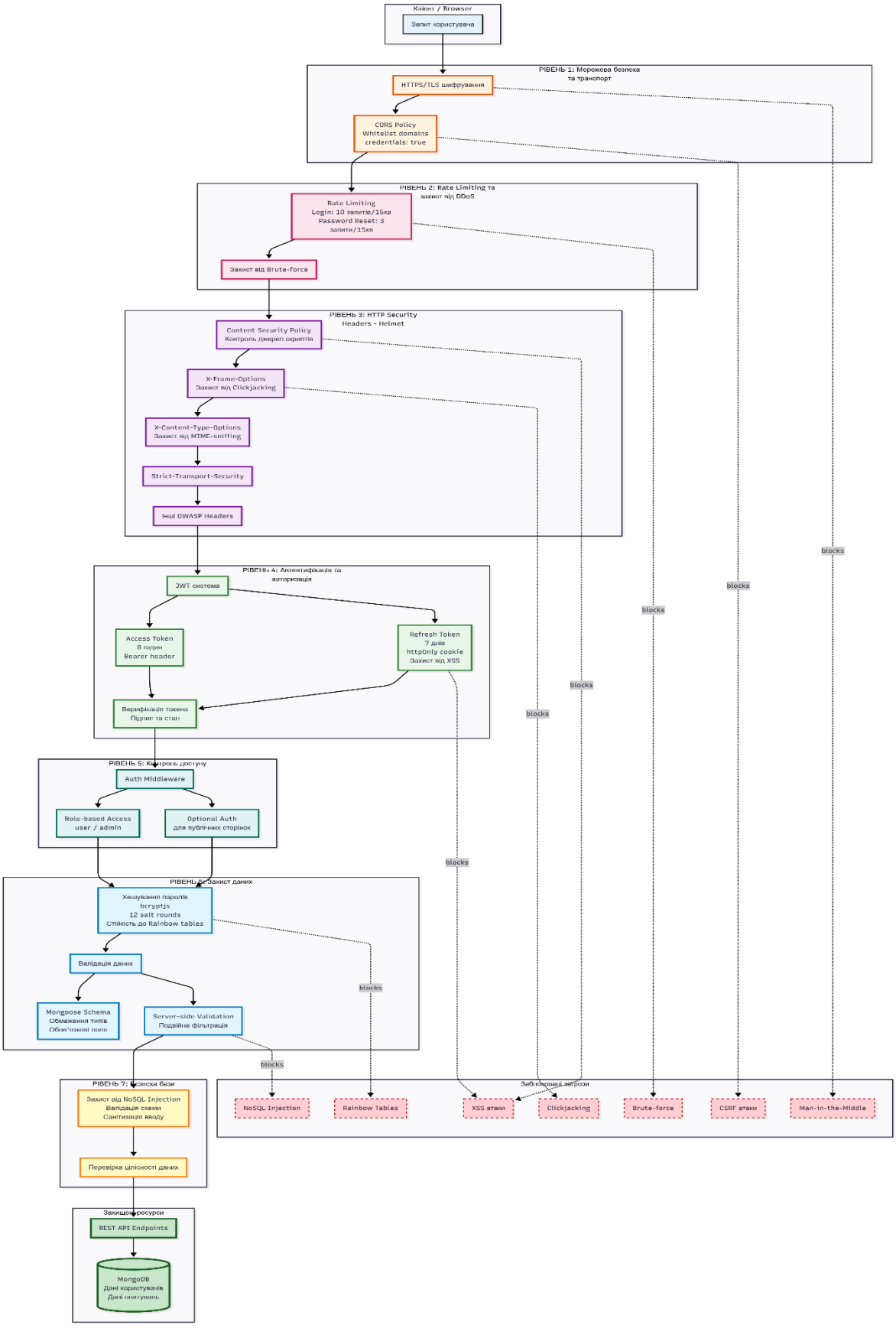

Рисунок 4.2 – Багаторівнева діаграма системи безпеки веб-сервісу опитувань

Усього система безпеки охоплює сім рівнів захисту, кожен із яких відповідає за окремий аспект безпеки та блокує певний клас загроз.

На першому рівні реалізовано мережевий захист, що включає протокол HTTPS/TLS та політику CORS. Другий рівень відповідає за обмеження частоти запитів та протидію brute-force атакам. Третій рівень складається з набору HTTP-заголовків безпеки Helmet, які забезпечують захист від XSS, clickjacking та MIME-sniffing.

Четвертий рівень охоплює механізми автентифікації й авторизації, що реалізовані через систему токенів (Access і Refresh Tokens), їх верифікацію та захищене зберігання. П’ятий рівень забезпечує контроль доступу до API шляхом перевірки ролей і застосування middleware-фільтрів. На шостому рівні виконується перевірка коректності даних, їх валідація та хешування паролів через bcrypt. Сьомий рівень забезпечує захист бази даних від NoSQL-ін’єкцій, збереження цілісності та структури даних.

У правій частині діаграми також наведено перелік загроз (XSS, CSRF, brute-force, clickjacking тощо), які блокуються відповідними механізмами на різних етапах обробки запиту. Таким чином, діаграма демонструє комплексний підхід до безпеки, де кожен шар підсилює попередній та створює багаторівневий бар’єр захисту для всієї системи опитувань.

4.2. Інструкція користувача системи

Веб-сервіс опитувань є доступним у мережі Інтернет і не потребує встановлення додаткового програмного забезпечення. Для роботи з системою користувачу необхідно перейти за постійною адресою веб-додатку:

https://survey-app-ua-d6ffd2c53e7c.herokuapp.com/login

Завантаження застосунку здійснюється через веб-браузер, після чого користувач автоматично перенаправляється на сторінку автентифікації. Інтерфейс побудований за принципами адаптивного дизайну, тому коректно відображається як на стаціонарних комп’ютерах, так і на мобільних пристроях.

На сторінці входу доступні два способи авторизації:

Стандартна автентифікація через електронну адресу та пароль.

Користувач вводить зареєстровану електронну адресу та пароль, після чого система перевіряє коректність даних та створює пару токенів доступу (Access та Refresh Tokens), що забезпечує подальшу роботу в межах сесії.

Авторизація через Google-акаунт.

У разі використання цього методу користувач перенаправляється на офіційну сторінку сервісу Google OAuth 2.0, де підтверджує дозвіл на автентифікацію. Після успішного повернення в систему створюється новий обліковий запис або прив’язується існуючий.

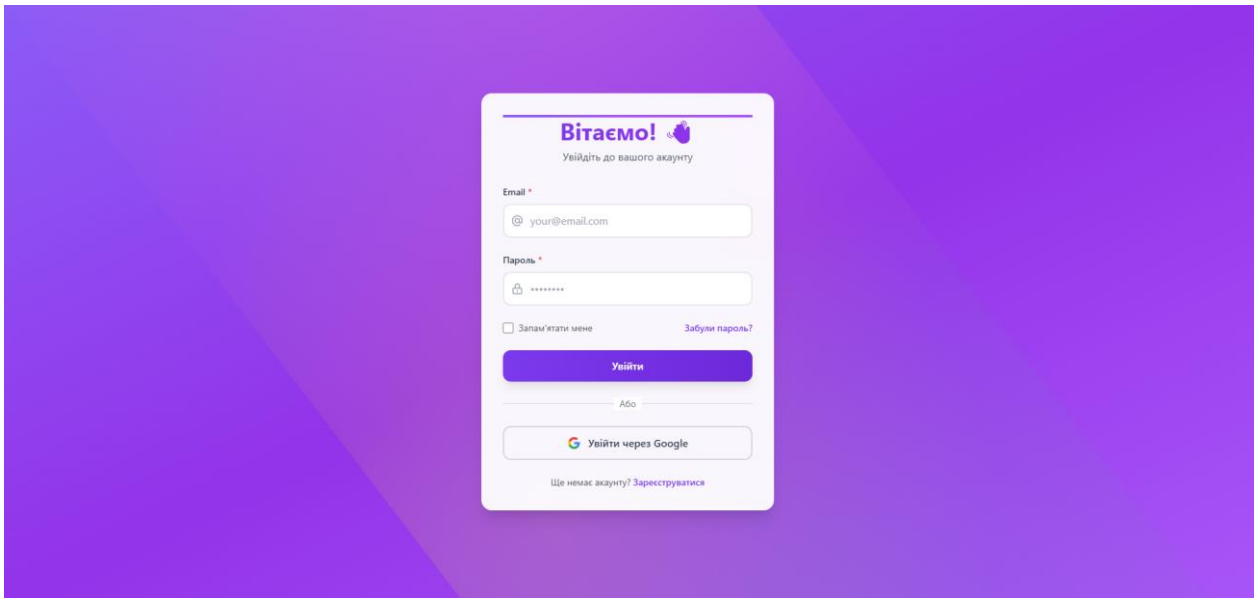

Рисунок 4.3 – Вікно авторизації веб-сервісу для генерації опитувань

У разі втрати доступу користувач може скористатися механізмом відновлення пароля. Для цього призначена опція «Forgot password», яка дозволяє надіслати на електронну адресу посилання для створення нового пароля.

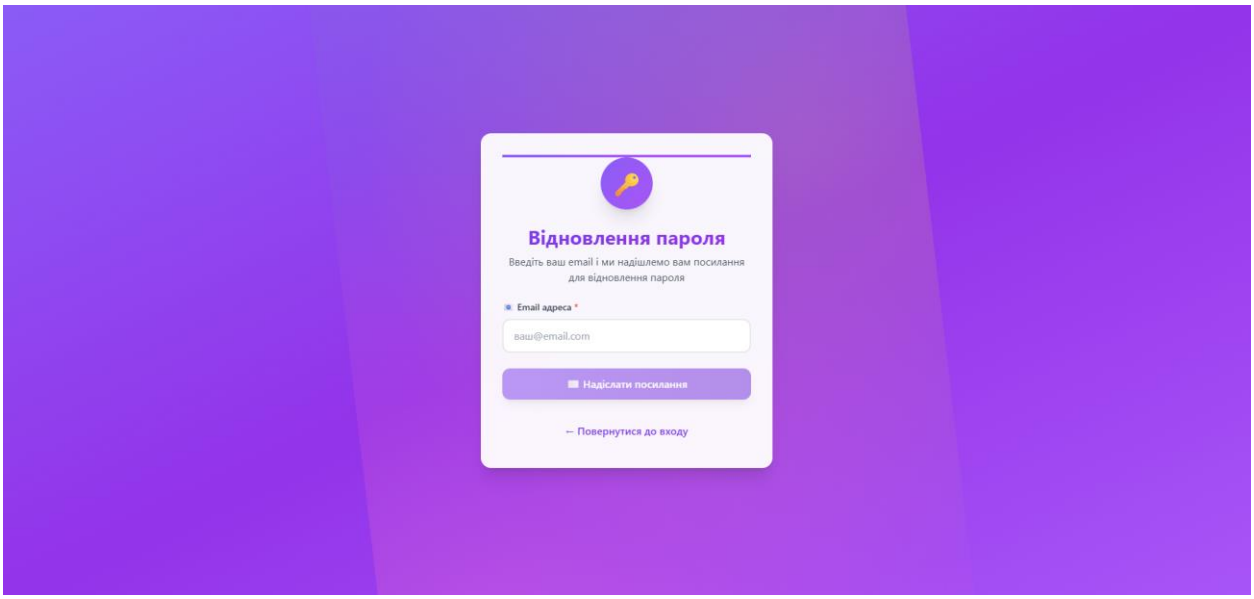

Рисунок 4.4 – Сторінка відновлення паролю

Після успішної автентифікації користувач переноситься до особистого кабінету, у якому він може створювати, переглядати та редагувати власні опитування.

Створення нового опитування

Після успішної авторизації користувач переходить до особистого кабінету, де розміщено список власних опитувань та панель керування. Для створення нового опитування необхідно натиснути кнопку “Створити опитування”, після чого відкривається інтерактивний конструктор.

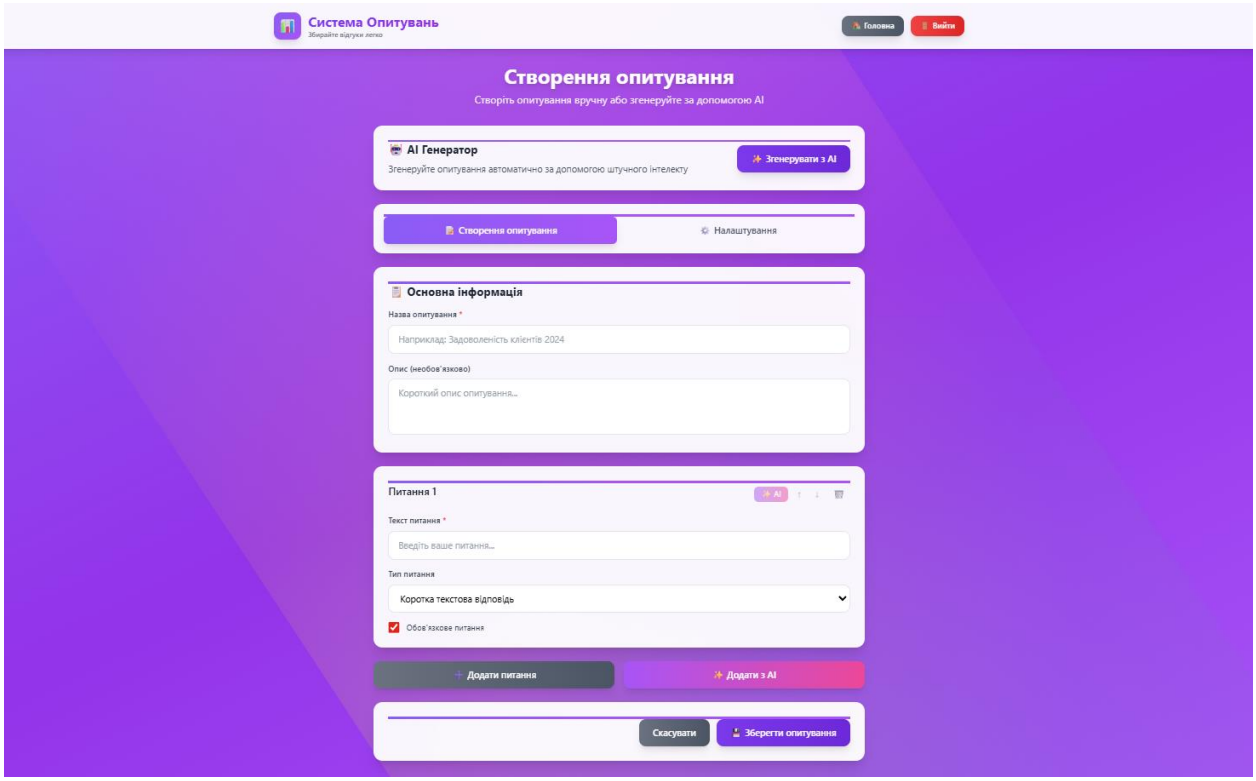

Рисунок 4.5 – Сторінка створення опитування

Інтерфейс створення опитування побудований таким чином, щоб користувач міг послідовно формувати структуру форми та налаштовувати її параметри без необхідності володіння технічними знаннями. У процесі створення необхідно виконати такі кроки:

1. Вказати основну інформацію про опитування.

Користувач задає назву, короткий опис і за потреби — категорію або тематичну область. Ці дані відображаються на публічній сторінці та допомагають респондентам зрозуміти мету опитування.

2. Додати запитання.

Конструктор дозволяє створювати запитання шести типів: текстові, числові, питання з одним або декількома варіантами відповіді, шкали оцінювання та вибір дати. Кожне запитання має окремі налаштування, зокрема: обов’язковість, можливість додавання варіантів відповіді.

Веб-сервіс підтримує можливість автоматичного створення опитувань за допомогою інтегрованого модуля штучного інтелекту. Ця функція значно спрощує роботу користувача, дозволяючи формувати якісні структури опитувань на основі короткого текстового опису або теми дослідження. Модуль AI працює на базі Google Gemini та додаткових моделей Hugging Face, що забезпечує високу точність формування запитань та адаптивність до різних предметних областей.

Для використання AI-генерації користувач переходить на сторінку створення опитування та обирає режим “Створити за допомогою AI”. Після цього відкривається окреме діалогове вікно, де необхідно вказати тему або мету опитування. Сама взаємодія з AI побудована у форматі короткої підказки: система аналізує введений текст і пропонує набір запитань, який найбільше відповідає заявленій темі. При цьому формуються різні типи запитань — текстові, вибіркові, шкальні, рейтингові — залежно від контексту.

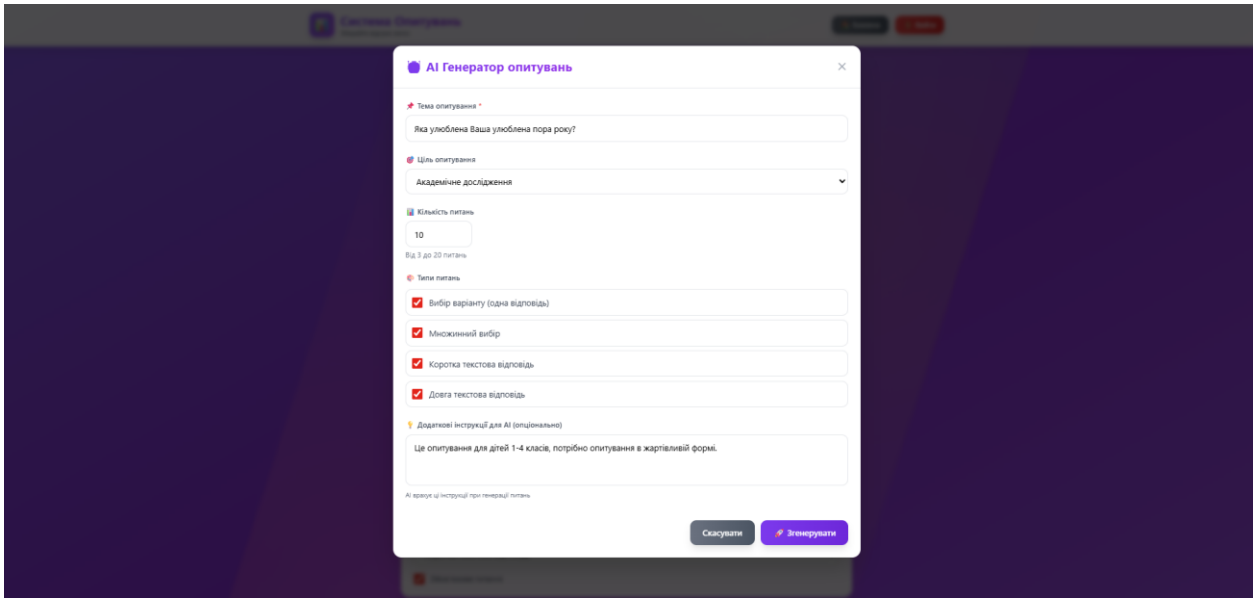

Рисунок 4.6 – Вікно AI генератора опитувань.

Перед додаванням згенерованого набору користувач має можливість переглянути попередній результат, відредагувати текст формулювань або видалити зайві запитання. Це дозволяє поєднати точність автоматичного підходу з гнучкістю ручного налаштування. У разі потреби користувач може повторити генерацію — система сформує альтернативний варіант структури опитування.

Після підтвердження запитання автоматично додаються до редактора опитування, де можуть бути доопрацьовані за всіма доступними параметрами: логічні переходи, обов'язковість, типи відповідей, варіанти вибору. Для складних тематик AI-модуль також може запропонувати варіанти відповідей для питань типу "radio" та "checkbox", що значно прискорює процес підготовки опитування.

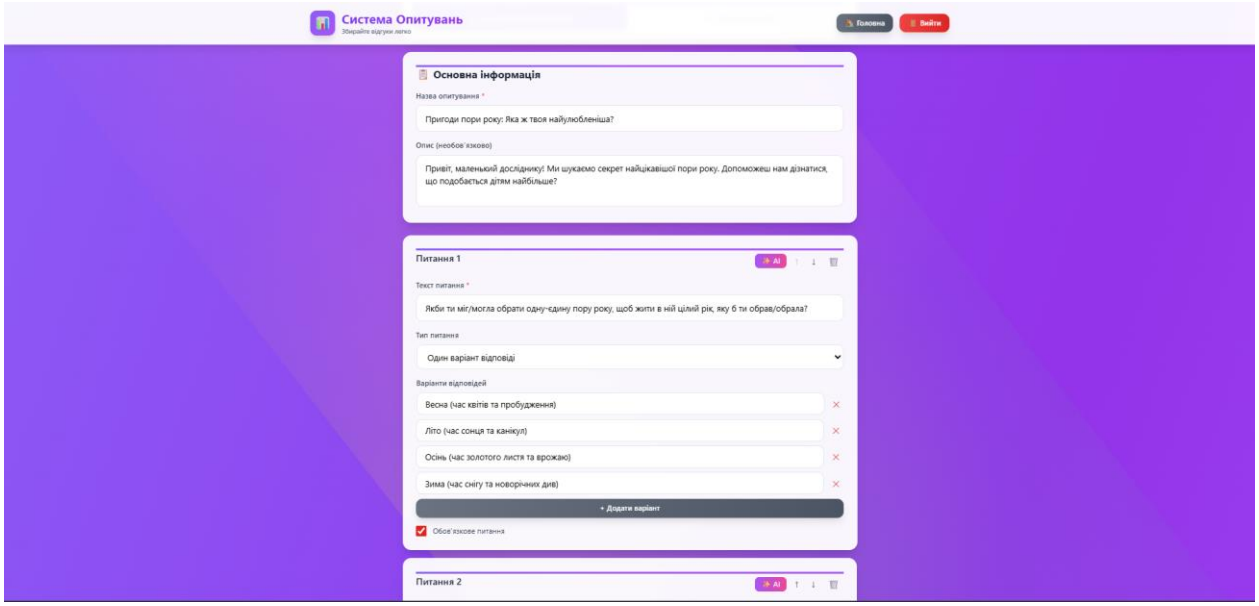

Рисунок 4.7 – Приклад створеного за допомогою AI опитування

Перевагою використання AI є можливість швидко отримати якісну основу для майбутнього опитування навіть без глибоких знань у методології анкетування. Це особливо корисно у випадках, коли потрібно створити опитування за обмежений час або коли автор не має достатнього досвіду в структуризації запитань. Крім того, AI-модуль адаптується до різних стилів та тематичних запитів, що дозволяє використовувати його як для академічних досліджень, так і для внутрішніх корпоративних опитувань або маркетингових форм.

Завдяки такій функціональності веб-сервіс забезпечує не лише інструмент для створення опитувань, але й інтелектуального асистента, який допомагає формувати структуровані та методологічно коректні анкети з мінімальними зусиллями з боку користувача.

3. Налаштувати умовну логіку (опціонально).

Для окремих типів питань можна додати правила переходів. Це дозволяє створювати адаптивні анкети, в яких подальші запитання залежать від попередніх відповідей респондента.

4. Встановити параметри доступності.

Користувач визначає дату завершення опитування, максимальну кількість учасників, а також рівень доступу — публічний або приватний. Приватні опитування можуть бути доступні лише за спеціальним посиланням.

5. Додати співавторів (за потреби).

Система підтримує спільну роботу над опитуванням: автор може надати доступ до редагування іншим зареєстрованим користувачам. Це корисно для командних досліджень або групових проєктів.

6. Зберегти та опублікувати опитування.

Після заповнення всіх параметрів опитування можна одразу опублікувати. Система автоматично генерує унікальне публічне посилання, яке можна поширювати серед респондентів через будь-які канали — електронну пошту, соціальні мережі або інші засоби комунікації.

Після публікації опитування система генерує унікальне посилання, за яким респонденти можуть пройти анкету без необхідності реєстрації. Це забезпечує максимально простий доступ і сприяє збільшенню кількості відповідей. Перехід за посиланням відкриває публічну сторінку опитування, де користувачу відображається назва, опис та інтерактивний інтерфейс запитань.

Під час проходження опитування респондент взаємодіє з покроковою формою, яка автоматично перевіряє правильність введених даних. Для обов’язкових питань передбачено миттєву валідацію, що не дозволяє перейти до наступного етапу, доки не буде надано відповідь. У разі використання умовної логіки система

динамічно приховує або відображає певні запитання залежно від попередніх відповідей, формуючи індивідуальний сценарій проходження.

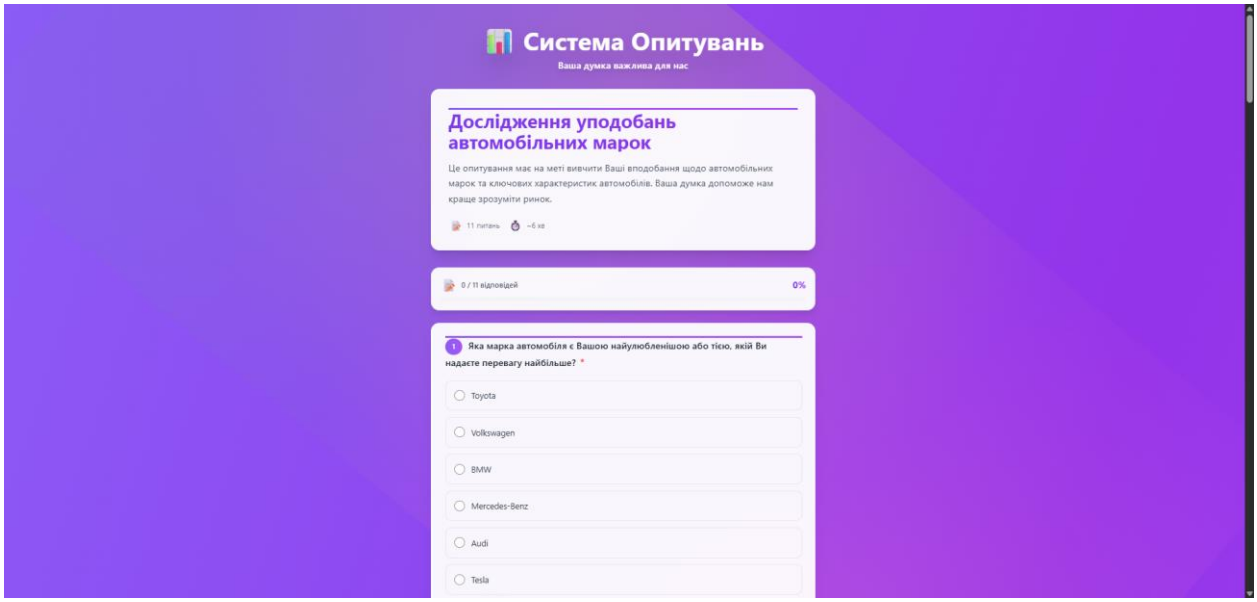

Рисунок 4.8 – Публічна сторінка проходження опитування

Після завершення заповнення всі дані надсилаються на сервер, де система додатково перевіряє:

• чи не перевищено встановлений автором ліміт респондентів;

• чи не сплив термін доступності;

• чи не надійшла відповідь повторно з того самого браузера або IP (якщо встановлені відповідні обмеження);

• чи відповідає структура отриманих даних вимогам опитування.

Лише після успішної перевірки відповідь зберігається у базі даних та прив’язується до відповідного опитування. Система фіксує дату й час надсилання, а також анонімізовані технічні параметри — IP-адресу та userAgent, що забезпечує захист від багаторазового проходження та дозволяє виявляти аномальну активність.

Оскільки інтерфейс публічної форми повністю адаптивний, опитування коректно відображається як на комп’ютерах, так і на мобільних пристроях, забезпечуючи зручність респондентам та підвищуючи завершуваність проходження.

Після завершення збору відповідей система надає автору опитування інструменти для детального перегляду й аналізу результатів. Усі статистичні дані обробляються в реальному часі, що дозволяє отримувати актуальну інформацію навіть під час активного надходження відповідей. Доступ до звітів здійснюється через окрему сторінку інтерфейсу, де автор може ознайомитися зі структурою відповідей у зручному та візуально зрозумілому форматі.

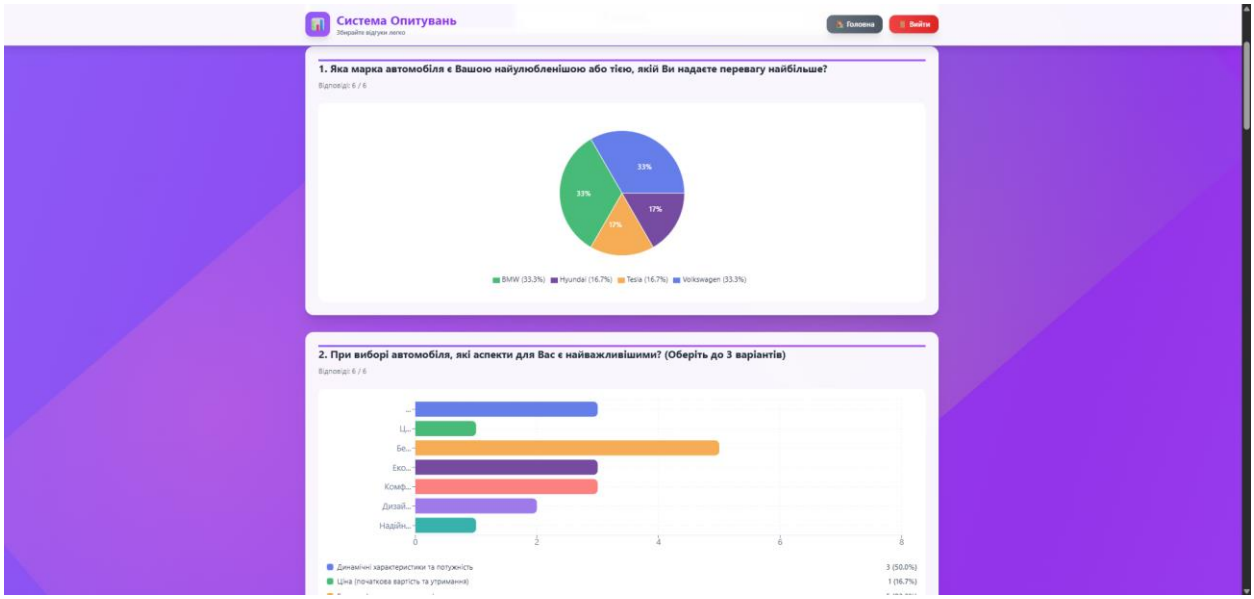

Рисунок 4.9 – Сторінка аналізу результатів

Для кожного запитання система формує окремий блок зі статистичними показниками, які залежать від типу запитання. Для варіантів вибору (radio, checkbox) автоматично обчислюється кількість виборів кожного варіанту, будується діаграма (стовпчикова або кругова) та обчислюються відсоткові співвідношення. Для числових або рейтингових запитань відображаються мінімальні, максимальні та середні значення. Текстові відповіді подаються у вигляді списку з можливістю швидкого перегляду.

Окрім базової статистики, система підтримує використання інтегрованих AI-модулів, які дозволяють глибше інтерпретувати дані. За допомогою Google Gemini користувач може автоматично отримати підсумковий аналіз результатів, включно з виявленням тенденцій, типових патернів та ймовірних причин поведінки респондентів. Для текстових запитань доступна класифікація, стислий огляд або

генерація рекомендацій, що значно спрощує роботу з великими обсягами відкритих відповідей.

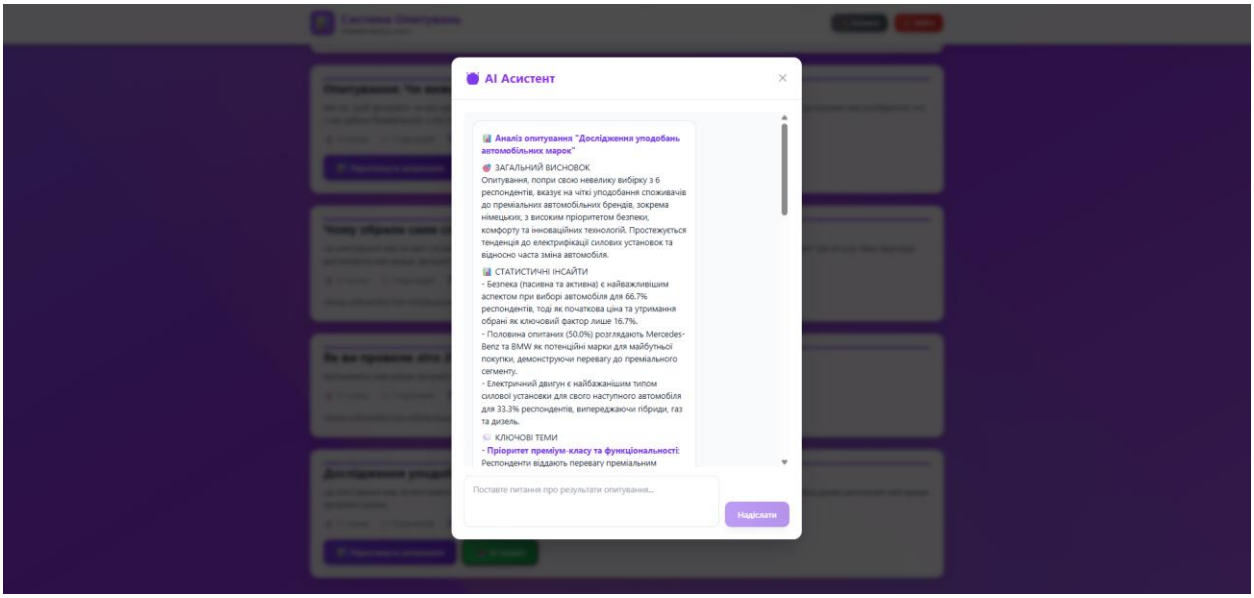

Рисунок 4.10 – Вікно AI асистента

Для подальшого використання результати опитування можуть бути експортовані у формат PDF — повноцінний звіт, що містить метадані опитування, агреговану статистику, діаграми та індивідуальні відповіді.

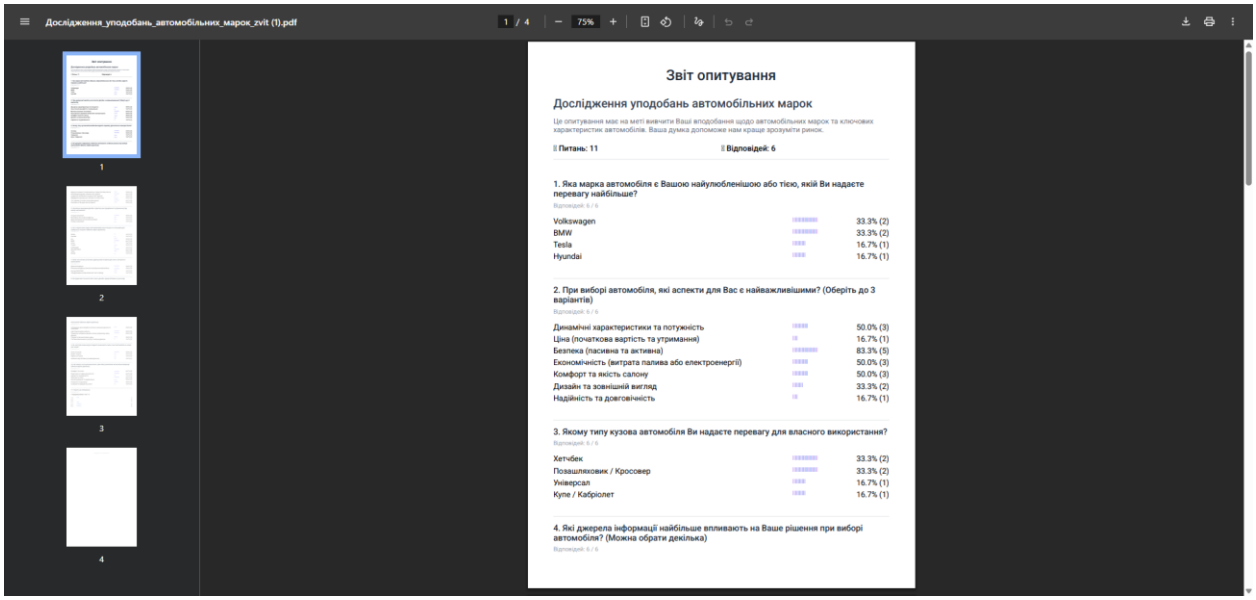

Рисунок 4.11 – Експортований звіт в PDF

Експорт формується безпосередньо на сервері, що гарантує цілісність даних і правильність форматування. Після генерації користувачу пропонується завантажити файл одним кліком.

Завдяки такому підходу процес аналізу результатів стає не лише гнучким і наочним, але й придатним для подальшого використання в дослідженнях, звітах або презентаціях, що значно розширює практичну цінність веб-сервісу.

Розглянута інструкція користувача демонструє, що веб-сервіс опитувань забезпечує повний, логічно вибудований цикл взаємодії з системою — від процесу автентифікації та створення опитування до його проходження респондентами та подальшого аналізу результатів. Завдяки продуманій структурі інтерфейсу, підтримці адаптивного дизайну та використанню системи підказок користувач може швидко освоїти базовий і розширений функціонал. Інтеграція штучного інтелекту значно спрощує підготовку змістовних опитувань, а можливості експорту та візуалізації даних роблять сервіс зручним для практичного застосування у дослідженнях, освітніх та корпоративних середовищах. Таким чином, реалізована система не лише відповідає вимогам до сучасного інструменту для створення анкет, але й підвищує ефективність роботи користувача за рахунок автоматизації ключових етапів.

ВИСНОВКИ

У магістерській роботі вирішено актуальне науково-прикладне завдання розробки веб-сервісу для автоматизованого створення та аналізу опитувань із використанням технологій штучного інтелекту. На основі проведених теоретичних та практичних досліджень встановлено, що більшість існуючих рішень для проведення онлайн-опитувань, таких як Google Forms, SurveyMonkey, Typeform та LimeSurvey, мають суттєві обмеження у безкоштовних версіях, зокрема щодо кількості запитань, логіки переходів та можливостей експорту даних. Це підтвердило необхідність створення доступного інструменту, який поєднує гнучкість налаштувань із сучасними можливостями генеративного штучного інтелекту.

Для реалізації системи було обґрунтовано та використано технологічний стек MERN (MongoDB, Express.js, React, Node.js). Вибір такої архітектури дозволив забезпечити високу продуктивність обробки запитів, масштабованість веб-сервісу та ефективну роботу з неструктурованими даними опитувань завдяки використанню документоорієнтованої бази даних MongoDB. Побудова клієнтської частини на базі бібліотеки React 19 із застосуванням мови TypeScript гарантувала надійність програмного коду та високу реактивність інтерфейсу користувача.

У результаті виконання роботи спроєктовано та програмно реалізовано повнофункціональний веб-сервіс, який включає модуль авторизації, конструктор опитувань та аналітичну панель. Ключовою особливістю розробленої системи стала інтеграція з API Google Gemini, що дозволило автоматизувати процес створення анкет на основі текстового опису теми, а також забезпечити інтелектуальний аналіз відкритих відповідей респондентів. Вагомим функціональним досягненням стала реалізація механізму умовної логіки (Skip Logic), який надає можливість адаптувати сценарій проходження опитування залежно від відповідей користувача, що значно підвищує якість зібраних даних порівняно з лінійними анкетами.

Особливу увагу в роботі приділено питанням безпеки та захисту даних. Впроваджено надійну систему автентифікації на основі стандарту JWT із використанням dual-token підходу (Access Token та Refresh Token) та хешування паролів за алгоритмом bcrypt. Додатково налаштовано захист від поширених веб-атак, зокрема XSS, CSRF та brute-force, за допомогою бібліотеки Helmet та механізмів валідації вхідних даних.

Практична цінність роботи полягає у створенні готового до впровадження програмного продукту, який дозволяє суттєво скоротити час на підготовку досліджень та автоматизувати обробку результатів. Система забезпечує візуалізацію статистики в реальному часі та експорт звітів у форматі PDF, тому може бути ефективно використана у навчальному процесі, маркетингових дослідженнях та HR-діяльності. Результати роботи пройшли апробацію на науково-практичних конференціях, що підтверджує їх актуальність та відповідність сучасним тенденціям розвитку веб-технологій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Rubin R. The Best Online Survey Tools. *PCMag*. URL: https://www.pcmag.com/picks/the-best-online-survey-tools

2. SurveyMonkey Free Vs. Paid Surveys & Forms. *SurveyMonkey*. URL: https://www.surveymonkey.com/mp/free-vs-paid-plans/

3. Guay M., Klosowski T. The best online survey apps in 2025. *Zapier*. URL: https://zapier.com/blog/best-survey-apps/

4. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design. — 5th ed. — Boston: Addison-Wesley, 2012. — 1065 p.

5. Massé M. REST API Design Rulebook. O'Reilly Media, 2011. 116 p.

6. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF. URL: https://datatracker.ietf.org/doc/html/rfc7519

7. Радівілова Т., Кіріченко Л., Пантєлєєв В., Мазепа А., Білодід В. Аналіз методів автентифікації для вебзастосунків та реалізація вебзастосунку з інтегрованою системою автентифікації. *ITTSI Journal*. URL: https://www.itssi-journal.com/index.php/ittsi/article/view/509

8. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Pearson, 2020. 800 p.

9. Grassi P. A. et al. Digital Identity Guidelines : NIST Special Publication 800-63B. National Institute of Standards and Technology, 2017. URL: https://doi.org/10.6028/NIST.SP.800-63b

10. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. IETF, 2015. URL: https://tools.ietf.org/html/rfc7519

11. React. The library for web and native user interfaces. *Meta Platforms*. URL: https://react.dev/

12. Node.js Documentation. *OpenJS Foundation*. URL: https://nodejs.org/docs/latest/api/

13. Express: Fast, unopinionated, minimalist web framework for Node.js. *OpenJS Foundation*. URL: https://expressjs.com/

14. MongoDB Manual. *MongoDB, Inc*. URL: https://www.mongodb.com/docs/

15. Vite: Next Generation Frontend Tooling. *Vite*. URL: https://vitejs.dev/guide/

16. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML. *Tailwind Labs*. URL: https://tailwindcss.com/docs

17. Микитенко В. І., Парфьонова Т.О. Порівняння технологій для розробки веб-сервісів опитувань: Node.js проти альтернатив. Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті : матеріали XLVIII Міжнар. наук. студ. конф. (м. Полтава, 10 квіт. 2025 р.). Полтава : ПУЕТ, 2025. С. 263–265.

18. Микитенко В. І., Парфьонова Т.О. Інтеграція AI-технологій для автоматизації створення та аналізу користувацьких опитувань: підхід на основі Google Gemini API. Наука і молодь в XXI сторіччі : матеріали XI Міжнар. наук.-практ. конф. (м. Полтава, 10 листоп. 2025 р.). Полтава : ПУЕТ, 2025.

19. Zustand Documentation. *Poimandres*. URL: https://zustand.docs.pmnd.rs/getting-started/introduction

20. Gemini API Documentation. *Google AI for Developers*. URL: https://ai.google.dev/gemini-api/docs?hl=ua

Додаток А

А.1 - Схема опитування з вкладеними питаннями та skip logic

```
const mongoose = require('mongoose');
// Схема для питання
const questionSchema = new mongoose.Schema({
  text: {
    type: String,
    required: [true, 'Текст питання обов\'язковий'],
    trim: true
  },
  type: {
    type: String,
    enum: ['radio', 'checkbox', 'text', 'textarea', 'rating'],
    required: [true, 'Тип питання обов\'язковий']
  },
  options: [{
    text: String,
    value: String
  }],
  required: {
    type: Boolean,
    default: false
  },
  order: {
    type: Number,
    required: true
  },
  // Skip Logic - умовна логіка
  skipLogic: {
    enabled: {
      type: Boolean,
      default: false
    },
```

```
          condition: {
            questionId: {
              type: String  // ID питання на яке дивимось (після конвертації)
            },
            questionIndex: {
                 type: Number  // Індекс питання (тимчасово, для конвертації в
questionId)
            },
            operator: {
              type: String,
                   enum: ['equals', 'not_equals', 'contains', 'not_contains',
'is_answered'],
              default: 'equals'
            },
                  value: mongoose.Schema.Types.Mixed  // Очікуване значення
(String/Number/Array)
          }
        }
      });
      // Основна схема опитування
      const surveySchema = new mongoose.Schema({
        title: {
          type: String,
          required: [true, 'Назва опитування обов\'язкова'],
          trim: true,
          maxlength: [200, 'Назва не може бути довшою за 200 символів']
        },
        description: {
          type: String,
          trim: true,
          maxlength: [1000, 'Опис не може бути довшим за 1000 символів']
        },
        questions: [questionSchema],

        creator: {
          type: mongoose.Schema.Types.ObjectId,
```

```
  ref: 'User',
  required: true
},

collaborators: [{
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  addedAt: {
    type: Date,
    default: Date.now
  }
}],
uniqueLink: {
  type: String,
  unique: true,
  required: true
},

isActive: {
  type: Boolean,
  default: true
},

closingDate: {
  type: Date
},

allowMultipleResponses: {
  type: Boolean,
  default: false
},

participantLimit: {
```

```
    type: Number,
    default: null, // null означає необмежено
    min: [1, 'Мінімальна кількість учасників - 1']
  },
  createdAt: {
    type: Date,
    default: Date.now
  },

  updatedAt: {
    type: Date,
    default: Date.now
  }
});
// Автоматичне оновлення дати модифікації
surveySchema.pre('save', function(next) {
  this.updatedAt = Date.now();
  next();
});
// Генерація унікального посилання
surveySchema.methods.generateUniqueLink = function() {
  const randomString = Math.random().toString(36).substring(2, 15) +
                       Math.random().toString(36).substring(2, 15);
  this.uniqueLink = randomString;
  return this.uniqueLink;
};

module.exports = mongoose.model('Survey', surveySchema);
```

А.2 Реалізація інтеграції сервісу з генеративними моделями штучного інтелекту

```
const { HfInference } = require('@huggingface/inference');
const { GoogleGenerativeAI } = require('@google/generative-ai');

class AIService {
  constructor() {
    console.log('Ініціалізація AI Service...');
                        console.log('          GOOGLE_AI_API_KEY          exists:',
!!process.env.GOOGLE_AI_API_KEY);
                       console.log('         HUGGINGFACE_API_KEY         exists:',
!!process.env.HUGGINGFACE_API_KEY);

    this.hf = process.env.HUGGINGFACE_API_KEY
      ? new HfInference(process.env.HUGGINGFACE_API_KEY)
      : null;

    this.gemini = process.env.GOOGLE_AI_API_KEY
      ? new GoogleGenerativeAI(process.env.GOOGLE_AI_API_KEY)
      : null;

    this.providers = [
      { name: 'gemini', enabled: !!this.gemini },
      { name: 'huggingface', enabled: !!this.hf },
      { name: 'fallback', enabled: true }
    ];

    console.log('AI Service ініціалізовано. Провайдери:', this.providers);
  }

  async generateSurvey(params) {
     const { topic, goal, questionCount = 7, questionTypes = ['radio',
'checkbox', 'text'], additionalInstructions } = params;

    console.log(`Генерую опитування: "${topic}" (${questionCount} питань)`);
```

```
        if (additionalInstructions) {
          console.log(`Додаткові інструкції: ${additionalInstructions}`);
        }

        for (const provider of this.providers) {
          if (!provider.enabled) continue;

          try {
              const result = await this[`generate_${provider.name}`](topic, goal, questionCount, questionTypes, additionalInstructions);
            console.log(`Успішно згенеровано через ${provider.name}`);
            return result;
          } catch (error) {
            console.log(` ${provider.name} failed:`, error.message);
          }
        }

        throw new Error('Всі AI провайдери недоступні');
      }

       async generate_gemini(topic, goal, questionCount, questionTypes, additionalInstructions) {
        if (!this.gemini) throw new Error('Google Gemini не налаштовано');

        const model = this.gemini.getGenerativeModel({
          model: 'gemini-2.5-flash'
        });

        const prompt = `Створи опитування українською мовою на тему "${topic}".
    Ціль: ${goal || 'збір відгуків'}
    Кількість питань: ${questionCount}
    ${additionalInstructions ? `\nДодаткові вимоги: ${additionalInstructions}` : ''}

    ВАЖЛИВО: Відповідай ТІЛЬКИ у форматі JSON. Не додавай жодного тексту до або після JSON. Не використовуй markdown code blocks.
```

```
Формат відповіді:
{
  "title": "Назва опитування",
  "description": "Короткий опис (1-2 речення)",
  "questions": [
    {
      "text": "Текст питання",
      "type": "radio",
      "options": [{"text": "Варіант 1", "value": "value1"}],
      "required": true
    }
  ]
}

Типи питань для використання: ${questionTypes.join(', ')}
- radio: питання з одним варіантом відповіді (додай 4-5 варіантів)
- checkbox: питання з декількома варіантами (додай 4-6 варіантів)
- text: коротка текстова відповідь (options: [])
- textarea: довга текстова відповідь (options: [])

Створи ${questionCount} різноманітних, професійних питань українською мовою.
Розподіли типи питань рівномірно.
Питання повинні бути чіткими, зрозумілими та релевантними темі.${additionalInstructions ? `\n\nОБОВ'ЯЗКОВО враховуй додаткові вимоги користувача!` : ''}`;

    const result = await model.generateContent(prompt);
    const response = await result.response;
    const text = response.text();

    return this.parseAIResponse(text, questionTypes, 'gemini');
  }

  async generate_huggingface(topic, goal, questionCount, questionTypes, additionalInstructions) {
```

```
        if (!this.hf) throw new Error('Hugging Face не налаштовано');

            const prompt = this.buildPrompt(topic, goal, questionCount, questionTypes);

        const response = await this.hf.textGeneration({
          model: 'mistralai/Mistral-7B-Instruct-v0.2',
          inputs: prompt,
          parameters: {
            max_new_tokens: 2000,
            temperature: 0.7,
            top_p: 0.95,
            return_full_text: false
          }
        });

            return this.parseAIResponse(response.generated_text, questionTypes, 'huggingface');
      }

        async generate_fallback(topic, goal, questionCount, questionTypes, additionalInstructions) {
        console.log('Використовую fallback генерацію з шаблонами');

        const templates = this.getQuestionTemplates(topic, goal);
        const questions = [];

            const typeCounts = this.distributeQuestionTypes(questionCount, questionTypes);

        for (const [type, count] of Object.entries(typeCounts)) {
          const typeTemplates = templates[type] || [];

          for (let i = 0; i < count && i < typeTemplates.length; i++) {
            questions.push({
              text: typeTemplates[i].text,
```

```
            type: type,
            options: typeTemplates[i].options || [],
            required: true,
            order: questions.length + 1
          });
        }
      }

      return {
        title: `Опитування: ${topic}`,
         description: `Дякуємо за участь в опитуванні. Ваша думка дуже важлива для нас!`,
        questions: questions.slice(0, questionCount),
        aiGenerated: true,
        aiProvider: 'fallback'
      };
    }

    async improveQuestion(questionText) {
      try {
        if (this.hf) {
           const prompt = `Покращ це питання для опитування, зроби його більш чітким та професійним. Відповідай ТІЛЬКИ покращеним питанням без додаткових пояснень:\n\n"${questionText}"`;

          const response = await this.hf.textGeneration({
            model: 'mistralai/Mistral-7B-Instruct-v0.2',
            inputs: prompt,
            parameters: {
              max_new_tokens: 150,
              temperature: 0.5
            }
          });

          const improved = response.generated_text.trim().replace(/^["']|["']$/g, '');
```

```
        return { improved, original: questionText };
      }
    } catch (error) {
      console.log('Не вдалося покращити питання:', error.message);
    }

    return {
      improved: this.basicImprove(questionText),
      original: questionText
    };
  }
  buildPrompt(topic, goal, questionCount, questionTypes) {
    return `Створи опитування українською мовою на тему "${topic}".
Ціль: ${goal || 'збір відгуків'}
Кількість питань: ${questionCount}
Типи питань: ${questionTypes.join(', ')}
Формат відповіді (JSON):
{
  "title": "Назва опитування",
  "description": "Короткий опис (1-2 речення)",
  "questions": [
    {
      "text": "Текст питання",
      "type": "radio",
      "options": [{"text": "Варіант 1", "value": "variant1"}],
      "required": true
    }
  ]
}

Створи ТІЛЬКИ JSON без додаткового тексту:`;
  }

  parseAIResponse(text, questionTypes, provider = 'huggingface') {
    try {
```

```
            // Очищаємо текст від markdown code blocks
            let cleanText = text.trim();
            cleanText = cleanText.replace(/```json\s*/g, '').replace(/```\s*/g, '');

            // Спроба знайти JSON в тексті
            const jsonMatch = cleanText.match(/\{[\s\S]*\}/);
            if (jsonMatch) {
              const parsed = JSON.parse(jsonMatch[0]);
              // Перевірка чи є необхідні поля
              if (!parsed.questions || !Array.isArray(parsed.questions)) {
                throw new Error('Відповідь не містить масив питань');
              }
              // Додаємо order до питань
              parsed.questions = parsed.questions.map((q, index) => ({
                ...q,
                order: index + 1,
                required: q.required !== undefined ? q.required : true
              }));

              parsed.aiGenerated = true;
              parsed.aiProvider = provider;

               console.log(`Успішно розпарсено ${parsed.questions.length} питань від
${provider}`);

              return parsed;
            }
          } catch (error) {
            console.log('Помилка парсингу AI відповіді:', error.message);
              console.log(' Відповідь AI (перші 200 символів):', text.substring(0,
200));
          }

          throw new Error('Не вдалося розібрати відповідь AI');
        }
```

```
basicImprove(text) {
  if (!text.trim().endsWith('?')) {
    text = text.trim() + '?';
  }
  text = text.charAt(0).toUpperCase() + text.slice(1);

  return text;
}

distributeQuestionTypes(total, types) {
  const distribution = {};
  const baseCount = Math.floor(total / types.length);
  const remainder = total % types.length;

  types.forEach((type, index) => {
    distribution[type] = baseCount + (index < remainder ? 1 : 0);
  });

  return distribution;
}

getQuestionTemplates(topic, goal) {
  return {
    radio: [
      {
        text: 'Як би ви оцінили загальну якість за шкалою від 1 до 5?',
        options: [
          { text: 'Відмінно (5)', value: '5' },
          { text: 'Добре (4)', value: '4' },
          { text: 'Задовільно (3)', value: '3' },
          { text: 'Погано (2)', value: '2' },
          { text: 'Дуже погано (1)', value: '1' }
        ]
      },
```

```
{
  text: 'Наскільки ви задоволені загалом?',
  options: [
    { text: 'Повністю задоволений', value: 'fully_satisfied' },
    { text: 'Скоріше задоволений', value: 'satisfied' },
    { text: 'Нейтрально', value: 'neutral' },
    { text: 'Скоріше незадоволений', value: 'dissatisfied' },
    { text: 'Повністю незадоволений', value: 'fully_dissatisfied' }
  ]
},
{
  text: 'Чи порекомендуєте ви нас іншим?',
  options: [
    { text: 'Обов\'язково порекомендую', value: 'definitely' },
    { text: 'Швидше за все порекомендую', value: 'probably' },
    { text: 'Не впевнений', value: 'not_sure' },
    { text: 'Швидше за все ні', value: 'probably_not' },
    { text: 'Точно ні', value: 'definitely_not' }
  ]
},
{
  text: 'Як часто ви користуєтесь нашими послугами?',
  options: [
    { text: 'Щодня', value: 'daily' },
    { text: 'Кілька разів на тиждень', value: 'weekly' },
    { text: 'Раз на місяць', value: 'monthly' },
    { text: 'Рідше ніж раз на місяць', value: 'rarely' },
    { text: 'Вперше', value: 'first_time' }
  ]
},
{
  text: 'Чи відповідає якість вашим очікуванням?',
  options: [
    { text: 'Перевищує очікування', value: 'exceeds' },
    { text: 'Відповідає очікуванням', value: 'meets' },
```

```
      { text: 'Частково відповідає', value: 'partially' },
      { text: 'Не відповідає', value: 'not_meets' }
    ]
  }
],
checkbox: [
  {
    text: 'Які аспекти вам найбільше сподобалися?',
    options: [
      { text: 'Якість обслуговування', value: 'quality' },
      { text: 'Швидкість роботи', value: 'speed' },
      { text: 'Зручність використання', value: 'convenience' },
      { text: 'Співвідношення ціна/якість', value: 'price' },
      { text: 'Професіоналізм', value: 'professionalism' },
      { text: 'Технічна підтримка', value: 'support' }
    ]
  },
  {
    text: 'Що потребує покращення на вашу думку?',
    options: [
      { text: 'Функціональність', value: 'functionality' },
      { text: 'Зручність інтерфейсу', value: 'interface' },
      { text: 'Швидкість роботи', value: 'performance' },
      { text: 'Документація', value: 'documentation' },
      { text: 'Служба підтримки', value: 'support' },
      { text: 'Ціноутворення', value: 'pricing' }
    ]
  },
  {
    text: 'Які додаткові функції ви б хотіли бачити?',
    options: [
      { text: 'Мобільний додаток', value: 'mobile_app' },
      { text: 'Інтеграції з іншими сервісами', value: 'integrations' },
      { text: 'Розширена аналітика', value: 'analytics' },
      { text: 'Налаштування під себе', value: 'customization' },
```

```
      { text: 'Багатомовність', value: 'multilingual' }
    ]
  }
],
text: [
  {
    text: 'Що вам найбільше сподобалося?',
    options: []
  },
  {
    text: 'Які покращення ви б хотіли бачити?',
    options: []
  },
  {
    text: 'Поділіться вашою думкою або враженнями',
    options: []
  },
  {
    text: 'Чого не вистачає для ідеального досвіду?',
    options: []
  },
  {
    text: 'Що найбільше запам\'яталося під час використання?',
    options: []
  }
],
textarea: [
  {
    text: 'Будь ласка, поділіться детальним відгуком про ваш досвід',
    options: []
  },
  {
    text: 'Які у вас є пропозиції щодо покращення?',
    options: []
  },
```

```
            {
              text: 'Опишіть ваш досвід використання максимально детально',
              options: []
            }
          ]
        };
      }

      async analyzeResponses(surveyData) {
        console.log(`Аналізую відповіді для опитування: "${surveyData.title}"`);

        // Перевірка наявності Gemini
        if (!this.gemini) {
          throw new Error('Google Gemini не налаштовано. AI аналіз недоступний.');
        }

        // Перевірка наявності відповідей
        if (!surveyData.responses || surveyData.responses.length === 0) {
          throw new Error('Недостатньо відповідей для аналізу. Потрібно мінімум 1 відповідь.');
        }

        try {
          const model = this.gemini.getGenerativeModel({
            model: 'gemini-2.5-flash'
          });

          // Підготовка даних для аналізу
          const analysisData = this.prepareDataForAnalysis(surveyData);

          const prompt = `Ти експерт з аналізу даних опитувань. Проаналізуй результати опитування та надай детальний звіт українською мовою.

📋 ОПИТУВАННЯ: "${surveyData.title}"

📊 Кількість відповідей: ${surveyData.responses.length}
```

🗓️ Період: ${new Date(surveyData.createdAt).toLocaleDateString('uk-UA')}

${analysisData}

Твоє завдання:
1. Проаналізувати всі відповіді респондентів
2. Виявити ключові тренди та патерни
3. Визначити загальний настрій (позитивний/нейтральний/негативний)
4. Знайти найчастіші теми та згадування
5. Надати конкретні, дієві рекомендації

ФОРМАТ ВІДПОВІДІ (дотримуйся точно):

🎯 ЗАГАЛЬНИЙ ВИСНОВОК
[2-3 речення про основний результат опитування]

📊 СТАТИСТИЧНІ ІНСАЙТИ
- [Конкретний факт з цифрами]
- [Конкретний факт з цифрами]
- [Конкретний факт з цифрами]

💭 КЛЮЧОВІ ТЕМИ
- [Тема 1]: [короткий опис]
- [Тема 2]: [короткий опис]
- [Тема 3]: [короткий опис]

😊 НАСТРІЙ РЕСПОНДЕНТІВ
[Визнач загальний настрій: позитивний/нейтральний/негативний з обґрунтуванням]

💡 РЕКОМЕНДАЦІЇ
1. [Конкретна дієва рекомендація]
2. [Конкретна дієва рекомендація]
3. [Конкретна дієва рекомендація]

```
⚠ ЗОНИ УВАГИ
- [Що потребує негайної уваги]
- [Потенційні проблеми]

Будь конкретним, використовуй цифри та факти з даних. Пиши професійно, але зрозуміло.`;

            const result = await model.generateContent(prompt);
            const response = await result.response;
            const analysisText = response.text();

            console.log('AI аналіз завершено успішно');

            return {
              analysis: analysisText,
              metadata: {
                surveyTitle: surveyData.title,
                totalResponses: surveyData.responses.length,
                analyzedAt: new Date().toISOString(),
                aiProvider: 'gemini-2.5-flash'
              }
            };

          } catch (error) {
            console.error('Помилка AI аналізу:', error.message);
            throw new Error(`Не вдалося виконати AI аналіз: ${error.message}`);
          }
        }

        prepareDataForAnalysis(surveyData) {
          let analysisText = '📝 ДАНІ ДЛЯ АНАЛІЗУ:\n\n';

          surveyData.questionStats.forEach((question, index) => {
            analysisText += `ПИТАННЯ ${index + 1}: ${question.text}\n`;
            analysisText += `Тип: ${question.type}\n`;
```

```
            if (question.type === 'text' || question.type === 'textarea') {
              // Текстові відповіді
              analysisText += 'Відповіді респондентів:\n';
              const answers = Object.keys(question.answers);
              answers.forEach((answer, idx) => {
                const count = question.answers[answer];
                if (count > 1) {
                  analysisText += `- "${answer}" (згадано ${count} разів)\n`;
                } else {
                  analysisText += `- "${answer}"\n`;
                }
              });
            } else {
              // Варіанти вибору (radio/checkbox)
              analysisText += 'Розподіл відповідей:\n';
              Object.entries(question.answers).forEach(([answer, count]) => {
                    const percentage = ((count / surveyData.responses.length) *
100).toFixed(1);
                        analysisText += `- "${answer}": ${count} відповідей
(${percentage}%)\n`;
              });
            }

            analysisText += '\n';
          });

          return analysisText;
        }

        /**
         * Чат про результати опитування
         * @param {Object} surveyContext - Контекст опитування (назва, статистика)
         * @param {String} userMessage - Повідомлення користувача
         * @param {Array} history - Історія розмови
```

```
   * @returns {Promise<String>} - Відповідь AI
   */
  async chatAboutSurvey(surveyContext, userMessage, history = []) {
    console.log('AI Chat: Обробка повідомлення');
    if (!this.gemini) {
      throw new Error('AI сервіс недоступний');
    }

    // Формування контексту про опитування
    const contextInfo = `ОПИТУВАННЯ: "${surveyContext.title}"
КІЛЬКІСТЬ ВІДПОВІДЕЙ: ${surveyContext.totalResponses}

СТАТИСТИКА ПО ПИТАННЯХ:
${surveyContext.questionStats.map((q, i) => {
  const answersText = Object.entries(q.answers)
    .map(([answer, count]) => `  - "${answer}": ${count} відповідей`)
    .join('\n');
  return `${i + 1}. ${q.text} (${q.type})\n${answersText}`;
}).join('\n\n')}`;
    // Формування повного промпта з історією
    let conversationHistory = '';
    if (history && history.length > 1) {
      // Пропускаємо перше повідомлення (початковий аналіз)
      conversationHistory = '\n\nПОПЕРЕДНЯ РОЗМОВА:\n';
      for (let i = 1; i < history.length; i++) {
        const msg = history[i];
        if (msg.role === 'user') {
          conversationHistory += `Користувач: ${msg.content}\n`;
        } else if (msg.role === 'assistant') {
          conversationHistory += `AI: ${msg.content}\n`;
        }
      }
    }
```

```
          const prompt = `Ти - AI асистент, який допомагає аналізувати результати опитування.

      ${contextInfo}

      ${conversationHistory}

      Користувач запитує: ${userMessage}

      Відповідай українською мовою. Будь корисним, конкретним та професійним. Використовуй дані зі статистики вище для відповіді. Якщо питання про кількість - дай точну цифру з даних.`;


          try {
            const model = this.gemini.getGenerativeModel({
              model: 'gemini-2.5-flash'
            });
            const result = await model.generateContent(prompt);
            const response = await result.response;
            const responseText = response.text();


            return responseText;
          } catch (error) {
            console.error('Помилка Gemini chat:', error);
            throw new Error('Не вдалося отримати відповідь від AI');
          }
        }
      }
      module.exports = new AIService();
```