Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника»

Виконала ст. гр. КН-22-1 _____ Перепелиця А. Р.

Керівник _____ Тронь В. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-1 *Перепелиці Аріни Русланівни*

1. Тема кваліфікаційної роботи: *«Розробка інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника»*

затверджено наказом по університету № *254с* від *13.05.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 90с., додатки, презентація у Microsoft PowerPoint (12 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Тронь В. В.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.26*
2	*Розділ 1*	*05.04.26*
3	*Розділ 2*	*01.05.26*
4	*Висновки*	*25.05.26*
5	*Оформлення кваліфікаційної роботи*	*28.05.26*
6	*Підготовка презентації та графічного матеріалу*	*20.05.26*
7	*Підготовка доповіді до захисту*	*10.06.26*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Тронь В. В./

7. Запевнення: *Я, Перепелиця Аріна Русланівна, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студентка _____ / Перепелиця А. Р./

АНОТАЦІЯ

Перепелиця А. Р. Розробка інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника: кваліфікаційна робота бакалавра: 122 – Комп'ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 90 с.

Актуальність теми полягає у необхідності розробки сучасних програмних рішень, що поєднують можливості інформаційних технологій та штучного інтелекту з метою покращення якості освіти. Використання інтелектуальних систем дозволяє забезпечити індивідуалізовану підтримку, зробити навчання більш інтерактивним та спростити доступ до знань.

Метою роботи є розробка інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника, яка забезпечує ефективну взаємодію з користувачем, сприяє кращому засвоєнню навчального матеріалу та дозволяє здійснювати перевірку знань.

У першому розділі було розглянуто поняття інформаційних систем у сфері освіти, здійснено аналіз існуючих інтелектуальних систем підтримки навчання. Визначено проблеми сучасного навчання, такі як відсутність персоналізації та обмежена інтерактивність, і сформульовано постановку задачі на розробку.

У другому розділі розглядаються питання проєктування загальної архітектури системи, здійснено вибір технологій (React.js, Supabase, Groq API) та спроєктовано логічну модель бази даних. Окрім цього, наведено програмну реалізацію розробленої системи (чат-інтерфейс, підключення до LLM API, система авторизації) та результати її тестування, які підтвердили стабільність роботи, високу швидкість взаємодії та зручність користувацького інтерфейсу.

Ключові слова:

ІНФОРМАЦІЙНА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ, AI-ПОМІЧНИК, ВЕБЗАСТОСУНОК, REACT, SUPABASE, ВЕЛИКА МОВНА МОДЕЛЬ, ГЕНЕРАЦІЯ ТЕСТІВ.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій відбувається глибока трансформація всіх сфер суспільного життя, зокрема освітньої галузі. Впровадження цифрових рішень у навчальний процес сприяє підвищенню його ефективності, доступності та гнучкості. Освіта поступово переходить від традиційних, стандартизованих форм подачі матеріалу до інтерактивних, орієнтованих на потреби конкретного користувача підходів. У цьому контексті особливої актуальності набувають інформаційні системи, що забезпечують комплексну підтримку навчання з використанням найсучасніших інноваційних технологій.

Одним із ключових і найбільш перспективних напрямків розвитку освітніх технологій є застосування систем штучного інтелекту (ШІ). Завдяки прориву в галузі машинного навчання та появі великих мовних моделей (Large Language Models), штучний інтелект вийшов на новий рівень взаємодії з людиною. Такі системи дозволяють автоматизувати процес обробки великих масивів інформації, семантично аналізувати запити користувачів, адаптувати навчальний матеріал та динамічно формувати індивідуальні рекомендації. Використання інтелектуальних помічників у навчанні відкриває безпрецедентні можливості для студентів, оскільки вони можуть отримувати детальні пояснення складних тем у зручному діалоговому форматі, ставити запитання в режимі реального часу та отримувати миттєвий, розгорнутий зворотний зв’язок.

Сучасні здобувачі освіти, зокрема студенти ІТ-спеціальностей, щоденно стикаються з величезною кількістю нової інформації, яку необхідно швидко опрацьовувати, розуміти та застосовувати на практиці. При цьому традиційні методи навчання та існуючі системи управління навчанням (LMS) не завжди дозволяють ефективно адаптувати навчальний контент під індивідуальні особливості, темп засвоєння та поточний рівень підготовки кожного окремого студента. Відсутність персоналізованого підходу часто призводить до труднощів у

розумінні складних технічних тем, втрати мотивації до навчання та, як наслідок, зниження загальної академічної успішності.

Саме тому виникає гостра потреба у створенні новітніх інформаційних інструментів — віртуальних AI-помічників, які б виконували роль персонального тьютора. Такий помічник здатний працювати цілодобово, надавати підказки, генерувати приклади програмного коду, пояснювати помилки та автоматично створювати тести для перевірки знань.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю розробки та впровадження сучасних програмних рішень, що поєднують передові можливості веброзробки та хмарних API великих мовних моделей для подолання недоліків традиційних освітніх систем. Створення інформаційної системи на основі AI-помічника є своєчасним і затребуваним кроком на шляху до цифровізації та індивідуалізації вищої освіти.

Метою кваліфікаційної роботи є розробка інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника, яка забезпечує ефективну, швидку та персоналізовану взаємодію зі студентом, сприяє кращому засвоєнню навчального матеріалу та дозволяє здійснювати автоматизовану перевірку набутих знань.

Для досягнення поставленої мети було сформульовано та вирішено такі *основні завдання:*

1. Здійснити аналіз предметної області, дослідити поняття інформаційних систем у сфері освіти та визначити основні проблеми сучасного навчання;
2. Провести огляд існуючих інтелектуальних систем підтримки навчання та визначити їхні переваги і недоліки;
3. Спроєктувати загальну архітектуру розроблюваної інформаційної системи та схему взаємодії її компонентів;
4. Обґрунтувати вибір сучасних технологій для програмної реалізації (react.js, supabase, groq api);
5. Розробити логічну модель бази даних для збереження користувацької інформації, історії діалогів та результатів тестування;

6. Розробити алгоритми та логіку роботи аі-помічника, включно з механізмом потокової генерації відповідей (streaming) та режимом генерації тестів;

7. Здійснити програмно-технічну реалізацію клієнтського вебзастосунку, забезпечивши інтуїтивно зрозумілий та зручний інтерфейс користувача;

8. Провести комплексне тестування розробленої системи для перевірки її стабільності, швидкодії та коректності виконання всіх закладених функцій.

Об'єктом дослідження є процес інформаційної та інтелектуальної підтримки навчання користувачів у сучасному цифровому освітньому середовищі.

Предметом дослідження є методи, алгоритми та програмно-технічні засоби розробки інформаційної системи на базі технологій штучного інтелекту та великих мовних моделей.

Методи дослідження. Під час виконання роботи використовувалися методи системного аналізу (для дослідження предметної області та проєктування архітектури системи); методи об'єктно-орієнтованого програмування (для розробки програмного коду клієнтського застосунку); методи проєктування реляційних баз даних (для розробки структури збереження даних у Supabase); методи тестування програмного забезпечення (для перевірки працездатності готового продукту).

Практичне значення одержаних результатів полягає в тому, що розроблена інформаційна система є повністю готовим до використання програмним продуктом. Запропонований AI-помічник може бути імплементований у навчальний процес закладів вищої освіти як додатковий інструмент для самостійної роботи студентів, що дозволить значно знизити рутинне навантаження на викладачів, підвищити інтерес здобувачів до навчання та покращити загальні показники засвоєння складних дисциплін.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, двох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг пояснювальної записки становить 90 сторінок основного тексту, який ілюстровано рисунками та таблицями.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття інформаційних систем у сфері освіти

У сучасному інформаційному суспільстві поняття інформаційної системи (ІС) у сфері освіти зазнало кардинальних змін [5]. Якщо раніше під освітніми інформаційними системами розуміли переважно статичні бази даних або класичні системи управління навчанням (Learning Management Systems, LMS), основна мета яких полягала у кабінетному адмініструванні та зберіганні матеріалів, то сьогодні фокус остаточно змістився в бік інтерактивності та персоналізації. Сучасна інформаційна система в освіті — це комплексне програмно-технічне середовище, здатне не лише надавати доступ до навчального контенту, а й активно взаємодіяти зі здобувачем освіти, динамічно адаптуючись до його індивідуальних потреб, рівня поточної підготовки та специфіки сприйняття матеріалу [21].

Особливо гостро потреба в модернізації інформаційних систем відчувається у сфері технічної освіти, зокрема при вивченні прикладних дисциплін галузі «Комп'ютерні науки». Традиційні платформи, такі як Moodle чи Canvas, добре справляються з організаційними завданнями, проте з точки зору самого процесу навчання вони залишаються пасивними інструментами [22]. Вони не здатні в режимі реального часу проаналізувати написаний студентом програмний код, пояснити логічну помилку в побудові алгоритму або згенерувати унікальне практичне завдання на основі тих тем, які викликають найбільше труднощів. У таких системах студент залишається сам на сам із статичним текстом або відеолекцією, що суттєво знижує ефективність засвоєння складних інженерних концепцій та призводить до швидкої втрати мотивації.

Саме тому на зміну класичним LMS приходить нове покоління програмних рішень — інформаційні системи інтелектуальної підтримки навчання (ІСПН). Їхньою ключовою відмінністю є імплементація методів штучного інтелекту, зокрема інтеграція великих мовних моделей (Large Language Models, LLM), для

глибокої семантичної обробки природної мови та генерації контекстно-залежних відповідей [8]. Інтелектуальна освітня система здатна імітувати діяльність живого викладача (ментора): вона аналізує контекст запиту студента, формує релевантну пояснювальну відповідь, наводить фрагменти коду та здійснює автоматизовану перевірку знань. У такому цифровому середовищі студент перетворюється з пасивного слухача на активного учасника безперервного евристичного діалогу.

Розроблювана в межах даної кваліфікаційної роботи система є яскравим представником цього сучасного класу інтелектуальних освітніх платформ. Замість створення чергового сховища навчальних файлів, було спроєктовано динамічний вебзастосунок, архітектурним ядром якого виступає віртуальний AI-помічник. Ця система докорінно трансформує поняття освітньої підтримки: вона функціонує як персональний тьютор, доступний у будь-який час. Використовуючи обчислювальні потужності зовнішнього хмарного сервісу Groq API та передові моделі лінійки LLaMA, розроблена система здатна миттєво генерувати розгорнуті пояснення з програмування, суворо дотримуючись заданого педагогічного стилю завдяки гнучко налаштованому системному промпту (system prompt) [10].

Важливою складовою функціонування сучасних інтелектуальних систем є забезпечення ергономічності та безперервності взаємодії. У розробленій системі це реалізовано через механізм потокової передачі даних (streaming API) під час генерації відповідей штучним інтелектом. Завдяки цьому студент не чекає завершення обробки всього масиву тексту, а отримує інформацію поступово (токенами), що створює психологічний ефект живого спілкування [19]. Крім того, невід'ємним атрибутом повноцінної освітньої ІС є модуль контролю та верифікації знань. Замість використання заздалегідь підготовлених статичних банку питань, розроблена система наділена здатністю генерувати унікальні багатоваріантні тестові завдання (у машиночитаному форматі JSON) безпосередньо за текстовим запитом користувача на конкретну тему. Це інноваційне рішення робить процес самоперевірки адаптивним і повністю нівелює проблему механічного заучування правильних відповідей.

З технічної та архітектурної точок зору, еволюція поняття інформаційної системи в освіті також відображається у масовому переході від монолітних серверних структур на користь гнучких хмарних та мікросервісних рішень [21]. Розроблений AI-помічник базується на клієнт-серверній архітектурі Single Page Application (за допомогою React.js) та використовує сучасну інфраструктуру Backend-as-a-Service (на базі Supabase) для автентифікації студентів і надійного, ізольованого збереження історії навчальних діалогів та результатів тестування у базі даних PostgreSQL [11, 12]. Такий архітектурний підхід забезпечує абсолютну конфіденційність персональних даних, високу швидкість відгуку інтерфейсу та легкість горизонтального масштабування платформи.

Отже, поняття інформаційної системи у сфері освіти сьогодні нерозривно пов'язане з технологіями штучного інтелекту, хмарними обчисленнями та адаптивними алгоритмами. Розроблювана інформаційна система інтелектуальної підтримки навчання повністю відповідає цій новій академічній парадигмі. Вона успішно інтегрує передові можливості мовних моделей у зручний, захищений та орієнтований на здобувача користувацький інтерфейс, перетворюючи процес засвоєння складних технічних дисциплін із пасивного споживання інформації на активний, персоналізований та глибоко інтерактивний дослідницький досвід.

1.2 Інтелектуальні системи підтримки навчання

Еволюція освітніх технологій пройшла довгий шлях від простих електронних бібліотек до складних алгоритмічних середовищ. Проте справжній парадигмальний зсув у цій галузі відбувся з появою інтелектуальних систем підтримки навчання (ІСПН) нового покоління, які базуються на технологіях машинного навчання (Machine Learning) та обробки природної мови (Natural Language Processing). Традиційні експертні системи, які використовувалися в освіті раніше, працювали на основі жорстких дерев рішень і попередньо запрограмованих правил [6]. Вони могли реагувати лише на обмежений набір тригерів і були абсолютно безпорадними, коли студент формулював запит нестандартно або робив комплексну

помилку в програмному коді. Натомість сучасні інтелектуальні системи здатні до глибокого семантичного аналізу: вони розуміють контекст, виявляють приховані закономірності в помилках користувача та генерують унікальний навчальний контент у режимі реального часу [16].

Фундаментом для створення сучасних віртуальних тьюторів стали великі мовні моделі (Large Language Models, LLM), побудовані на архітектурі трансформерів [9]. Завдяки механізмам уваги (attention mechanisms), ці моделі здатні утримувати в пам'яті довгі контексти діалогів, що є критично важливим для навчального процесу [8]. Навчання не відбувається за один запит — це завжди ітеративний процес, у якому студент ставить уточнюючі запитання, просить навести інші приклади або пояснити незрозумілі фрагменти. Інтелектуальна система, що розробляється у даній кваліфікаційній роботі, використовує саме такі передові моделі (зокрема лінійку LLaMA 3 через інтеграцію з Groq API), що дозволяє їй вести зв'язний, багатокроковий академічний діалог, імітуючи індивідуальну роботу з кваліфікованим викладачем [10].

Специфіка підготовки фахівців у галузі комп'ютерних наук вимагає від інтелектуальної системи не лише лінгвістичних, але й високих алгоритмічних компетенцій [22]. Загальнодоступні чат-боти часто видають прямі відповіді або готові блоки коду, що стимулює студентів до бездумного копіювання (так званого "copy-paste" програмування) і шкодить процесу формування інженерного мислення [7]. Щоб вирішити цю проблему, у розробленій системі застосовано метод інженерії промптів (Prompt Engineering). За допомогою жорстко заданого системного промпта (system prompt), мовній моделі делегується роль "навчального ментора" [19]. Це означає, що AI-помічник свідомо уникає надання готових розв'язків для лабораторних чи практичних робіт. Натомість він використовує метод Сократа: ставить навідні запитання, вказує на логічні хиби у міркуваннях студента, пояснює концепції на абстрактних прикладах та спонукає користувача самостійно дійти до правильного рішення.

Окрім діалогової підтримки, критично важливою функцією розробленої інтелектуальної системи є здатність працювати з технічним форматуванням.

Система вміє розпізнавати програмний код, переданий користувачем, проводити його синтаксичний та семантичний аналіз, виявляти потенційні вразливості або неоптимальні ділянки. Відповіді AI-помічника рендеряться з використанням розмітки Markdown, що дозволяє виділяти ключові терміни, будувати структуровані списки та відображати фрагменти коду зі спеціальним підсвічуванням синтаксису. Таке візуальне структурування інформації значно знижує когнітивне навантаження на студента та сприяє швидшому читанню й розумінню складних технічних текстів [18].

Ще однією інноваційною характеристикою розробленої системи є використання технології потокової передачі даних (streaming) під час генерації відповідей. У традиційних HTTP-запитах клієнт змушений чекати, поки сервер повністю згенерує весь обсяг тексту, що у випадку з LLM може займати до кількох десятків секунд. В освітньому контексті такі затримки руйнують ілюзію живого спілкування та знижують концентрацію уваги [21]. Реалізація streaming-інтерфейсу дозволяє системі виводити відповідь на екран студента по мірі її генерації — токен за токеном. Це не лише оптимізує метрики продуктивності застосунку, але й має важливий психологічний ефект: користувач одразу починає читати та аналізувати перші рядки пояснення, перебуваючи в стані безперервного когнітивного залучення.

Окремим, найбільш складним компонентом інтелектуальної системи підтримки є автоматизована генерація засобів контролю знань. У класичних системах дистанційного навчання викладач змушений вручну складати банки тестових питань, які швидко втрачають актуальність і легко компрометуються студентами [5]. Розроблений AI-помічник вирішує цю проблему принципово іншим шляхом: він містить ізольований модуль, який за текстовим запитом на певну тему динамічно генерує унікальний набір тестових питань. Завдяки налаштуванням API, модель повертає ці дані у суворо структурованому машиночитаному форматі (JSON), що дозволяє клієнтському застосунку автоматично розпарсити їх, вивести у вигляді інтерактивного графічного інтерфейсу та самостійно перевірити правильність відповідей користувача. Кожен згенерований тест містить не лише

питання та варіанти відповідей, але й детальне пояснення до кожного пункту, що перетворює сам процес тестування на додатковий етап навчання.

Таким чином, розроблена інтелектуальна система підтримки навчання виходить далеко за межі традиційного розуміння освітнього програмного забезпечення. Вона поєднує в собі потужність великих мовних моделей, передові архітектурні рішення веб-розробки та сучасні педагогічні практики. Інтегруючи функції розумного діалогу, аналізу програмного коду, потокової передачі інформації та динамічної генерації тестів, ця система створює високоадаптивне цифрове середовище, здатне ефективно супроводжувати студента на всіх етапах опанування складних технічних дисциплін.

1.3 Аналіз існуючих рішень

Проєктування будь-якої нової інформаційної системи вимагає ретельного дослідження ринку та аналізу вже існуючих програмних продуктів. На сьогодні індустрія освітніх технологій (EdTech) пропонує широкий спектр рішень, спрямованих на цифровізацію навчання. Проте детальне вивчення їхнього функціоналу свідчить про те, що більшість із них вирішують лише вузькоспеціалізовані завдання і не здатні забезпечити комплексну, персоналізовану та інтелектуальну підтримку студента в режимі реального часу. Усі існуючі на ринку рішення умовно можна поділити на чотири основні категорії: класичні системи управління навчанням, масові відкриті онлайн-курси, універсальні чат-боти загального призначення та вузькоспеціалізовані інструменти для розробників [15].

Перша і найпоширеніша категорія — це класичні системи управління навчанням (Learning Management Systems, LMS), лідерами серед яких є Moodle, Canvas та Blackboard. Ці системи стали стандартом де-факто для адміністрування освітнього процесу в університетах [21]. Їхньою беззаперечною перевагою є потужний функціонал для структурування курсів, ведення журналів успішності, управління ролями користувачів та організації тестування. Однак головний недолік

LMS полягає в їхній статичності та відсутності інтелектуальної адаптивності. Вони функціонують виключно як сховища інформації: викладач завантажує матеріали, а студент їх споживає. Якщо під час виконання лабораторної роботи з програмування у студента виникає помилка компіляції, Moodle не зможе підказати шлях до її вирішення. Зворотний зв'язок у таких системах є асинхронним і повністю залежить від фізичної наявності викладача, що робить їх неефективними для самостійної роботи здобувачів освіти у позааудиторний час [5].

Друга категорія — платформи масових відкритих онлайн-курсів (MOOC), такі як Coursera, edX або Udemy. Вони демократизували доступ до високоякісного контенту від провідних університетів світу. Проте педагогічний дизайн цих платформ орієнтований на лінійне, одностороннє навчання, де основним інструментом виступає відеолекція. Згідно з дослідженнями, відсоток успішного завершення таких курсів залишається вкрай низьким саме через брак індивідуального менторства та неможливість отримати миттєву відповідь на специфічне запитання [18]. Навіть вбудовані форуми для обговорень не вирішують цієї проблеми, оскільки очікування відповіді від інших учасників або кураторів може тривати годинами чи днями, що розриває когнітивний процес засвоєння нової інформації.

Третя категорія є технологічно найближчою до теми даної кваліфікаційної роботи — це універсальні системи на базі великих мовних моделей, найвідомішою з яких є ChatGPT від компанії OpenAI [7]. Безумовно, ChatGPT володіє колосальною базою знань і здатний генерувати високоякісний текст та програмний код. Однак використання універсальних LLM-моделей у чистому вигляді має суттєві недоліки в контексті академічного навчання. По-перше, такі системи не мають педагогічних обмежень: на запит студента "як вирішити цю задачу" універсальний чат-бот, як правило, просто видає готовий, оптимізований код [19]. Студент отримує результат, але не розуміє алгоритмічного шляху до нього, що стимулює механічне копіювання і шкодить формуванню інженерного мислення. По-друге, інтерфейс універсальних ботів не оптимізований під структурований навчальний процес — у них відсутні

вбудовані модулі для перевірки знань (наприклад, генератори тестів) або збереження навчального прогресу за конкретними темами.

Четверта категорія стосується спеціалізованих AI-інструментів для розробників програмного забезпечення, таких як GitHub Copilot або Tabnine [16]. Їхнім головним завданням є підвищення продуктивності професійних програмістів шляхом автодоповнення коду безпосередньо в середовищі розробки (IDE). Хоча ці інструменти є надзвичайно ефективними у комерційній розробці, їх використання в освітньому процесі є деструктивним. Студент, який тільки вивчає основи синтаксису, отримує автоматично написані блоки коду до того, як сам встигає осмислити логіку алгоритму. Такі системи створені для того, щоб писати код замість людини, а не для того, щоб навчати людину писати код.

Синтезуючи результати аналізу існуючих рішень, можна стверджувати, що на ринку існує гострий дефіцит спеціалізованих платформ, які б поєднували інтелектуальну міць сучасних LLM з правильним педагогічним дизайном. Саме цю нішу заповнює інформаційна система, що розробляється у даній кваліфікаційній роботі. Її архітектура та логіка роботи спроєктовані з урахуванням виявлених недоліків конкурентних рішень.

Головною перевагою розробленої системи є використання спеціалізованого системного промпта (system prompt), який перетворює універсальну мовну модель на академічного ментора. Система свідомо обмежує видачу прямих відповідей на завдання, натомість пропонуючи студенту покрокові пояснення, підказки, приклади використання патернів та запитання для саморефлексії [8]. Іншою унікальною перевагою є вбудований модуль динамічного тестування: на відміну від ChatGPT, який може лише написати текст питання в чаті, розроблена система генерує форматований JSON-об'єкт, який клієнтський застосунок (на базі React.js) перетворює на інтерактивний графічний інтерфейс для проходження тесту з автоматичним підрахунком балів. У поєднанні з безпечним збереженням історії діалогів у базі даних Supabase та інтуїтивним чат-інтерфейсом, це робить запропоноване рішення значно ефективнішим, безпечнішим та адаптивнішим інструментом для університетської освіти порівняно з існуючими аналогами.

1.4 Проблеми сучасного навчання

Сучасна система вищої освіти, зокрема у сфері підготовки фахівців із комп'ютерних наук та інформаційних технологій, перебуває у стані глибокої трансформації. Цей процес зумовлений експоненційним зростанням обсягів технічної інформації, швидкою зміною актуальних інструментів розробки та підвищенням вимог ринку праці до випускників закладів вищої освіти. Незважаючи на активне впровадження систем управління навчанням (LMS) та цифрових платформ, у навчальному процесі досі залишається низка системних проблем, які суттєво знижують ефективність засвоєння складних інженерних та алгоритмічних концепцій.

Однією з найголовніших проблем сучасного навчання є критичне когнітивне перевантаження здобувачів освіти. Вивчення комп'ютерних наук вимагає одночасного засвоєння абстрактних математичних моделей, синтаксису різних мов програмування, принципів проєктування баз даних та розуміння архітектури програмного забезпечення. Традиційні методи викладання, що базуються на статичних підручниках або попередньо записаних відеолекціях, не здатні забезпечити інтерактивне пояснення матеріалу [16]. Коли студент стикається зі складною темою (наприклад, розумінням рекурсії, асинхронного програмування або оптимізації SQL-запитів), статичний матеріал не може адаптуватися під його поточний рівень знань. Це призводить до формального, поверхневого заучування замість глибокого структурного розуміння предмета.

Другою суттєвою проблемою є відсутність персоналізації навчальних траєкторій. В умовах класичної академічної групи викладач змушений орієнтуватися на "середнього" студента, що залишає поза увагою як тих, хто потребує додаткового часу на засвоєння бази, так і тих, хто готовий переходити до більш складних завдань [22]. Стандартні освітні платформи пропонують однаковий контент для всіх користувачів незалежно від їхнього попереднього досвіду. Відсутність адаптивності призводить до того, що навчання стає негнучким: система не може автоматично генерувати простіші приклади, якщо студент робить помилки,

або, навпаки, ускладнювати тестові завдання, якщо матеріал засвоюється надто швидко.

Третя, і, можливо, найбільш критична проблема для ІТ-спеціальностей — це відсутність миттєвого та розгорнутого зворотного зв'язку (feedback loop). У процесі написання програмного коду або проєктування алгоритмів студенти неминуче роблять синтаксичні та логічні помилки. У традиційному форматі студент може годинами шукати причину помилки у своєму коді, чекаючи на консультацію викладача або шукаючи розрізнену інформацію на профільних форумах. Такий тривалий цикл зворотного зв'язку викликає розчарування, психологічне вигорання та стрімке падіння мотивації до навчання [18]. Студенту потрібен не просто правильний варіант коду, а детальне пояснення того, чому саме його підхід був помилковим, і як слід оптимізувати алгоритм. Традиційні системи тестування здатні лише констатувати факт помилки (наприклад, невідповідність очікуваному результату компіляції), але не здатні провести семантичний аналіз коду та надати менторську пораду.

Крім того, існує проблема розриву між теоретичним матеріалом та його практичним закріпленням. Після прослуховування лекції студент потребує негайної практики для закріплення нових нейронних зв'язків. Проте створення якісних, різнорівневих практичних завдань та тестів для кожної теми вимагає величезних витрат часу від викладача [5]. Як наслідок, студенти часто стикаються з дефіцитом свіжих тестових завдань для самоперевірки. Вони змушені проходити одні й ті самі тести багаторазово, що призводить до механічного запам'ятовування відповідей, а не до реальної перевірки знань.

Усі ці фактори — когнітивне перевантаження, дефіцит персоналізації, відсутність миттєвого менторського зворотного зв'язку та брак адаптивної практики — підтверджують гостру необхідність модернізації освітнього інструментарію. Вирішення цих проблем неможливе лише за рахунок покращення традиційних LMS; воно вимагає впровадження систем принципово нового класу — інформаційних систем інтелектуальної підтримки, які здатні імітувати поведінку живого тьютора. Використання великих мовних моделей (LLM) дозволяє закрити

всі вищезазначені прогалини: такі системи можуть працювати зі студентом 24/7, надавати індивідуальні пояснення будь-якого рівня складності, проводити рев'ю коду у реальному часі та автоматично генерувати унікальні тести для закріплення знань [8].

1.5 Постановка задачі

Усвідомлення критичних проблем сучасного освітнього процесу, описаних у попередньому підрозділі, а також аналіз існуючих технологічних рішень формують чітке підґрунтя для розробки власного програмного продукту. Постановка задачі є фундаментальним етапом проєктування, який формалізує вимоги до майбутньої системи, окреслює межі дослідження та визначає критерії успішності розробки.

Мета розробки

Головною метою виконання даної кваліфікаційної роботи є проєктування та програмно-технічна реалізація інформаційної системи інтелектуальної підтримки навчання на основі AI-помічника. Ця система повинна функціонувати як сучасний, високопродуктивний вебзастосунок, здатний у режимі реального часу взаємодіяти з користувачем, надавати розгорнуті пояснення навчального матеріалу (зокрема з дисциплін комп'ютерних наук), генерувати приклади програмного коду та автоматизовано створювати тестові завдання для перевірки засвоєних знань.

Об'єкт і предмет дослідження

Об'єктом дослідження є процес організації індивідуалізованого навчання та перевірки знань здобувачів освіти за допомогою цифрових інформаційних систем у сучасному освітньому середовищі. Предметом дослідження виступають моделі, алгоритми, архітектурні рішення та програмні інструменти (вебфреймворки, хмарні бази даних, API великих мовних моделей), які використовуються для створення інтелектуальних освітніх платформ.

Функціональні вимоги до системи

Для досягнення поставленої мети розроблювана інформаційна система повинна задовольняти низку суворих функціональних вимог. Згідно з сучасними стандартами веброзробки, система має забезпечувати:

– *Повноцінну взаємодію у форматі чату:* користувач повинен мати можливість вводити текстові запити, а система — обробляти їх через зовнішнє AI API та повертати змістовні, контекстно-залежні відповіді.

– *Збереження контексту діалогу:* система зобов'язана зберігати історію поточного діалогу, щоб мовна модель могла розуміти посилання на попередні питання користувача і вести логічно послідовну розмову.

– *Потокову генерацію відповідей (Streaming):* відповіді від штучного інтелекту повинні виводитися на екран користувача поступово (токенами) у режимі реального часу, мінімізуючи затримку очікування.

– *Форматування технічного контенту:* система повинна вміти розпізнавати та коректно рендерити блоки програмного коду (з підсвічуванням синтаксису) та форматування Markdown, що є критичним для IT-навчання.

– *Автоматичну генерацію тестів:* застосунок повинен містити окремий модуль, який за запитом користувача генеруватиме структуровані тестові питання (у форматі JSON) із варіантами відповідей та детальними поясненнями.

– *Персоналізацію та авторизацію:* наявність захищеної системи реєстрації та входу користувачів для збереження персональної історії діалогів та результатів тестування у базі даних.

Нефункціональні вимоги (вимоги до якості системи)

Окрім прямого функціоналу, програмний продукт має відповідати високим стандартам якості, що визначаються нефункціональніми вимогами:

– *Продуктивність та швидкодія:* час відгуку інтерфейсу має бути миттєвим завдяки архітектурі Single Page Application (SPA). Затримка перед початком відображення відповіді від AI не повинна перевищувати 2–5 секунд.

– *Масштабованість та надійність:* архітектура системи повинна базуватися на хмарних рішеннях (Backend-as-a-Service), що дозволить легко

масштабувати базу даних та витримувати зростання кількості одночасних користувачів без падіння серверів.

– *Безпека та ізоляція даних:* система повинна гарантувати конфіденційність користувацьких даних шляхом впровадження механізмів безпеки на рівні бази даних (Row Level Security), щоб жоден користувач не міг отримати доступ до історії запитів інших студентів.

– *Ергономічність (UI/UX):* користувацький інтерфейс повинен бути інтуїтивно зрозумілим, розробленим у сучасному стилі (з підтримкою "темної теми" для зменшення навантаження на зір), та повністю адаптивним для коректного відображення як на персональних комп'ютерах, так і на мобільних пристроях.

Очікувані результати

В результаті виконання всіх поставлених завдань буде створено готовий до експлуатації програмний продукт, який вирішує проблему браку індивідуального менторства в сучасній освіті. Очікується, що впровадження такої інформаційної системи дозволить студентам значно швидше опановувати складні технічні дисципліни, знизить рівень стресу під час виконання практичних робіт та підвищить загальну об'єктивність самооцінювання завдяки генерації унікальних тестових завдань. Створена система стане надійним фундаментом для подальшого розширення, зокрема для можливої інтеграції з існуючими університетськими системами управління навчанням у майбутньому.

Висновки до розділу:

У першому розділі проведено детальний аналіз предметної області та досліджено роль сучасних інформаційних систем у сфері освіти. Розглянуто еволюцію освітніх платформ від класичних систем управління навчанням (LMS) до інтелектуальних систем підтримки навчання на базі великих мовних моделей (LLM). Здійснено аналіз існуючих програмних рішень (MOOC, загальні чат-боти, спеціалізовані інструменти для розробників), визначено їхні переваги та суттєві недоліки, такі як відсутність персоналізації та надання готових розв'язків замість

менторства. На основі виявлених проблем сучасного навчання, зокрема когнітивного перевантаження та дефіциту миттєвого зворотного зв'язку, сформульовано постановку задачі та визначено чіткі функціональні й нефункціональні вимоги до розроблюваної інформаційної системи на основі AI-помічника.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Загальна архітектура системи

Розроблювана інформаційна система інтелектуальної підтримки навчання на основі AI-помічника є сучасним вебзастосунком, який забезпечує безперервну та індивідуалізовану взаємодію користувача з великою мовною моделлю. Основною метою архітектурного проєктування системи було забезпечення стабільної, масштабованої та зручної комунікації між користувацьким інтерфейсом та хмарними сервісами штучного інтелекту. Загальна архітектура системи побудована за розподіленим клієнт-серверним принципом із глибокою інтеграцією зовнішніх хмарних платформ. Такий підхід дозволяє радикально мінімізувати обчислювальне навантаження на локальний пристрій користувача, делегуючи найбільш ресурсоємні завдання (обробку природної мови та роботу з базами даних) на віддалені сервери, що забезпечує кросплатформність та доступність системи з будь-якого пристрою, підключеного до мережі Інтернет.

Архітектура системи концептуально поділяється на кілька взаємопов'язаних рівнів, базовим з яких є клієнтський рівень (Frontend). Цей рівень реалізований на основі бібліотеки React.js і функціонує за моделлю односторінкового застосунку (Single Page Application). Головним завданням клієнтського рівня є забезпечення миттєвої реакції на дії користувача без необхідності перезавантаження сторінок у браузері. Frontend відповідає за рендеринг інтерфейсу чату, перехоплення введених текстових повідомлень, навігацію між основними модулями (маркетинговою сторінкою, модулем тестування та зоною авторизації) та локальне управління станом застосунку. Завдяки компонентному підходу, клієнтський рівень є високоуніфікованим: кожен елемент інтерфейсу працює незалежно, що суттєво спрощує подальшу підтримку та масштабування коду.

Рівень обробки запитів та інтелектуальної взаємодії винесений за межі локального середовища і реалізується через інтеграцію із зовнішнім прикладним

програмним інтерфейсом великої мовної моделі (Groq API). Система не потребує розгортання та навчання власної нейромережі, що робить архітектуру легкою та економічно доцільною. Коли користувач ініціює запит, клієнтський застосунок формує пакет даних, що включає системну інструкцію (промпт) та історію поточного діалогу, і відправляє його до AI-сервера. Важливою архітектурною особливістю цього рівня є підтримка потокової передачі даних (streaming-режиму). Мовна модель не акумулює всю відповідь на сервері, а повертає її клієнту поступово, токенами. Це створює ефект живого діалогу і значно підвищує ергономічність системи.

Замість розгортання класичного монолітного сервера, рівень збереження даних та автентифікації побудований за моделлю Backend-as-a-Service (BaaS) з використанням хмарної платформи Supabase. Цей рівень забезпечує функціонування реляційної бази даних PostgreSQL та системи безпеки. Архітектурно модуль бази даних відповідає за надійне збереження структурованої інформації: профілів користувачів, історії діалогів, окремих повідомлень та метаданих результатів тестування. Водночас інтегрований модуль Supabase Auth бере на себе завдання з реєстрації користувачів, генерації сесійних токенів та управління доступом. Завдяки цьому забезпечується сувора індивідуалізація навчального процесу — кожен студент має доступ виключно до власних чат-сесій та результатів.

Наскрізна взаємодія всіх цих архітектурних рівнів відбувається синхронно та у строго визначеному порядку. Життєвий цикл базової операції розпочинається з того, що користувач вводить питання у графічному інтерфейсі. Клієнтська частина миттєво відображає цей запит і паралельно ініціює два мережеві виклики: перший — до AI API для отримання змістовної відповіді, другий — до бази даних Supabase для фіксації повідомлення в історії. Отримана від штучного інтелекту потокова відповідь рендериться у чаті, після чого також синхронізується з хмарним сховищем. Цей механізм гарантує консистентність даних і безперебійну роботу віртуального помічника.

Запропонована архітектурна модель має низку вагомих переваг, серед яких простота технічної підтримки, відсутність необхідності адміністрування власного фізичного сервера та здатність до безболісного горизонтального масштабування за рахунок потужностей хмарних провайдерів. Крім того, використання готових API-рішень дозволило зосередити основні зусилля розробки на навчальній бізнес-логіці та інтерфейсі. Водночас архітектура містить і певні об'єктивні обмеження, властиві всім хмарним застосункам: вона є повністю залежною від стабільності інтернет-з'єднання та працездатності сторонніх сервісів (Groq та Supabase), а також підпорядковується лімітам пропускної здатності їхніх API. Проте в умовах сучасного підключення до мережі ці обмеження є прийнятними, і загальна архітектура повністю задовольняє вимоги до створення сучасних освітніх платформ.

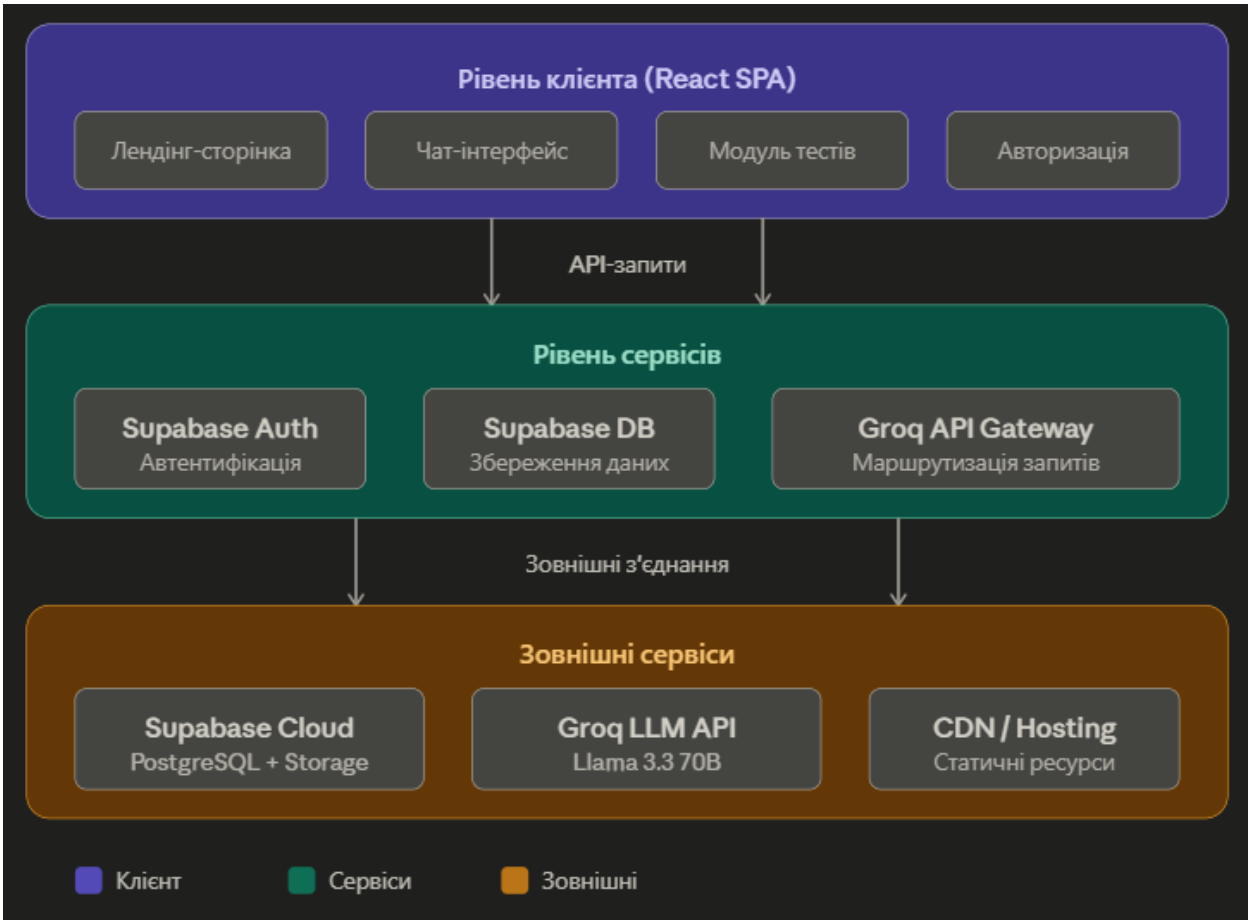


Рисунок. 2.1 – Загальна архітектура системи

2.2 Вибір технологій реалізації інформаційної системи

Вибір технологій для реалізації інформаційної системи інтелектуальної підтримки навчання є одним із ключових етапів проєктування, оскільки саме від нього залежить ефективність, продуктивність, масштабованість та зручність подальшої підтримки системи. Оскільки розроблювана система базується на принципах роботи AI-помічника з використанням сучасних великих мовних моделей (LLM), архітектура повинна забезпечувати надійну інтеграцію із зовнішніми API, обробку запитів користувачів у режимі реального часу, а також збереження навчальних даних.

Основною мовою програмування для реалізації клієнтської частини системи було обрано JavaScript (ES6+) у поєднанні з бібліотекою React.js. Використання React дозволяє побудувати вебзастосунок за принципом Single Page Application (SPA), що забезпечує динамічне оновлення інтерфейсу без необхідності повного перезавантаження сторінки. Завдяки компонентному підходу значно спрощується підтримка та масштабування коду: кожен елемент інтерфейсу (вікно чату, форма авторизації, модуль тестування) функціонує як незалежний компонент зі своїм станом. Візуальне оформлення реалізовано за допомогою каскадних таблиць стилів (CSS), що забезпечує адаптивний дизайн для різних пристроїв та дозволяє впроваджувати складні анімації, зокрема плавне відображення повідомлень. Розробка здійснюється у середовищі Visual Studio Code, яке є стандартом індустрії завдяки зручній інтеграції з системами контролю версій (Git) та розширеним інструментам дебагінгу.

Для реалізації серверної логіки та роботи з даними було вирішено відмовитися від класичного монолітного бекенд-сервера на користь хмарної платформи Supabase, яка працює за моделлю Backend-as-a-Service (BaaS). Вона забезпечує роботу з надійною реляційною базою даних PostgreSQL. У межах системи Supabase виконує кілька критичних функцій: керує автентифікацією користувачів, зберігає історію чатів (діалоги та повідомлення) та фіксує результати пройдених тестів. Взаємодія між клієнтським застосунком та базою даних

відбувається через автоматично згенерований REST API за допомогою стандартних методів GET, POST, INSERT та DELETE. Вибір цієї платформи обґрунтований її здатністю до швидкої інтеграції, відсутністю потреби у складній серверній інфраструктурі та підтримкою обробки даних у реальному часі.

Ключовим інтелектуальним ядром інформаційної системи є велика мовна модель. Для її інтеграції було обрано хмарний сервіс Groq API, який надає швидкий доступ до сучасних моделей лінійки LLaMA 3. Використання зовнішнього API дозволило уникнути необхідності розгортання та навчання власної нейромережі, що потребувало б значних обчислювальних потужностей. Модель виконує роль віртуального викладача: вона генерує відповіді на запитання студентів, пояснює складні теми, наводить приклади програмного коду та автоматично формує тестові завдання. Взаємодія з Groq API здійснюється через HTTP-запити, у яких передається системний промпт (інструкція, що задає роль AI як навчального асистента), історія попереднього діалогу та параметри генерації.

Важливою особливістю інтеграції штучного інтелекту є використання режиму потокової передачі даних (streaming). Це дозволяє системі не чекати повної генерації відповіді мовною моделлю, а виводити її на екран користувача поступово. Такий підхід створює ефект живого діалогу та суттєво покращує загальний користувацький досвід.

Безпека даних у розробленій системі гарантується модулем Supabase Auth, який підтримує реєстрацію за електронною поштою, генерацію токенів сесій та ізоляцію даних користувачів. Завдяки використанню політик безпеки на рівні рядків (Row Level Security) у PostgreSQL, кожен студент має доступ виключно до власних діалогів та результатів тестування, що унеможливлює несанкціонований витік конфіденційної інформації.

У підсумку, обраний гібридний стек технологій забезпечує високу швидкість розробки, відмінну масштабованість та низькі вимоги до апаратного забезпечення, оскільки найважчі обчислювальні процеси делеговані зовнішнім хмарним сервісам.

2.3 Структура програмної системи

Структура програмної системи інтелектуальної підтримки навчання визначає логічну організацію її компонентів, їхні взаємозв’язки та принципи взаємодії. Оскільки розроблена система реалізує складну функціональність віртуального навчального помічника, її архітектура побудована за принципом розподіленої клієнт-серверної взаємодії з активним використанням хмарних сервісів. Для забезпечення високої гнучкості, масштабованості та чіткого розподілу відповідальності систему концептуально поділено на три основні рівні: клієнтський рівень (Frontend), рівень бізнес-логіки та інтеграції штучного інтелекту, а також рівень збереження даних (Database/Backend-as-a-Service).

Клієнтський рівень системи відповідає за безпосередню взаємодію користувача з програмним продуктом і реалізований за допомогою сучасної бібліотеки React.js. Вибір цієї технології зумовлений необхідністю створення Single Page Application (SPA) — односторінкового застосунку, який забезпечує миттєву реакцію інтерфейсу без перезавантаження сторінок браузера. Це критично важливо для освітнього чату, де користувач очікує плавного і безперервного спілкування. Структура React-додатка побудована на основі ізольованих компонентів. Головною точкою входу є інформаційна сторінка (LandingPage), після якої користувач потрапляє на модуль авторизації (AuthPage). Основним робочим простором є компонент чат-інтерфейсу (ChatPage), який включає допоміжні підкомпоненти: панель історії діалогів (Sidebar) для швидкої навігації та модуль рендерингу повідомлень (MessageContent), що відповідає за коректне відображення тексту, розмітки Markdown та підсвічування синтаксису програмного коду. Додатково виділено незалежний компонент тестування (QuizPage), який формує інтерактивний графічний інтерфейс для проходження згенерованих завдань.

Рівень бізнес-логіки виступає сполучною ланкою між користувацьким інтерфейсом, базою даних та зовнішніми обчислювальними потужностями. У даній системі бізнес-логіка не винесена на окремий монолітний сервер, а реалізована у вигляді інтеграційних API-запитів. Головним її завданням є правильне формування

контексту для штучного інтелекту. Коли користувач надсилає повідомлення, система не просто пересилає цей текст до AI. Вона збирає масив даних, який включає жорстко заданий системний промпт (правила поведінки віртуального викладача), історію поточного діалогу (для збереження контексту) та безпосередньо новий запит користувача.

Окремим і найважливішим модулем цього рівня є модуль штучного інтелекту, що функціонує через інтеграцію з Groq API. Вибір Groq зумовлений його архітектурною особливістю — надзвичайно низькою затримкою (latency) при виведенні тексту. AI-модуль вирішує ключові задачі: генерує пояснення складних тем, аналізує код користувача та формує структуру тестових завдань. Завдяки реалізації потокової обробки даних (streaming), відповідь від моделі відображається у чаті поступово (токен за токеном), що створює ілюзію живого набору тексту реальним викладачем.

Рівень зберігання даних побудований на базі платформи Supabase, яка надає готову інфраструктуру Backend-as-a-Service на основі реляційної системи керування базами даних PostgreSQL. Вибір Supabase обґрунтований тим, що ця платформа забезпечує високий рівень безпеки (через політики Row Level Security) та дозволяє швидко реалізувати авторизацію без написання власного бекенд-коду. База даних складається з чотирьох основних нормалізованих таблиць: таблиці користувачів (users), списку створених діалогів (conversations), безпосередньо повідомлень у цих чатах (messages) та таблиці збереження результатів пройдених тестів (quiz_results). Зв'язки між таблицями побудовані за класичним принципом "один до багатьох", що запобігає дублюванню інформації та гарантує цілісність збереження навчальної історії кожного студента.

Взаємодія між усіма описаними компонентами відбувається синхронно та у режимі реального часу. Життєвий цикл одного запиту виглядає наступним чином: користувач вводить питання у React-інтерфейсі, після чого клієнтський застосунок формує HTTP-запит до Groq API. Отримавши перші байти відповіді, система починає потоковий рендеринг тексту на екрані. Після повного завершення генерації відповіді, фронтенд ініціює запит до Supabase для збереження пари "питання-

відповідь" у базі даних, що автоматично оновлює історію в боковій панелі користувача. Запропонована модульна структура гарантує високу продуктивність системи, простоту її технічної підтримки та відсутність необхідності у складному серверному адмініструванні.

2.4 Архітектурна схема взаємодії компонентів

Архітектурна схема взаємодії компонентів інформаційної системи інтелектуальної підтримки навчання описує спосіб обміну даними між основними модулями програмного забезпечення, а також визначає строгий порядок виконання операцій. Оскільки розроблювана система функціонує як інтерактивний AI-асистент, головний акцент в архітектурі зроблено на швидкості обробки користувацьких запитів, асинхронній взаємодії із зовнішніми прикладними програмними інтерфейсами (API) та гарантованій безпеці збереження персональних даних. Архітектура побудована за принципом розподіленої взаємодії, де кожен модуль виконує єдину, чітко визначену функцію (Single Responsibility Principle) і комунікує з іншими частинами системи через стандартизовані HTTP/REST запити.

Глобально архітектурну модель системи можна уявити у вигляді трьох взаємопов'язаних рівнів: рівня презентації (Presentation Layer), рівня обробки даних (Application Layer) та рівня збереження даних (Data Layer). Рівень презентації реалізовано на базі React-додатка, і він відповідає виключно за рендеринг користувацького інтерфейсу: відображення чату, вікон авторизації та інтерактивних форм тестування. Рівень обробки даних виступає посередником: він керує станом додатку, формує динамічні запити до штучного інтелекту та парсить отримані відповіді. Рівень збереження даних базується на хмарній інфраструктурі Supabase (PostgreSQL), яка забезпечує персистентність — тобто надійне збереження історії діалогів, облікових записів та результатів тестування між сесіями користувача.

Будь-яка взаємодія в системі ініціюється подією з боку користувача, найчастішою з яких є відправка текстового повідомлення у чат. Коли здобувач

освіти вводить запит, фронтенд-компонент перехоплює цю подію та передає текст до головної функції обробки. Оскільки зовнішня AI-модель (Groq API) є «безстанковою» (stateless) і не зберігає пам'ять про попередні повідомлення, система бере на себе завдання з формування контексту. Вона об'єднує системний промпт (який задає роль віртуального викладача), масив попередніх повідомлень із поточного діалогу та новий запит користувача, після чого пакує ці дані у формат JSON і відправляє через HTTP POST запит до сервера генерації.

Ключовою архітектурною особливістю цієї взаємодії є використання потокової обробки даних (streaming-режиму). Замість того, щоб чекати повного завершення генерації відповіді (що може зайняти значний час при обробці складних алгоритмічних питань), API відкриває потокове з’єднання. Сервер штучного інтелекту починає надсилати згенерований текст невеликими фрагментами (токенами), а React-інтерфейс миттєво відображає їх на екрані. Це суттєво зменшує затримку сприйняття (latency) та створює психологічний ефект живого спілкування, що є критично важливим для утримання уваги студента під час навчання.

Окремої уваги заслуговує схема взаємодії під час роботи модуля тестування. Процес генерації тестів відрізняється від звичайного чату високими вимогами до структуризації даних. Коли користувач ініціює створення тесту на певну тему, система формує спеціалізований запит, який змушує мовну модель повернути результат виключно у суворому машиночитаному форматі JSON (рис. 2.2). Отримавши таку відповідь, рівень обробки даних (Application Layer) парсить об'єкт, розділяючи його на масиви питань, варіантів відповідей та правильних рішень. Після цього дані передаються на рівень презентації для відображення інтерактивної форми опитування. По завершенню тесту система автоматично підраховує бали та ініціює INSERT-запит до бази даних Supabase для фіксації результатів успішності.

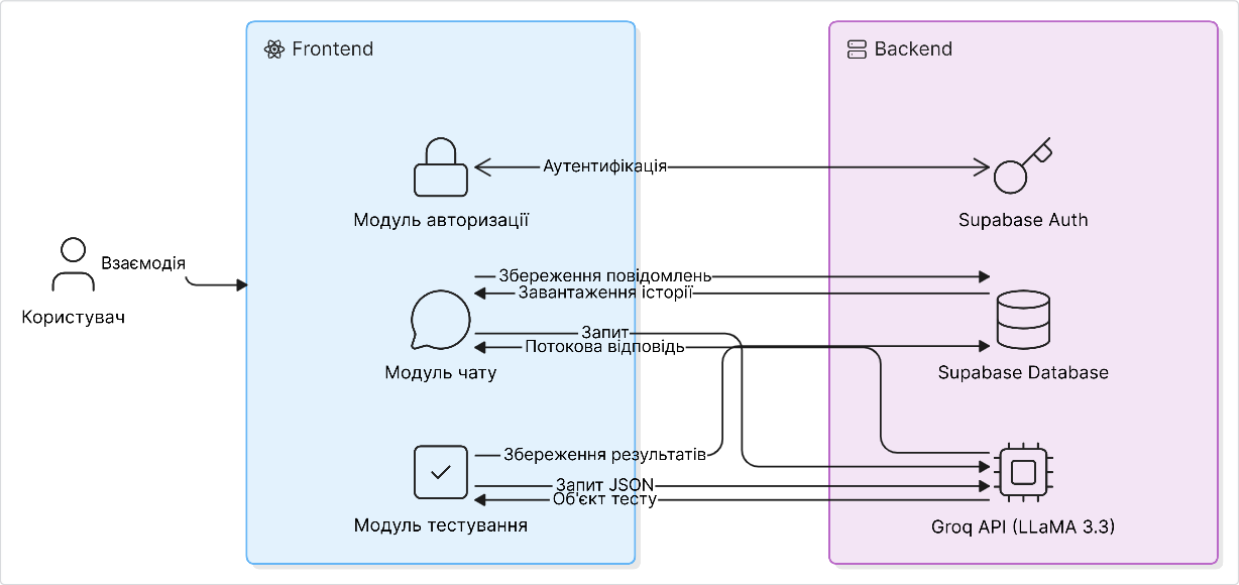


Рисунок 2.2 – Архітектурна схема взаємодії компонентів системи EduMind AI

З точки зору безпеки, взаємодія компонентів спирається на надійні механізми автентифікації та розмежування прав доступу. Усі запити до бази даних проходять через Supabase Auth з використанням захищених JWT-токенів. Крім того, на рівні бази даних налаштовано політики безпеки на рівні рядків (Row Level Security), що гарантує абсолютну ізоляцію даних: кожен студент має доступ виключно до власних діалогів та оцінок. Запропонована архітектурна схема з чітким поділом на незалежні компоненти не лише забезпечує високу продуктивність та захищеність системи на даному етапі, але й створює надійний фундамент для її майбутнього масштабування, наприклад, для додавання модулів голосового розпізнавання чи інтеграції з університетськими системами дистанційного навчання.

2.5 Проєктування бази даних

Проєктування бази даних є одним із фундаментальних етапів розробки інформаційної системи інтелектуальної підтримки навчання. Оскільки розроблювана система орієнтована на інтенсивну роботу з великими масивами текстової інформації (користувацькі запити, розгорнуті відповіді штучного інтелекту, пояснення алгоритмів та згенеровані тестові завдання), правильна

організація збереження даних є критично важливою. Головна мета проєктування полягала у створенні надійної, швидкодіючої та масштабованої структури, яка забезпечує цілісність даних, усуває їх дублювання та дозволяє ефективно відновлювати контекст попередніх діалогів для мовної моделі. Як технологічну основу було обрано реляційну систему керування базами даних PostgreSQL, розгорнуту на базі хмарної платформи Supabase.

Важливою архітектурною особливістю бази даних є використання вбудованої системи Supabase Auth. Замість створення окремої публічної таблиці користувачів, система покладається на захищену внутрішню схему `auth.users`, яка автоматично генерує та зберігає унікальні ідентифікатори (UUID) для кожного зареєстрованого студента. Логічна модель публічної схеми (`public`) спроєктована з дотриманням правил нормалізації та складається з трьох основних взаємопов'язаних сутностей (табл. 2.1 – 2.3):

- Таблиця діалогів (conversations): відповідає за логічне групування повідомлень в окремі чат-сесії. Поля таблиці включають `id` (первинний ключ діалогу), `user_id` (зовнішній ключ, що вказує на ідентифікатор власника з внутрішньої таблиці `auth.users`), `title` (коротка назва розмови для відображення у навігаційній панелі) та `created_at` (мітка часу). Таке виокремлення сесій є необхідним для того, щоб користувач міг паралельно вести кілька незалежних обговорень з різних тем.
- Таблиця повідомлень (messages): є найбільш навантаженою сутністю системи, оскільки зберігає безпосередню історію взаємодії. Структура включає `id` (первинний ключ), `conversation_id` (зовнішній ключ зв'язку з конкретним діалогом), `role` (роль відправника: `user` для студента або `assistant` для штучного інтелекту), `content` (безпосередньо текстове наповнення повідомлення) та `created_at`. Збереження ролей відправників є критичним для коректного відновлення контексту при наступних зверненнях до AI API.
- Таблиця результатів тестів (quiz_results): ізольована сутність для збереження академічної успішності користувача. Вона містить поля `id`, `user_id`

(зовнішній ключ, пов'язаний із системою авторизації), `topic` (тема пройденого тесту), `score` (кількість правильних відповідей), `total` (загальна кількість згенерованих питань) та `created_at`. Ця таблиця дозволяє формувати аналітику успішності та відстежувати прогрес студента.

Взаємодія між цими сутностями побудована на основі строгих реляційних зв'язків типу «один до багатьох» (1:M). Згідно з цією схемою, один користувач (з системи Supabase Auth) може ініціювати безліч діалогів та мати множину записів про результати проходження тестів. У свою чергу, кожен діалог може містити необмежену кількість повідомлень. Завдяки такій нормалізованій архітектурі, повідомлення не містять прямої інформації про користувача, а пов'язуються з ним виключно транзитивно через таблицю діалогів, що суттєво оптимізує швидкодію SQL-запитів (рис. 2.3).

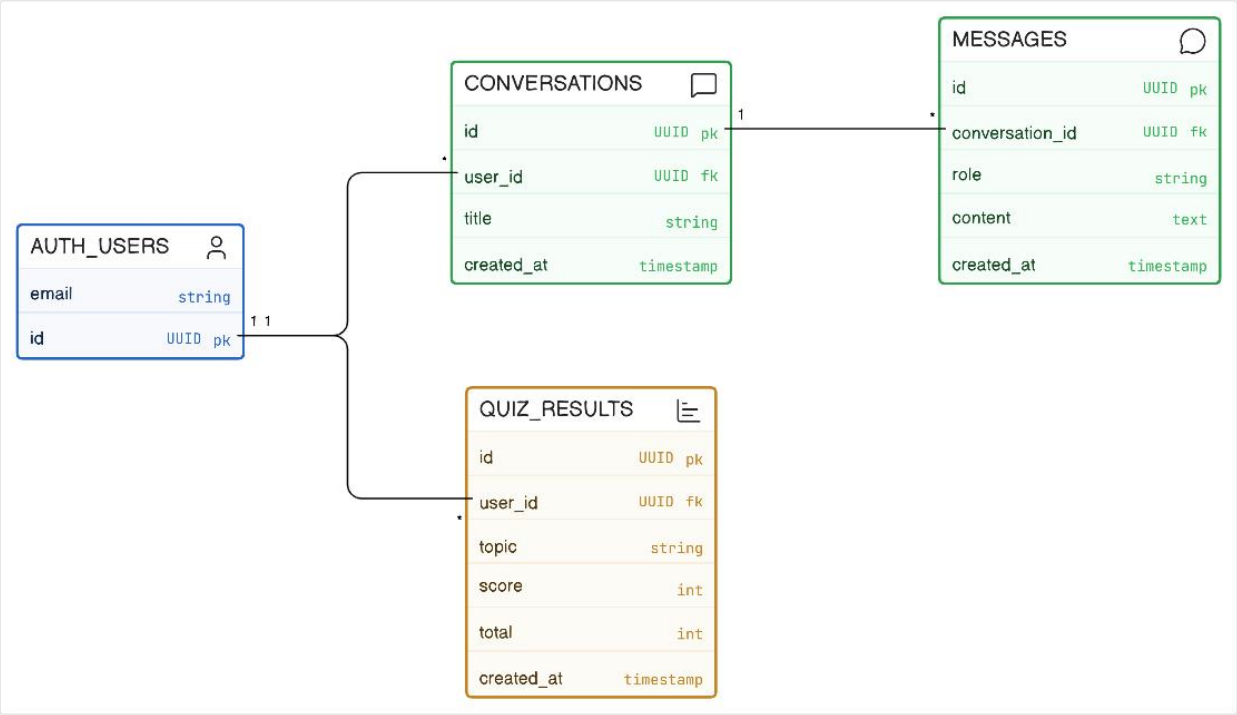


Рисунок 2.3 – ER-діаграма зв'язків бази даних системи

Окремим і надзвичайно важливим аспектом проєктування бази даних стало забезпечення безпеки та ізоляції персональної інформації. Оскільки система зберігає навчальні дані різних студентів, архітектура передбачає використання вбудованого в PostgreSQL механізму безпеки на рівні рядків (Row Level Security —

RLS). До кожної з трьох публічних таблиць застосовуються політики доступу, які перевіряють автентифікаційний JWT-токен (`auth.uid()`) і дозволяють виконувати операції виключно з тими записами, де значення `user_id` збігається з ідентифікатором поточного авторизованого користувача. Це повністю унеможливлює ситуацію, за якої студент міг би отримати несанкціонований доступ до діалогів чи оцінок інших користувачів системи.

Таблиця 2.1 – Опис полів таблиці `conversations` (Діалоги)

Назва поля	Тип даних (PostgreSQL)	Призначення	Ключ
id	UUID	Унікальний ідентифікатор діалогу	PK (Первинний)
user_id	UUID	Ідентифікатор автора (зв'язок із `auth.users`)	FK (Зовнішній)
title	Text	Коротка назва діалогу для бічної панелі	-
created_at	Timestampz	Дата та час створення діалогу	-

Таблиця 2.2 – Опис полів таблиці `messages` (Повідомлення)

Назва поля	Тип даних (PostgreSQL)	Призначення	Ключ
id	UUID	Унікальний ідентифікатор повідомлення	PK (Первинний)
conversation_id	UUID	Прив'язка до конкретного діалогу	FK (Зовнішній)
role	Text	Роль відправника (`user` або `assistant`)	-

content	Text	Текстове наповнення повідомлення (вкл. Markdown)	-
created_at	Timestampz	Дата та час відправки повідомлення	-

Таблиця 2.3 – Опис полів таблиці `quiz_results` (Результати тестів)

Назва поля	Тип даних (PostgreSQL)	Призначення	Ключ
id	UUID	Унікальний ідентифікатор результату	PK (Первинний)
user_id	UUID	Ідентифікатор студента (зв'язок із `auth.users`)	FK (Зовнішній)
topic	Text	Тема згенерованого тесту	-
score	Integer (int4)	Кількість правильних відповідей	-
total	Integer (int4)	Загальна кількість запитань у тесті	-
created_at	Timestampz	Дата та час проходження тесту	-

2.6 Логіка роботи AI-помічника

Інтелектуальний навчальний помічник є обчислювальним та логічним ядром системи. Він забезпечує взаємодію користувача зі знаннями предметної області, формує відповіді на запитання та генерує освітні матеріали. Логіка його роботи вибудовується як багатоетапний конвеєр обробки даних, що базується на використанні великої мовної моделі (LLM) у поєднанні з алгоритмами збереження контексту та елементами генерації, доповненої пошуком (RAG). Цей процес охоплює кілька стадій: від отримання тексту до визначення інтенту запиту,

формування контекстного вікна і генерації потокового результату на клієнтський пристрій.

Взаємодія розпочинається з первинної обробки текстового повідомлення від студента. Отримана строка проходить базове очищення, нормалізацію формату та автоматичне визначення мови. На цьому етапі система здійснює семантичний аналіз для визначення типу запиту. Алгоритм класифікує, чи є поточне звернення пояснювальним, практичним або тестовим. Від цієї класифікації залежить подальша маршрутизація запиту.

Оскільки великі мовні моделі є системами без збереження внутрішнього стану (stateless), етапом логіки є динамічне формування контексту. Система використовує базу даних для витягування історії попередніх повідомлень поточного діалогу. Це дозволяє моделі утримувати нитку розмови. Для підвищення точності відповідей логіка розширюється механізмом RAG. Його суть полягає у пошуку релевантних фрагментів у базі знань. Знайдені матеріали додаються до прихованого контексту, завдяки чому мовна модель генерує відповідь, спираючись на перевірені джерела (рис. 2.4).

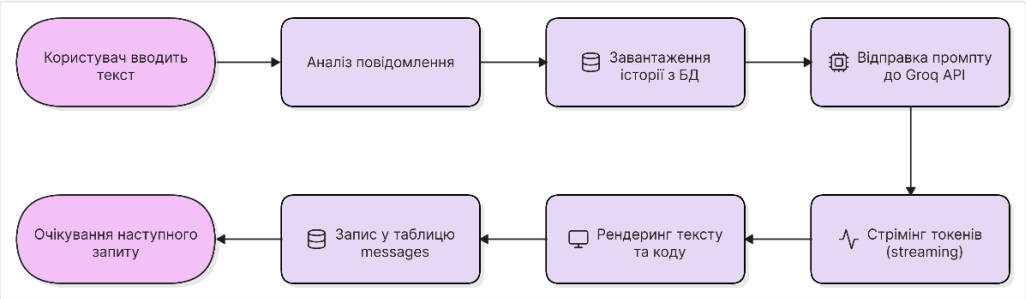


Рисунок 2.4 – UML-діаграма активності обробки запиту AI-помічником

Наступним етапом є генерація відповіді через інтеграцію із зовнішнім API. У сформований пакет даних входить системна інструкція (System Prompt), повідомлення користувача та історія діалогу. Системна інструкція зобов'язує штучний інтелект відповідати українською мовою, використовувати аналогії та наводити приклади програмного коду. Залежно від виявленого типу запиту, логіка помічника адаптується: у пояснювальному режимі генерується текст; у кодовому —

фокус зміщується на синтаксичну правильність коду; у тестовому — генеруються структуровані об'єкти JSON.

Фіксація історії чатів та результатів тестів дозволяє повторно використовувати контекст і формувати базу для аналізу успішності. Алгоритми роботи враховують технологічні обмеження: залежність від інтернет-з'єднання, швидкість відгуку API мовної моделі та апаратне обмеження розміру контекстного вікна (ліміт токенів).

2.7 Режим генерації тестів

Функціональною можливістю інформаційної системи є режим автоматичної генерації тестових завдань. Цей інструмент призначений для перевірки рівня знань студентів, закріплення навчального матеріалу та забезпечення зворотного зв’язку. Концептуально режим базується на використанні LLM, яка аналізує задану тему і створює структуровані запитання. Після введення теми користувачем система автоматично формує набір завдань. Мета підходу полягає у створенні тестів, перевірці засвоєння матеріалу та формуванні пояснень до відповідей (рис. 2.5).

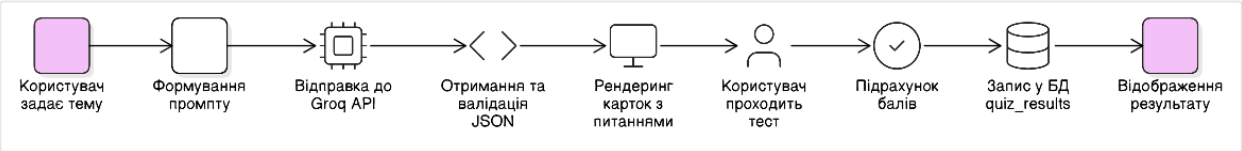


Рисунок 2.5 – Діаграма активності генерації та проходження тесту

Процес генерації тестів реалізується як алгоритм взаємодії клієнтського застосунку з хмарним API. Ініціалізація відбувається шляхом отримання теми тесту, після чого система формує запит до мовної моделі, генерує набір питань, здійснює парсинг відповіді, відображає графічний інтерфейс тесту, збирає відповіді, оцінює результати та надає пояснення. Формування запиту відбувається за допомогою промпта з інструкцією щодо структури вихідних даних. Алгоритм вимагає створити 5 питань, 4 варіанти відповіді (з дистракторами), зазначити індекс

правильного рішення та дидактичне пояснення. Вимогою є повернення відповіді виключно у машиночитаному форматі JSON.

Отриманий масив даних має визначену внутрішню структуру. У графічному інтерфейсі цей режим реалізовано у вигляді серії карток, де студент бачить питання та набір кнопок із варіантами відповідей.

Після завершення проходження тесту система переходить до автоматичного оцінювання. Алгоритм порівнює обрані варіанти з правильними індексами, підраховує кількість балів та формує результат. Система оцінювання базується на підрахунку: від 0 до 5 правильних відповідей. Також модуль відображає пояснення до кожного питання. Це дозволяє здобувачу освіти зрозуміти логіку правильного результату та проаналізувати помилки.

2.8 Інтерфейс користувача

Інтерфейс користувача є компонентом інформаційної системи, який виступає точкою контакту між студентом та AI-помічником. Від ергономічності інтерфейсу залежить якість користувацького досвіду. Клієнтську частину спроєктовано за моделлю односторінкового вебзастосунку (Single Page Application), що забезпечує динамічне оновлення контенту без перезавантаження сторінки. Візуальна концепція побудована навколо сучасних чат-платформ.

Взаємодія починається з екрана привітання (Landing Page) (рис. 2.6), який ознайомлює відвідувача з можливостями платформи.

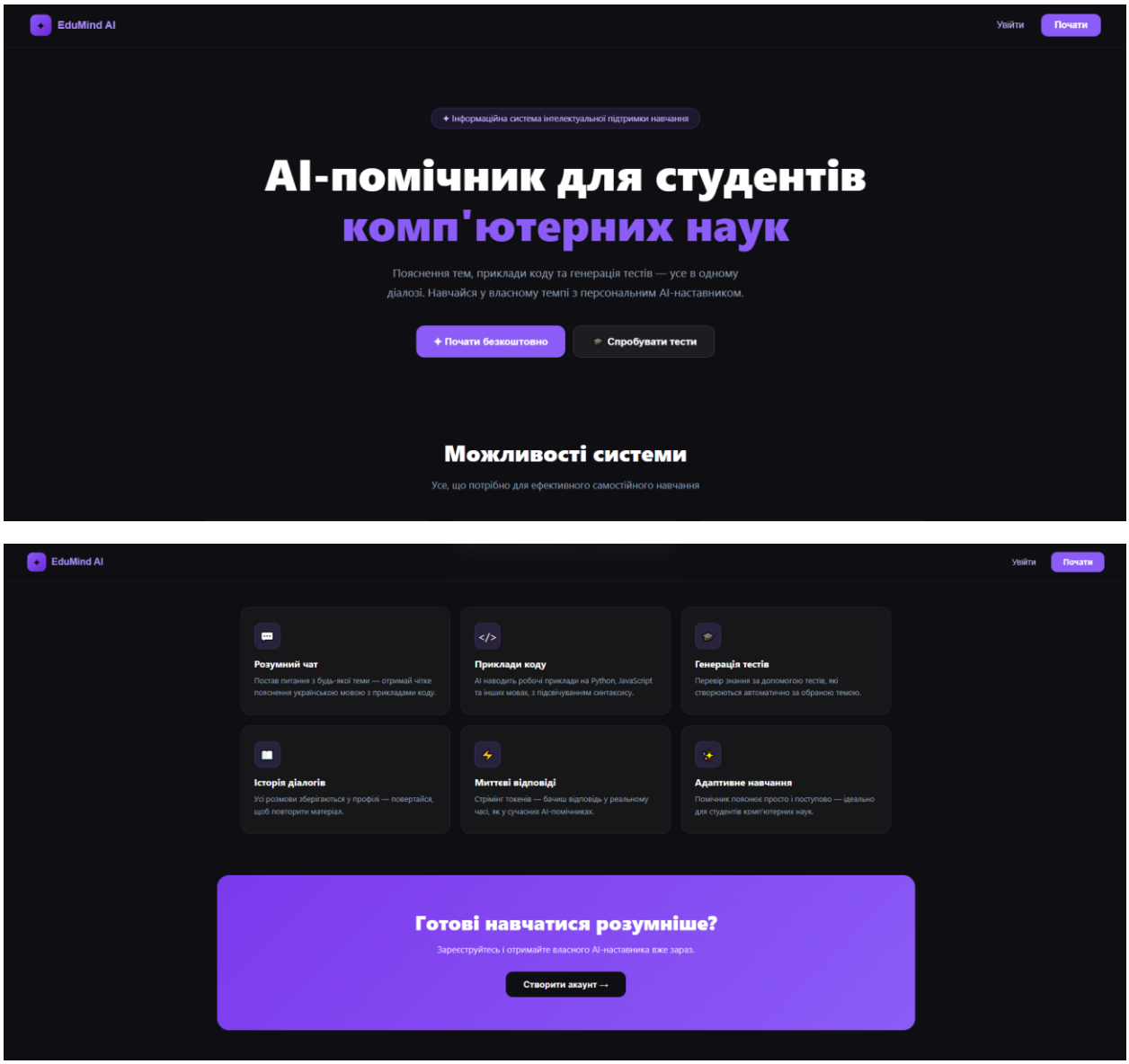


Рисунок 2.6 – Головна сторінка (Landing Page) вебзастосунку EduMind AI

Для переходу до навчального середовища користувач взаємодіє з модулем авторизації, який дозволяє створити обліковий запис або увійти до існуючого. Цей етап є обов'язковим для збереження історії діалогів та прив'язки результатів тестів до профілю.

Елементом архітектури інтерфейсу є робочий простір чату (рис. 2.7). Ліву частину екрана займає бічна панель (Sidebar), де відображається історія сесій. Основна область відведена під рендеринг повідомлень, які розділені за ролями. Інтерфейс підтримує форматування: генерацію списків, виділення термінів та рендеринг блоків програмного коду з підсвічуванням синтаксису. У нижній частині

розташовано поле введення багаторядкового тексту. Важливою складовою інтерфейсу є реалізація потокового виводу відповідей (streaming). Користувач спостерігає за генерацією тексту в режимі реального часу. Це мінімізує відчуття затримки та підвищує рівень інтерактивності.

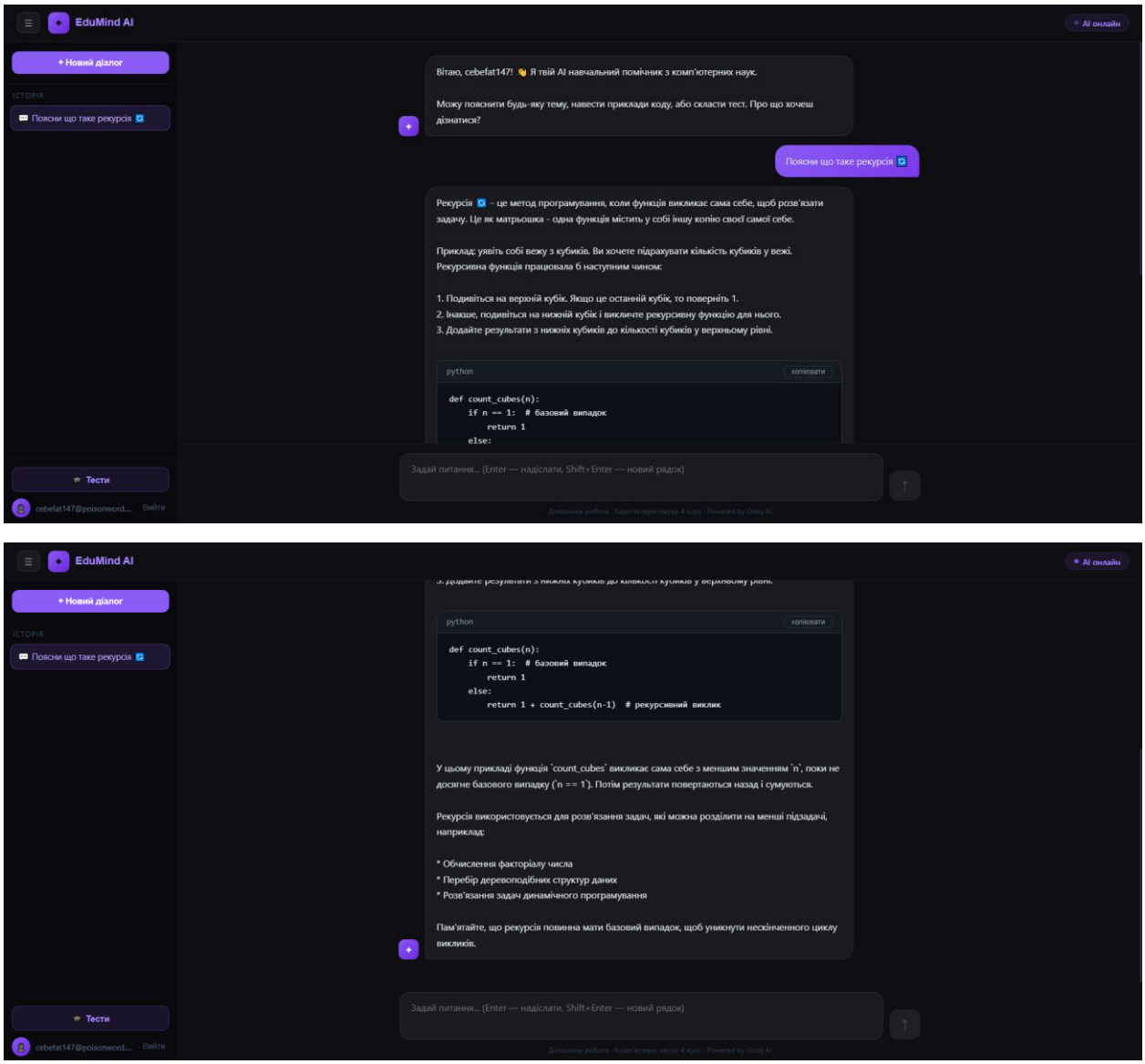


Рисунок 2.7 – Інтерфейс інтелектуального чату з AI-помічником

Окрім діалогового вікна, в інтерфейс інтегровано модуль генерації тестів (рис. 2.8). Після введення теми система трансформує JSON-об'єкт у графічний формат: серію карток із запитаннями. Після опитування інтерфейс перебудовується, відображаючи бал, підсвічування правильних та хибних виборів і розгорнуті пояснення.

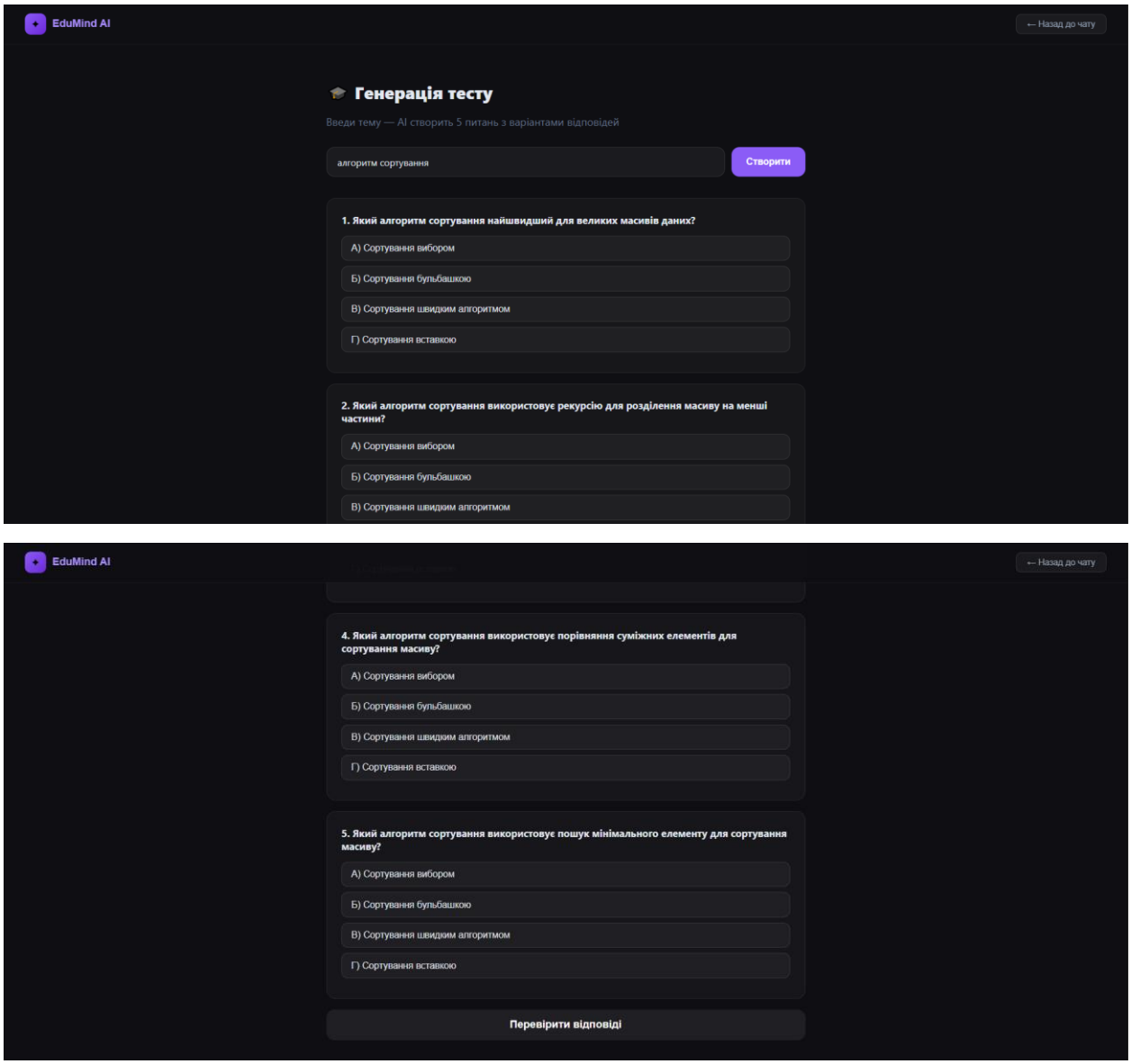


Рисунок 2.8 – Інтерфейс модуля генерації та проходження тестів

Графічний інтерфейс системи спроєктовано за принципами UI/UX дизайну. Адаптивна верстка дозволяє вебзастосунку підлаштовуватися під роздільну здатність екрана, забезпечуючи роботу на комп'ютерах та мобільних пристроях.

2.9 Програмна реалізація системи та структура коду

Програмна реалізація інформаційної системи інтелектуальної підтримки навчання спрямована на створення сучасного, високопродуктивного вебзастосунку, який забезпечує безперервну взаємодію користувача з AI-помічником у режимі

реального часу. З технічної точки зору продукт реалізовано як Single Page Application (SPA) на базі клієнт-серверної архітектури. Технологічним фундаментом клієнтської частини виступає бібліотека React.js у поєднанні з сучасним стандартом мови JavaScript (ES6+), що дозволяє будувати реактивні компонентні інтерфейси без необхідності перезавантаження сторінок. Для забезпечення функцій збереження даних та автентифікації інтегровано хмарну платформу Supabase, а логіка штучного інтелекту реалізується через взаємодію з великою мовною моделлю за допомогою Groq API. Вибір саме такого стеку технологій зумовлений його модульністю, високою швидкістю виконання коду та можливістю легкого масштабування функціоналу у майбутньому.

Структура програмного коду базується на суворому компонентному підході, де кожен функціональний блок ізольований і відповідає за окрему частину бізнес-логіки. Головним вузлом маршрутизації застосунку виступає кореневий компонент `App`. У ньому реалізовано управління глобальним станом (наприклад, через хук `useState`), який визначає поточний активний екран користувача: головну маркетингову сторінку (`landing`), сторінку авторизації (`auth`), основний робочий простір чату (`chat`) або модуль тестування (`quiz`). При ініціалізації застосунку в цьому ж компоненті відбувається перевірка активної сесії користувача через підключення клієнта Supabase. Додатково налаштовано прослуховування подій зміни стану авторизації (`onAuthStateChange`), що дозволяє системі миттєво перенаправляти неавторизованих відвідувачів на сторінку входу, гарантуючи безпеку навчальних даних.

Ключовим елементом системи, що займає центральне місце у програмній реалізації, є компонент чат-інтерфейсу `ChatPage`. Програмно він складається з кількох взаємопов'язаних блоків: області відображення повідомлень, адаптивного поля введення тексту, бічного меню історії діалогів (`Sidebar`) та панелі швидких підказок. Стан цього компонента динамічно керується набором React-хуків, які відстежують масив поточних повідомлень (`messages`), введений користувачем текст (`input`), індикатор завантаження (`loading`), процес потокової генерації

(`streaming`) та ідентифікатор активного діалогу (`activeConvId`). Для візуального розділення реплік застосовується умовний рендеринг: повідомлення з роллю `user` вирівнюються по правому краю екрана і мають один візуальний стиль, тоді як відповіді з роллю `assistant` вирівнюються по лівому краю.

Особливу увагу в коді приділено обробці текстового контенту, зокрема програмного коду, який генерує мовна модель. Оскільки система орієнтована на студентів технічних спеціальностей, отримані від API відповіді проходять через спеціальний парсер (компонент `MessageContent`). Цей модуль використовує регулярні вирази для пошуку блоків, виділених синтаксисом Markdown (наприклад, потрійними зворотними апострофами). Виявлений код автоматично передається до дочірнього компонента `CodeBlock`, який забезпечує його відображення моноширинним шрифтом, виконує логічне підсвічування синтаксису обраної мови програмування (у стилістиці сучасних IDE) та надає користувачу зручну кнопку для миттєвого копіювання фрагмента в буфер обміну.

Найбільш технічно складною частиною реалізації є алгоритм обробки запитів та потокової генерації відповідей (Streaming). Коли користувач ініціює відправку повідомлення, функція-обробник виконує низку асинхронних операцій. Спочатку нове повідомлення додається до локального масиву стану. Далі система формує HTTP-запит до Groq API, куди передає не лише поточний текст, але й заздалегідь визначені глобальні константи — системний промпт (`CHAT_PROMPT`). Цей промпт на програмному рівні змушує мовну модель відповідати виключно українською мовою та дотримуватися педагогічного стилю викладення матеріалу. Для реалізації ефекту "живого діалогу" запит використовує `fetch` API з читанням потоку через об'єкт `reader`. Замість очікування повного завершення генерації, система отримує відповідь дрібними токенами і поступово оновлює стан `streaming`. Це запускає безперервний перерендеринг інтерфейсу, завдяки якому користувач бачить анімацію набору тексту в реальному часі.

Паралельно з відображенням даних на клієнті, програмний код забезпечує надійне збереження історії взаємодії у базі даних. Після завершення генерації

потокової відповіді, система виконує асинхронні виклики до таблиць `conversations` та `messages` у Supabase. Якщо діалог ініційовано вперше, створюється новий запис із прив'язкою до ідентифікатора користувача, після чого записується пара повідомлень (запит і відповідь). Загальний користувацький досвід (UX/UI) посилюється завдяки використанню темної теми оформлення з глибокими фіолетовими акцентами, плавними анімаціями появи блоків та функцією автоматичного скролінгу до останнього повідомлення. Така продумана модульна структура програмного коду не лише забезпечує високу стабільність роботи системи, але й повністю відповідає сучасним інженерним стандартам веброзробки.

2.10 Реалізація підключення до мовної моделі (LLM API)

Реалізація підключення до великої мовної моделі є найбільш критичним етапом розробки інтелектуального ядра інформаційної системи. Оскільки обчислювальні потужності, необхідні для локального розгортання сучасних нейронних мереж, є надзвичайно високими, було прийнято рішення про використання хмарної мовної моделі через прикладний програмний інтерфейс (API). Як основний провайдер послуг штучного інтелекту було обрано платформу Groq, а як базову модель — `llama-3.3-70b-versatile`. Вибір платформи Groq зумовлений використанням спеціалізованих тензорних процесорів (LPU — Language Processing Units), які забезпечують наднизьку затримку (latency) та рекордно високу швидкість генерації токенів. Це є визначальним фактором для освітніх вебзастосунків, де користувач очікує миттєвої реакції на свої запити.

З технічної точки зору, взаємодія з мовною моделлю реалізована за принципами REST-архітектури через стандартні мережеві протоколи. Клієнтський застосунок використовує вбудований у браузер інтерфейс `fetch` для ініціалізації асинхронних HTTP-запитів методом POST до віддаленого сервера генерації. Для забезпечення безпеки та авторизації доступу до ресурсів API використовується механізм передачі секретного ключа (Bearer Token) у заголовках HTTP-запиту (Headers). Оскільки застосунок передає дані у форматі JSON, у заголовках також

обов'язково вказується тип контенту `application/json`. Формування корисного навантаження (payload) для відправки на сервер відбувається динамічно під час кожної транзакції та включає не лише текст користувача, але й масив історії попередніх повідомлень, що дозволяє моделі утримувати контекст тривалого діалогу.

Важливим аспектом реалізації є тонке налаштування гіперпараметрів мовної моделі. У програмному коді системи для моделі `llama-3.3-70b-versatile` встановлено параметр температури генерації (temperature) на рівні 0.5. Цей показник є оптимальним балансом для освітнього застосунку: він залишає моделі достатньо креативності для створення цікавих аналогій та зрозумілих пояснень, але водночас мінімізує ймовірність фактологічних помилок (галюцинацій) при генерації суворих алгоритмічних чи математичних концепцій. Додатково встановлено ліміт `max_tokens` на рівні 1500, що гарантує отримання розгорнутих відповідей та повних лістингів програмного коду без раптового обривання генерації на півслові.

Управління поведінкою штучного інтелекту реалізується за допомогою методів інженерії промптів (Prompt Engineering). У коді застосунку створено глобальну константу системного промпта (`CHAT_PROMPT`), яка передається моделі з роллю `system` на самому початку кожного масиву повідомлень. Ця інструкція жорстко позиціонує модель як "AI навчального помічника для студентів напряму Комп'ютерні науки". Системний промпт примусово встановлює українську мову спілкування, вимагає пояснювати складні концепції просто і з аналогіями, а також зобов'язує використовувати спеціальну розмітку для виділення блоків програмного коду. Завдяки цьому AI не виходить за межі своєї академічної ролі. Для модуля генерації тестів використовується окремий динамічний промпт (`QUIZ_PROMPT`), який формується на основі введеної користувачем теми. Цей промпт налаштований таким чином, щоб повністю вимкнути розмовний стиль моделі та змусити її повернути виключно валідний JSON-об'єкт із п'ятьма питаннями, варіантами відповідей та детальними поясненнями правильних рішень.

Для створення максимального ефекту живого спілкування, процес отримання даних від LLM API реалізовано у режимі потокової передачі (streaming). Замість традиційного підходу, коли клієнт очікує повного завершення генерації відповіді на сервері (що може тривати кілька секунд), застосунок встановлює постійне з'єднання та отримує згенерований текст фрагментами (токенами) по мірі їх готовності. Програмний код зчитує потік даних через інтерфейс `ReadableStream`, декодує байти у текст і поступово додає нові символи до стану компонента React. Кожне оновлення стану викликає негайний перерендеринг користувацького інтерфейсу, завдяки якому текст "друкується" на екрані користувача в режимі реального часу. Такий підхід не лише радикально зменшує метрику Time To First Token (час до появи першого символу), але й суттєво покращує загальний користувацький досвід, роблячи взаємодію з системою плавною та інтерактивною.

2.11 Реалізація системи авторизації та збереження даних

Реалізація надійної системи авторизації та збереження персональних даних користувачів є невід'ємною складовою архітектури розроблюваної інформаційної системи. Оскільки освітній процес вимагає безперервності та можливості повертатися до раніше пройденого матеріалу, система повинна вміти ідентифікувати кожного студента та гарантувати безпечне збереження його навчальної історії. Для вирішення цього завдання у програмному коді застосунку інтегровано хмарну платформу Backend-as-a-Service — Supabase, яка надає готові інструменти для управління автентифікацією (Supabase Auth) та реляційну базу даних PostgreSQL. Взаємодія з цими сервісами реалізується через офіційну клієнтську бібліотеку `@supabase/supabase-js`, що дозволяє виконувати безпечні запити безпосередньо з клієнтського React-застосунку.

Процес ідентифікації та автентифікації користувачів інкапсульовано у спеціалізованому компоненті `AuthPage`. Цей модуль підтримує два режими

роботи: реєстрацію нового облікового запису (`signUp`) та вхід до існуючого (`signInWithPassword`). З технічної точки зору, під час реєстрації клієнтський застосунок передає введені користувачем електронну адресу та пароль до серверів Supabase, які виконують криптографічне хешування пароля і створюють новий запис у системній таблиці користувачів. Якщо операція проходить успішно, користувач отримує сповіщення про необхідність підтвердження електронної пошти. У разі успішного входу (логіну) сервер повертає криптографічно захищений JSON Web Token (JWT), який застосунок автоматично зберігає у локальній пам'яті браузера для підтримки активної сесії. Для забезпечення високого рівня користувацького досвіду, компонент авторизації містить вбудовану обробку помилок: будь-які відхилення (наприклад, невірний пароль або існуючий email) перехоплюються блоком `catch` і негайно виводяться на екран у вигляді зрозумілих повідомлень.

Управління станом сесії реалізовано на найвищому рівні ієрархії компонентів — у кореневому модулі `App`. Під час монтування застосунку у браузері спрацьовує хук `useEffect`, який асинхронно запитує поточну сесію користувача через метод `getSession()`. Важливою архітектурною перевагою є використання слухача подій `onAuthStateChange`. Цей метод працює як глобальний перехоплювач (guard): він безперервно відстежує життєвий цикл токена і автоматично оновлює стан React-застосунку (`user` та `page`). Якщо користувач успішно авторизувався, система маршрутизації миттєво приховує маркетингову сторінку (`LandingPage`) і монтує основний робочий простір (`ChatPage`). У випадку виклику функції виходу (`signOut`) або завершення терміну дії токена, стан користувача очищується, і застосунок примусово повертає його на екран привітання, надійно захищаючи приватні дані від несанкціонованого доступу.

Логіка збереження історії навчальних діалогів глибоко інтегрована у функціонал компонента `ChatPage`. У системі реалізовано механізм динамічного створення чат-сесій. Коли студент відправляє своє перше повідомлення, програмний код перевіряє наявність ідентифікатора активного діалогу

(activeConvId). Якщо він відсутній, система ініціює створення нової розмови. Для забезпечення зручної навігації у бічному меню, застосунок автоматично генерує назву (title) для нового чату, витягуючи перші 50 символів із початкового запиту користувача за допомогою методу slice(0, 50). Після успішного виконання операції insert до таблиці conversations, система отримує унікальний ідентифікатор нового діалогу і використовує його як зовнішній ключ для збереження як повідомлення користувача, так і подальшої згенерованої відповіді штучного інтелекту в таблицю messages.

Окрім діалогів, система також реалізує персистентне збереження результатів контролю знань. У компоненті QuizPage, після завершення тестування та автоматичного підрахунку правильних відповідей, викликається функція запису до таблиці quiz_results. Застосунок формує об'єкт, що містить ідентифікатор користувача (user_id), тему тесту (topic), кількість набраних балів (score) та загальну кількість питань (total), після чого відправляє ці дані до бази. Це дозволяє формувати індивідуальну статистику успішності студента у процесі засвоєння курсу.

Варто окремо відзначити механізми безпеки та захисту даних, що застосовуються в системі. Завдяки архітектурі Supabase, клієнтський код ніколи не працює з паролями у відкритому вигляді. Крім того, всі запити до бази даних (як на читання select, так і на запис insert чи видалення delete) використовують фільтрацію за параметром user_id. Це доповнюється налаштованими на сервері політиками безпеки на рівні рядків (Row Level Security), які автоматично перевіряють JWT-токен при кожному зверненні до API. Таким чином, навіть якщо зловмисник спробує підмінити запит у браузері, ядро бази даних відхилить транзакцію, оскільки політики RLS математично гарантують, що користувач має фізичний доступ виключно до тих записів, де його унікальний ідентифікатор співпадає з криптографічним підписом сесії. Це забезпечує корпоративний рівень захисту конфіденційної навчальної інформації у рамках застосунку.

2.12 Тестування системи та аналіз результатів

Тестування є завершальним і критично важливим етапом життєвого циклу розробки інформаційної системи, спрямованим на перевірку відповідності реалізованого програмного продукту початковим вимогам, виявлення прихованих дефектів та оцінку загальної продуктивності. Головною метою тестування розробленого вебзастосунку EduMind AI було підтвердження коректності роботи клієнт-серверної архітектури, стабільності інтеграції з хмарною базою даних Supabase та надійності потокової взаємодії з великою мовною моделлю через Groq API. Стратегія перевірки охоплювала кілька рівнів: функціональне тестування інтерфейсу (UI), перевірку логіки управління станом (State Management) та стрес-тестування механізмів обробки зовнішніх API-запитів.

Першим етапом стало функціональне тестування модуля авторизації та ізоляції користувацьких даних, реалізованого у компоненті `AuthPage`. Було змодельовано базові сценарії реєстрації нового облікового запису та входу до системи. Тестування підтвердило, що система коректно валідує вхідні дані, а при спробі введення невірних облікових даних успішно перехоплює помилку через блок `catch` і відображає відповідне повідомлення в інтерфейсі (наприклад, про невірний пароль). Ключовим аспектом перевірки стала робота хука `onAuthStateChange` у кореневому компоненті `App`. Тести показали, що застосунок миттєво реагує на зміну статусу JWT-токена, безперебійно перемикаючи користувача між маркетинговою сторінкою (`LandingPage`) та робочим простором чату (`ChatPage`). Перевірка бази даних Supabase підтвердила, що політики безпеки на рівні рядків (RLS) працюють коректно: кожен користувач бачить лише власні діалоги та повідомлення, які фільтруються за параметром `user_id`.

Наступним і найбільш об'ємним етапом стало тестування інтелектуального чат-інтерфейсу та механізму потокової генерації відповідей (Streaming). Для перевірки комунікації з Groq API було ініційовано серію запитів різної складності, використовуючи масив заздалегідь підготовлених підказок (Suggestions), таких як

«Поясни що таке рекурсія» та «Що таке Big O нотація». Тестування функції `send()` підтвердило, що застосунок успішно формує контекст, додаючи системний промпт та історію попередніх повідомлень. Особливу увагу було приділено роботі інтерфейсу `ReadableStream` та об'єкта `TextDecoder`. Результати показали, що клієнтська частина безперебійно зчитує токени у форматі "Server-Sent Events" (data: ...), оновлює стан `streaming` і плавно рендерить текст на екрані. При цьому хук `useEffect`, прив'язаний до посилання `bottomRef`, успішно виконує функцію автоматичного скролінгу (`scrollIntoView`) донизу екрана під час генерації довгої відповіді, що забезпечує ідеальний користувацький досвід.

Окремо було протестовано модуль парсингу розмітки Markdown, який відповідає за коректне відображення програмного коду в чаті. Перевірка компонента `MessageContent` із застосуванням регулярних виразів показала 100% точність виявлення блоків коду у відповідях AI. Згенерований код успішно ізолюється у компонент `CodeBlock`, зберігає моноширинне форматування та коректно обробляє натискання кнопки «копіювати» (взаємодія з `navigator.clipboard.writeText`), після чого візуальний індикатор тимчасово змінює свій стан на «скопійовано».

Важливим кроком стала перевірка стабільності модуля генерації тестів (`QuizPage`). Це найбільш вразливий компонент системи, оскільки він покладається на те, що мовна модель поверне суворо валідний формат JSON. Тестування проводилося шляхом введення нестандартних тем для перевірки реакції системного промпта `QUIZ_PROMPT`. Аналіз результатів показав, що модель LLaMA-3.3 стабільно генерує об'єкти з масивом питань, варіантами відповідей та індексами правильних рішень. Вбудований у код обробник успішно очищує отриманий текст від зайвих Markdown-тегів (`replace(/```json|```/g, "")`) та парсить його через `JSON.parse`. Інтерактивне проходження згенерованого тесту підтвердило правильність алгоритму підрахунку балів та їх подальшого запису до таблиці `quiz_results` у базі даних Supabase (табл. 2.4).

Таблиця 2.4 – Сценарії функціонального тестування системи EduMind AI

№	Функція / Модуль	Вхідні дані або Дія	Очікуваний результат	Фактичний результат	Статус
1	Авторизація	Введення валідного email та пароля, натискання "Увійти"	Успішна аутентифікація, перенаправлення на сторінку чату	Система надає доступ, генерується JWT-токен	Успішно
2	Ізоляція даних (RLS)	Авторизація під тестовим користувачем №2	Відображення діалогів лише користувача №2	Чужі діалоги відсутні, RLS-політики працюють коректно	Успішно
3	Генерація відповіді	Введення запиту: "Що таке рекурсія?"	Поява індикатора AI, ініціалізація потоку даних (streaming)	Текст плавно відображається на екрані по токенах	Успішно
4	Форматування коду	Прохання навести приклад на мові Python	Виділення коду моноширинним шрифтом, підсвічування синтаксису	Код відрендерено у компоненті CodeBlock із кнопкою копіювання	Успішно
5	Генерація тесту	Введення теми: "Алгоритми сортування"	Повернення JSON, відображення 5 інтерактивних питань	Тест успішно згенеровано, доступний вибір відповідей	Успішно
6	Оцінювання тесту	Вибір варіантів і натискання "Перевірити"	Підрахунок балів, відображення пояснень, збереження в БД	Бали розраховано правильно (напр., 4/5), дані збережено в Supabase	Успішно

Заключний аналіз продуктивності та обробки помилок довів високу стійкість (resilience) розробленої системи. У разі імітації втрати інтернет-з'єднання або недоступності сервера Groq API, клієнтський застосунок не зазнає критичного збою (crash), а коректно зупиняє індикатор завантаження (`loading`) та виводить повідомлення «Помилка...», зберігаючи вже існуючий контекст розмови. Оцінка швидкодії користувацького інтерфейсу підтвердила плавність відтворення CSS-анімацій (таких як `bounce` для індикатора друку AI, `fadeIn` для появи повідомлень та `pulse` для статусу підключення) навіть на пристроях із низькою обчислювальною потужністю. Загальний аналіз результатів тестування дозволяє зробити висновок, що інформаційна система інтелектуальної підтримки навчання повністю відповідає поставленим завданням, працює стабільно і готова до дослідної експлуатації у реальному освітньому процесі.

Висновки до розділу:

У другому розділі здійснено проєктування та програмно-технічну реалізацію інформаційної системи інтелектуальної підтримки навчання. Обґрунтовано вибір технологічного стеку та спроєктовано розподілену клієнт-серверну архітектуру вебзастосунку з використанням бібліотеки React.js, хмарної платформи Supabase та сервісу Groq API. Розроблено логічну модель реляційної бази даних із застосуванням політик безпеки на рівні рядків (RLS) для ізольованого збереження історії діалогів та результатів тестування користувачів. Детально описано алгоритми роботи AI-помічника, зокрема використання системних промптів, реалізацію потокової генерації відповідей (streaming) та модуль динамічного формування інтерактивних тестів у машиночитаному форматі JSON. Проведено комплексне тестування розробленої системи за визначеними сценаріями, результати якого підтвердили її стабільність, високу швидкодію та повну відповідність поставленим завданням.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено інформаційну систему інтелектуальної підтримки навчання на основі AI-помічника, яка призначена для допомоги студентам у процесі навчання, засвоєння теоретичного матеріалу та перевірки знань. Основною метою роботи було створення сучасного програмного рішення, що поєднує технології штучного інтелекту, веб-розробки та хмарних сервісів для підвищення ефективності освітнього процесу.

У першому розділі було проведено аналіз предметної області, зокрема розглянуто поняття інформаційних систем у сфері освіти, особливості інтелектуальних навчальних систем та існуючі програмні рішення. Було встановлено, що сучасні освітні платформи часто не забезпечують достатнього рівня персоналізації навчання. Окрему увагу було приділено аналізу систем на зразок ChatGPT та GitHub Copilot, що дозволило сформулювати вимоги до власної системи, орієнтованої виключно на освітній процес.

У другому розділі було виконано проєктування та практичну реалізацію системи. Розроблено загальну архітектуру програмного забезпечення, яка включає клієнтську частину (React-додаток), серверну взаємодію через Groq API та хмарну базу даних Supabase. Було реалізовано повноцінний веб-застосунок із сучасним користувацьким інтерфейсом, який включає стартову сторінку, систему авторизації, чат з потоковою генерацією відповідей (streaming) та модуль автоматичної генерації тестів у форматі JSON.

Також у другому розділі було проведено комплексне тестування розробленої системи. Перевірка охоплювала функціональне тестування, оцінку продуктивності та зручності інтерфейсу. Результати тестування показали, що система працює стабільно, всі основні функції виконуються коректно, а середній час відповіді AI є оптимальним для навчального застосування.

У цілому, запропоноване рішення поєднує у собі функції навчального помічника, генератора тестів та системи збереження навчальної історії. Перспективами подальшого розвитку є розширення функціоналу, зокрема

додавання персоналізованих навчальних траєкторій та інтеграції з університетськими LMS. Таким чином, поставлену мету дипломної роботи було досягнуто, а всі завдання — виконано в повному обсязі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

3. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання. Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

4. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

5. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2021. 1152 p.

6. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p.

7. GPT-4 Technical Report / OpenAI. URL: *https://openai.com/research/gpt-4* (дата звернення: 22.04.2026).

8. Brown T. et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877–1901.

9. Vaswani A. et al. Attention Is All You Need. *NeurIPS*. 2017. P. 5998–6008.

10. Groq API Documentation / Groq Inc. URL: *https://console.groq.com/docs* (дата звернення: 22.04.2026).

11. Supabase Documentation / Supabase. URL: *https://supabase.com/docs* (дата звернення: 22.04.2026).

12. React – A JavaScript library for building user interfaces / React Documentation. URL: *https://react.dev/* (дата звернення: 22.04.2026).

13. JavaScript Guide / MDN Web Docs. URL: *https://developer.mozilla.org/* (дата звернення: 22.04.2026).

14. Node.js Documentation / Node.js Foundation. URL: *https://nodejs.org/en/docs* (дата звернення: 22.04.2026).

15. Introduction to AI-powered applications / Microsoft. URL: *https://learn.microsoft.com/* (дата звернення: 22.04.2026).

16. Vaswani A. et al. Transformers in Natural Language Processing: A Survey. *arXiv preprint arXiv:2106.04554*. 2021.

17. Goodfellow I. Generative Adversarial Networks. *arXiv:1406.2661*. 2014.

18. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS*. 2020.

19. Prompt Engineering Guide / OpenAI. URL: *https://platform.openai.com/docs* (дата звернення: 22.04.2026).

20. Authentication Guide / Supabase. URL: *https://supabase.com/docs/guides/auth* (дата звернення: 22.04.2026).

21. Web Application Architecture Basics / Mozilla. URL: *https://developer.mozilla.org/en-US/docs/Web/Architecture* (дата звернення: 22.04.2026).

22. CS221: Artificial Intelligence – Course Notes / Stanford University. URL: *https://stanford.edu* (дата звернення: 22.04.2026).

23. Goodfellow I. Deep Learning Book – Notes and Resources. URL: *https://www.deeplearningbook.org/* (дата звернення: 22.04.2026).

24. React Best Practices / React Community. URL: *https://react.dev/learn* (дата звернення: 22.04.2026).

ДОДАТОК А

Програмна реалізація ІС інтелектуальної підтримки навчання

Лістинг А.1 – Конфігурація системи та допоміжні компоненти інтерфейсу

```
import { useState, useRef, useEffect } from "react";
import { createClient } from "@supabase/supabase-js";

// ── KEYS ──────────────────────────────────────────────────────────
const SUPABASE_URL = "https://ttmnyiaebrtgejinolxx.supabase.co";
const SUPABASE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InR0bW55aWFlYnJ
0Z2VqaW5vbHh4Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NzY4NDgzNDUsImV4cCI6MjA5MjQyNDM0NX0.1Kvx
fhtwzPftrV3-jZM_5zpRK3bBc9nTrrPJAX0IHKc";
const GROQ_KEY = "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";

const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

const CHAT_PROMPT = `Ти — AI навчальний помічник для студентів напряму "Комп'ютерні
науки".
Відповідай українською мовою. Для коду використовуй блоки з позначенням мови.
Поясню просто, з аналогіями та прикладами. Будь дружнім але точним.
Ти є частиною дипломної роботи "Інформаційна система інтелектуальної підтримки
навчання".`;

const QUIZ_PROMPT = (topic) => `Створи тест з 5 питань на тему: "${topic}".
Відповідай ТІЛЬКИ у форматі JSON (без жодного тексту до або після):
{"questions":[{"q":"питання","options":["А) ...","Б) ...","В) ...","Г)
..."],"answer":0,"explanation":"пояснення"}]}
answer — індекс правильної відповіді (0-3). Мова: українська.`;

const SUGGESTIONS = [
  "Поясни що таке рекурсія 🔄",
  "Склади тест з алгоритмів 📝",
  "Що таке Big O нотація? ⏱️",
  "Стек і черга — різниця 🥞",
  "Двозв'язний список на Python 🐍",
  "Що таке REST API? 🌐",
];

const FEATURES = [
  { icon: "💬", title: "Розумний чат", desc: "Постав питання з будь-якої теми —
отримай чітке пояснення українською мовою з прикладами коду." },
  { icon: "</>", title: "Приклади коду", desc: "AI наводить робочі приклади на
Python, JavaScript та інших мовах, з підсвічуванням синтаксису." },
  { icon: "🎓", title: "Генерація тестів", desc: "Перевір знання за допомогою тестів,
які створюються автоматично за обраною темою." },
  { icon: "📖", title: "Історія діалогів", desc: "Усі розмови зберігаються у профілі
— повертайся, щоб повторити матеріал." },
```

```
  { icon: "⚡", title: "Миттєві відповіді", desc: "Стрімінг токенів — бачиш відповідь
у реальному часі, як у сучасних AI-помічниках." },
  { icon: "✨", title: "Адаптивне навчання", desc: "Помічник пояснює просто і
поступово — ідеально для студентів комп'ютерних наук." },
];

// ── HELPERS ──────────────────────────────────────────────────────
function CodeBlock({ code, lang }) {
  const [copied, setCopied] = useState(false);
  return (
    <div style={{ margin: "10px 0" }}>
      <div style={{ background: "#0d1117", borderRadius: 10, border: "1px solid
#30363d", overflow: "hidden" }}>
        <div style={{ display: "flex", justifyContent: "space-between", alignItems:
"center", padding: "6px 14px", background: "#161b22", borderBottom: "1px solid
#30363d" }}>
          <span style={{ color: "#8b949e", fontSize: 12, fontFamily: "monospace"
}}>{lang || "code"}</span>
          <button onClick={() => { navigator.clipboard.writeText(code);
setCopied(true); setTimeout(() => setCopied(false), 2000); }}
            style={{ background: "none", border: "1px solid #30363d", color: copied ?
"#a78bfa" : "#8b949e", borderRadius: 6, padding: "2px 10px", fontSize: 11, cursor:
"pointer", transition: "color 0.2s" }}>
            {copied ? "✓ скопійовано" : "копіювати"}
          </button>
        </div>
        <pre style={{ margin: 0, padding: "14px 18px", overflowX: "auto", fontFamily:
"monospace", fontSize: 13, lineHeight: 1.6, color: "#e6edf3" }}>{code}</pre>
      </div>
    </div>
  );
}

function MessageContent({ text }) {
  const parts = [];
  const codeRegex = /```(\w*)\n?([\s\S]*?)```/g;
  let last = 0, match;
  while ((match = codeRegex.exec(text)) !== null) {
    if (match.index > last) parts.push({ type: "text", content: text.slice(last,
match.index) });
    parts.push({ type: "code", lang: match[1], content: match[2].trim() });
    last = match.index + match[0].length;
  }
  if (last < text.length) parts.push({ type: "text", content: text.slice(last) });
  return (
    <div>
      {parts.map((p, i) =>
        p.type === "code"
          ? <CodeBlock key={i} code={p.content} lang={p.lang} />
```

```
          : <span key={i} style={{ whiteSpace: "pre-wrap", lineHeight: 1.7
}}>{p.content}</span>
      )}
    </div>
  );
}

function TypingDots() {
  return (
    <div style={{ display: "flex", gap: 5, padding: "12px 16px", alignItems: "center"
}}>
      <span style={{ color: "#64748b", fontSize: 12, marginRight: 4 }}>AI
друкує</span>
      {[0, 1, 2].map(i => (
        <div key={i} style={{ width: 6, height: 6, borderRadius: "50%", background:
"#a78bfa", animation: `bounce 1.2s ease-in-out ${i * 0.2}s infinite` }} />
      ))}
    </div>
  );
}

// ── LOGO BUTTON ──────────────────────────────────────────────
function Logo({ onClick }) {
  return (
    <button onClick={onClick} style={{ display: "flex", alignItems: "center", gap: 9,
background: "none", border: "none", cursor: "pointer", padding: 0 }}>
      <div style={{ width: 32, height: 32, borderRadius: 9, background: "linear-
gradient(135deg,#8b5cf6,#6d28d9)", display: "flex", alignItems: "center",
justifyContent: "center", fontSize: 15 }}>✦</div>
      <span style={{ fontWeight: 700, fontSize: 16, color: "#a78bfa" }}>EduMind
AI</span>
    </button>
  );
}
```

Лістинг А.2 – Реалізація модуля авторизації користувачів (AuthPage)

```
// ── AUTH PAGE ────────────────────────────────────────────────
function AuthPage({ onAuth, onBack }) {
  const [mode, setMode] = useState("login");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState("");
  const [success, setSuccess] = useState("");

  const handle = async () => {
    setError(""); setSuccess(""); setLoading(true);
    try {
      if (mode === "login") {
```

```
        const { data, error } = await supabase.auth.signInWithPassword({ email,
password });
        if (error) throw error;
        onAuth(data.user);
      } else {
        const { error } = await supabase.auth.signUp({ email, password });
        if (error) throw error;
        setSuccess("✅ Акаунт створено! Перевір пошту для підтвердження, потім
увійди.");
        setMode("login");
      }
    } catch (e) { setError(e.message); }
    setLoading(false);
  };

  return (
    <div style={{ minHeight: "100vh", background: "#0f0f13", display: "flex",
flexDirection: "column", fontFamily: "'Segoe UI', system-ui, sans-serif" }}>
      {/* Nav */}
      <nav style={{ display: "flex", justifyContent: "space-between", alignItems:
"center", padding: "16px 40px", borderBottom: "1px solid rgba(255,255,255,0.06)" }}>
        <Logo onClick={onBack} />
        <button onClick={onBack} style={{ background: "rgba(255,255,255,0.05)",
border: "1px solid rgba(255,255,255,0.1)", color: "#94a3b8", borderRadius: 8,
padding: "7px 16px", fontSize: 13, cursor: "pointer" }}>← Назад</button>
      </nav>

      <div style={{ flex: 1, display: "flex", alignItems: "center", justifyContent:
"center", padding: 20 }}>
        <div style={{ width: "100%", maxWidth: 420 }}>
          <div style={{ textAlign: "center", marginBottom: 32 }}>
            <div style={{ width: 52, height: 52, borderRadius: 14, background:
"linear-gradient(135deg,#8b5cf6,#6d28d9)", display: "flex", alignItems: "center",
justifyContent: "center", fontSize: 24, margin: "0 auto 16px" }}>✦</div>
            <h1 style={{ color: "#fff", fontSize: 26, fontWeight: 800, margin: "0 0
6px" }}>EduMind AI</h1>
            <p style={{ color: "#64748b", fontSize: 14, margin: 0 }}>Інтелектуальна
підтримка навчання</p>
          </div>

          <div style={{ background: "rgba(255,255,255,0.03)", border: "1px solid
rgba(255,255,255,0.08)", borderRadius: 18, padding: 32 }}>
            <div style={{ display: "flex", background: "rgba(255,255,255,0.05)",
borderRadius: 10, padding: 4, marginBottom: 24 }}>
              {["login", "register"].map(m => (
                <button key={m} onClick={() => { setMode(m); setError("");
setSuccess(""); }} style={{
                  flex: 1, padding: "8px", border: "none", borderRadius: 8, fontSize:
14, fontWeight: 600, cursor: "pointer", transition: "all 0.2s",
```

```
                  background: mode === m ? "#8b5cf6" : "transparent", color: mode ===
m ? "#fff" : "#64748b",
                }}>{m === "login" ? "Увійти" : "Реєстрація"}</button>
              ))}
            </div>

            {error && <div style={{ background: "rgba(239,68,68,0.1)", border: "1px
solid rgba(239,68,68,0.3)", borderRadius: 10, padding: "10px 14px", color: "#fca5a5",
fontSize: 13, marginBottom: 16 }}>{error}</div>}
            {success && <div style={{ background: "rgba(16,185,129,0.1)", border:
"1px solid rgba(16,185,129,0.3)", borderRadius: 10, padding: "10px 14px", color:
"#6ee7b7", fontSize: 13, marginBottom: 16 }}>{success}</div>}

            <div style={{ marginBottom: 14 }}>
              <label style={{ color: "#94a3b8", fontSize: 13, display: "block",
marginBottom: 6 }}>Email</label>
              <input type="email" value={email} onChange={e =>
setEmail(e.target.value)} placeholder="your@email.com"
                style={{ width: "100%", background: "rgba(255,255,255,0.05)", border:
"1px solid rgba(255,255,255,0.1)", borderRadius: 10, color: "#e2e8f0", fontSize: 14,
padding: "11px 14px", outline: "none", boxSizing: "border-box", fontFamily: "inherit"
}} />
            </div>
            <div style={{ marginBottom: 24 }}>
              <label style={{ color: "#94a3b8", fontSize: 13, display: "block",
marginBottom: 6 }}>Пароль</label>
              <input type="password" value={password} onChange={e =>
setPassword(e.target.value)} onKeyDown={e => e.key === "Enter" && handle()}
placeholder="••••••••"
                style={{ width: "100%", background: "rgba(255,255,255,0.05)", border:
"1px solid rgba(255,255,255,0.1)", borderRadius: 10, color: "#e2e8f0", fontSize: 14,
padding: "11px 14px", outline: "none", boxSizing: "border-box", fontFamily: "inherit"
}} />
            </div>

            <button onClick={handle} disabled={loading || !email || !password}
style={{
              width: "100%", background: email && password ? "linear-
gradient(135deg,#8b5cf6,#7c3aed)" : "rgba(255,255,255,0.07)",
              border: "none", borderRadius: 12, color: email && password ? "#fff" :
"#475569",
              fontSize: 15, fontWeight: 700, padding: "13px", cursor: email &&
password ? "pointer" : "default", transition: "all 0.2s",
            }}>
              {loading ? "⏳ Завантаження..." : mode === "login" ? "Увійти →" :
"Створити акаунт →"}
            </button>
          </div>
        </div>
      </div>
```

```
    </div>
  );
}
```

Лістинг А.3 – Реалізація модуля генерації тестів

```
// ── QUIZ PAGE ──────────────────────────────────────────────────
function QuizPage({ user, onBack, onHome }) {
  const [topic, setTopic] = useState("");
  const [loading, setLoading] = useState(false);
  const [questions, setQuestions] = useState(null);
  const [answers, setAnswers] = useState({});
  const [submitted, setSubmitted] = useState(false);
  const [score, setScore] = useState(0);
  const [error, setError] = useState("");

  const generate = async () => {
    if (!topic.trim()) return;
    setLoading(true); setError(""); setQuestions(null); setAnswers({});
setSubmitted(false);
    try {
      const res = await fetch("https://api.groq.com/openai/v1/chat/completions", {
        method: "POST",
        headers: { "Content-Type": "application/json", "Authorization": `Bearer
${GROQ_KEY}` },
        body: JSON.stringify({ model: "llama-3.3-70b-versatile", messages: [{ role:
"user", content: QUIZ_PROMPT(topic) }], max_tokens: 1500, temperature: 0.5 }),
      });
      const data = await res.json();
      if (data.error) throw new Error(data.error.message);
      let text = data.choices[0].message.content.trim().replace(/```json|```/g,
"").trim();
      setQuestions(JSON.parse(text).questions);
    } catch { setError("Помилка генерації тесту. Спробуй ще раз."); }
    setLoading(false);
  };

  const submit = async () => {
    let s = 0;
    questions.forEach((q, i) => { if (answers[i] === q.answer) s++; });
    setScore(s); setSubmitted(true);
    await supabase.from("quiz_results").insert({ user_id: user.id, topic, score: s,
total: questions.length });
  };

  return (
    <div style={{ minHeight: "100vh", background: "#0f0f13", color: "#e2e8f0",
fontFamily: "'Segoe UI',system-ui,sans-serif" }}>
      {/* Nav */}
```

```
      <nav style={{ display: "flex", justifyContent: "space-between", alignItems:
"center", padding: "14px 32px", borderBottom: "1px solid rgba(255,255,255,0.06)",
background: "rgba(15,15,19,0.97)", position: "sticky", top: 0, zIndex: 50 }}>
        <Logo onClick={onHome} />
        <button onClick={onBack} style={{ background: "rgba(255,255,255,0.05)",
border: "1px solid rgba(255,255,255,0.1)", color: "#94a3b8", borderRadius: 8,
padding: "7px 16px", fontSize: 13, cursor: "pointer" }}>← Назад до чату</button>
      </nav>

      <div style={{ maxWidth: 720, margin: "0 auto", padding: "32px 20px" }}>
        <h2 style={{ fontSize: 26, fontWeight: 800, marginBottom: 8 }}>🎓 Генерація
тесту</h2>
        <p style={{ color: "#64748b", marginBottom: 28 }}>Введи тему — AI створить 5
питань з варіантами відповідей</p>

        <div style={{ display: "flex", gap: 10, marginBottom: 32 }}>
          <input value={topic} onChange={e => setTopic(e.target.value)} onKeyDown={e
=> e.key === "Enter" && generate()}
            placeholder="Наприклад: алгоритми сортування, рекурсія, ООП..."
            style={{ flex: 1, background: "rgba(255,255,255,0.05)", border: "1px
solid rgba(255,255,255,0.1)", borderRadius: 12, color: "#e2e8f0", fontSize: 14,
padding: "12px 16px", outline: "none", fontFamily: "inherit" }} />
          <button onClick={generate} disabled={loading || !topic.trim()} style={{
            background: topic.trim() ? "#8b5cf6" : "rgba(255,255,255,0.07)", border:
"none", borderRadius: 12,
            color: topic.trim() ? "#fff" : "#475569", fontSize: 14, fontWeight: 600,
padding: "12px 22px", cursor: topic.trim() ? "pointer" : "default",
          }}>{loading ? "⏳ Генерую..." : "Створити"}</button>
        </div>

        {error && <div style={{ background: "rgba(239,68,68,0.1)", border: "1px solid
rgba(239,68,68,0.3)", borderRadius: 12, padding: 16, color: "#fca5a5", marginBottom:
20 }}>{error}</div>}

        {questions && (
          <div>
            {questions.map((q, i) => (
              <div key={i} style={{ background: "rgba(255,255,255,0.03)", border:
"1px solid rgba(255,255,255,0.08)", borderRadius: 14, padding: 22, marginBottom: 16
}}>
                <p style={{ fontWeight: 700, fontSize: 15, margin: "0 0 14px" }}>{i +
1}. {q.q}</p>
                {q.options.map((opt, j) => {
                  let bg = "rgba(255,255,255,0.04)", border =
"rgba(255,255,255,0.1)";
                  if (submitted) {
                    if (j === q.answer) { bg = "rgba(16,185,129,0.15)"; border =
"#10b981"; }
                    else if (answers[i] === j) { bg = "rgba(239,68,68,0.12)"; border
= "#ef4444"; }
```

```
                  } else if (answers[i] === j) { bg = "rgba(139,92,246,0.15)"; border
= "#8b5cf6"; }
                  return (
                    <button key={j} onClick={() => !submitted && setAnswers(a => ({
...a, [i]: j }))} style={{
                      display: "block", width: "100%", textAlign: "left", background:
bg, border: `1px solid ${border}`,
                      borderRadius: 10, color: "#e2e8f0", padding: "10px 14px",
marginBottom: 8, fontSize: 14,
                      cursor: submitted ? "default" : "pointer", transition: "all
0.2s",
                    }}>{opt}</button>
                  );
                })}
                {submitted && <div style={{ marginTop: 10, padding: "10px 14px",
background: "rgba(139,92,246,0.1)", borderRadius: 10, color: "#c4b5fd", fontSize: 13
}}>💡 {q.explanation}</div>}
              </div>
            ))}

            {!submitted ? (
              <button onClick={submit} disabled={Object.keys(answers).length <
questions.length} style={{
                width: "100%", background: Object.keys(answers).length >=
questions.length ? "#8b5cf6" : "rgba(255,255,255,0.07)",
                border: "none", borderRadius: 12, color: "#fff", fontSize: 16,
fontWeight: 700, padding: 14,
                cursor: Object.keys(answers).length >= questions.length ? "pointer" :
"default",
              }}>Перевірити відповіді</button>
            ) : (
              <div style={{ background: "linear-
gradient(135deg,rgba(139,92,246,0.2),rgba(109,40,217,0.2))", border: "1px solid
rgba(139,92,246,0.4)", borderRadius: 14, padding: 24, textAlign: "center" }}>
                <div style={{ fontSize: 40, marginBottom: 10 }}>{score >= 4 ? "🏆" :
score >= 3 ? "👍" : "📖"}</div>
                <h3 style={{ fontSize: 22, fontWeight: 800, margin: "0 0 8px"
}}>Результат: {score}/{questions.length}</h3>
                <p style={{ color: "#a78bfa", margin: "0 0 16px" }}>{score >= 4 ?
"Відмінно!" : score >= 3 ? "Добре!" : "Треба повторити матеріал"}</p>
                <button onClick={() => { setQuestions(null); setTopic("");
setAnswers({}); setSubmitted(false); }} style={{
                  background: "#8b5cf6", border: "none", borderRadius: 10, color:
"#fff", fontSize: 14, fontWeight: 600, padding: "10px 24px", cursor: "pointer",
                }}>Новий тест</button>
              </div>
            )}
          </div>
        )}
      </div>
```

```
    </div>
  );
}
```

Лістинг А.4 – Інтелектуальний чат-інтерфейс

```
// ── CHAT PAGE ──────────────────────────────────────────────────
function ChatPage({ user, onLogout, onQuiz, onHome }) {
  const [conversations, setConversations] = useState([]);
  const [activeConvId, setActiveConvId] = useState(null);
  const [messages, setMessages] = useState([{
    role: "assistant",
    content: `Вітаю, ${user.email.split("@")[0]}! 👋 Я твій AI навчальний помічник з комп'ютерних наук.\n\nМожу пояснити будь-яку тему, навести приклади коду, або скласти тест. Про що хочеш дізнатися?`,
  }]);
  const [input, setInput] = useState("");
  const [loading, setLoading] = useState(false);
  const [streaming, setStreaming] = useState("");
  const [sidebarOpen, setSidebarOpen] = useState(true);
  const [isScrolling, setIsScrolling] = useState(false);
  const bottomRef = useRef(null);
  const chatBoxRef = useRef(null);

  useEffect(() => { loadConversations(); }, []);

  // Auto-scroll to bottom while streaming or new message arrives
  useEffect(() => {
    if (streaming || loading) {
      bottomRef.current?.scrollIntoView({ behavior: "smooth" });
    }
  }, [streaming, loading]);

  useEffect(() => {
    bottomRef.current?.scrollIntoView({ behavior: "smooth" });
  }, [messages]);

  const loadConversations = async () => {
    const { data } = await supabase.from("conversations").select("*").eq("user_id", user.id).order("created_at", { ascending: false });
    setConversations(data || []);
  };

  const loadConversation = async (conv) => {
    setActiveConvId(conv.id);
    const { data } = await supabase.from("messages").select("*").eq("conversation_id", conv.id).order("created_at");
    setMessages(data || []);
  };
```

```
  const deleteConversation = async (id, e) => {
    e.stopPropagation();
    await supabase.from("conversations").delete().eq("id", id);
    if (activeConvId === id) { setActiveConvId(null); setMessages([{ role:
"assistant", content: "Розпочни новий діалог! 👋" }]); }
    loadConversations();
  };

  const send = async (text) => {
    const userText = (text || input).trim();
    if (!userText || loading) return;
    setInput("");

    const userMsg = { role: "user", content: userText };
    const newMessages = [...messages, userMsg];
    setMessages(newMessages);
    setLoading(true);
    setStreaming("");

    let convId = activeConvId;
    if (!convId) {
      const title = userText.slice(0, 50);
      const { data } = await supabase.from("conversations").insert({ user_id:
user.id, title }).select().single();
      convId = data.id;
      setActiveConvId(convId);
      loadConversations();
    }
    await supabase.from("messages").insert({ conversation_id: convId, role: "user",
content: userText });

    try {
      const res = await fetch("https://api.groq.com/openai/v1/chat/completions", {
        method: "POST",
        headers: { "Content-Type": "application/json", "Authorization": `Bearer
${GROQ_KEY}` },
        body: JSON.stringify({
          model: "llama-3.3-70b-versatile",
          messages: [{ role: "system", content: CHAT_PROMPT }, ...newMessages.map(m
=> ({ role: m.role, content: m.content }))],
          max_tokens: 1500, temperature: 0.7, stream: true,
        }),
      });

      const reader = res.body.getReader();
      const decoder = new TextDecoder();
      let fullReply = "";

      while (true) {
        const { done, value } = await reader.read();
```

```
        if (done) break;
        const lines = decoder.decode(value).split("\n").filter(l =>
l.startsWith("data: ") && l !== "data: [DONE]");
        for (const line of lines) {
          try {
            const token = JSON.parse(line.slice(6)).choices?.[0]?.delta?.content ||
"";
            fullReply += token;
            setStreaming(fullReply);
            // Scroll during streaming
            bottomRef.current?.scrollIntoView({ behavior: "smooth" });
          } catch {}
        }
      }

      setStreaming("");
      setMessages(prev => [...prev, { role: "assistant", content: fullReply }]);
      await supabase.from("messages").insert({ conversation_id: convId, role:
"assistant", content: fullReply });
    } catch (e) {
      setMessages(prev => [...prev, { role: "assistant", content: `⚠ Помилка:
${e.message}` }]);
    }
    setLoading(false);
  };

  const newChat = () => {
    setActiveConvId(null);
    setMessages([{ role: "assistant", content: "Розпочни новий діалог! Про що хочеш
дізнатися? 👋" }]);
  };

  return (
    <div style={{ height: "100vh", background: "#0f0f13", display: "flex",
flexDirection: "column", fontFamily: "'Segoe UI',system-ui,sans-serif", color:
"#e2e8f0", overflow: "hidden" }}>
      <style>{`
        @keyframes
bounce{0%,80%,100%{transform:translateY(0)}40%{transform:translateY(-7px)}}
        @keyframes
fadeIn{from{opacity:0;transform:translateY(8px)}to{opacity:1;transform:translateY(0)}
}
        @keyframes pulse{0%,100%{opacity:1}50%{opacity:.4}}
        @keyframes blink{0%,100%{opacity:1}50%{opacity:0}}
        .msg{animation:fadeIn .3s ease forwards}
        .conv-item:hover{background:rgba(139,92,246,0.1)!important;border-
color:rgba(139,92,246,0.3)!important}
        .conv-item.active{background:rgba(139,92,246,0.15)!important;border-
color:rgba(139,92,246,0.4)!important}
```

```
        .suggest-btn:hover{background:rgba(139,92,246,0.15)!important;border-
color:#8b5cf6!important}
        .del-btn{opacity:0;transition:opacity 0.2s}
        .conv-item:hover .del-btn{opacity:1}
        textarea:focus{border-color:#8b5cf6!important;box-shadow:0 0 0 3px
rgba(139,92,246,0.15)!important;outline:none}
        ::-webkit-scrollbar{width:4px}
        ::-webkit-scrollbar-thumb{background:#2d2d3a;border-radius:4px}
        .send-
btn:hover:not(:disabled){background:#7c3aed!important;transform:scale(1.05)}
      `}</style>

      {/* Top nav */}
      <div style={{ display: "flex", alignItems: "center", gap: 12, padding: "12px
20px", borderBottom: "1px solid rgba(255,255,255,0.06)", background:
"rgba(15,15,19,0.97)", flexShrink: 0 }}>
        <button onClick={() => setSidebarOpen(o => !o)} style={{ background:
"rgba(255,255,255,0.05)", border: "1px solid rgba(255,255,255,0.1)", color:
"#94a3b8", borderRadius: 8, padding: "5px 10px", cursor: "pointer", fontSize: 14
}}>☰</button>
        <Logo onClick={onHome} />
        <div style={{ flex: 1 }} />
        <div style={{ display: "flex", alignItems: "center", gap: 6, background:
"rgba(139,92,246,0.1)", border: "1px solid rgba(139,92,246,0.2)", borderRadius: 20,
padding: "4px 12px" }}>
          <div style={{ width: 7, height: 7, borderRadius: "50%", background:
"#8b5cf6", animation: "pulse 2s infinite" }} />
          <span style={{ color: "#a78bfa", fontSize: 12, fontWeight: 600 }}>AI
онлайн</span>
        </div>
      </div>

      <div style={{ flex: 1, display: "flex", overflow: "hidden" }}>
        {/* Sidebar */}
        {sidebarOpen && (
          <div style={{ width: 260, background: "#0a0a0e", borderRight: "1px solid
rgba(255,255,255,0.06)", display: "flex", flexDirection: "column", flexShrink: 0 }}>
            <div style={{ padding: "14px 12px", borderBottom: "1px solid
rgba(255,255,255,0.06)" }}>
              <button onClick={newChat} style={{ width: "100%", background:
"#8b5cf6", border: "none", borderRadius: 10, color: "#fff", fontSize: 13, fontWeight:
600, padding: "10px", cursor: "pointer", transition: "background 0.2s" }}>
                + Новий діалог
              </button>
            </div>

            <div style={{ flex: 1, overflowY: "auto", padding: "10px" }}>
              <p style={{ color: "#475569", fontSize: 11, textTransform: "uppercase",
letterSpacing: 1, margin: "0 4px 8px" }}>Історія</p>
```

```
            {conversations.length === 0 && <p style={{ color: "#334155", fontSize:
13, padding: "4px" }}>Діалогів ще немає</p>}
            {conversations.map(c => (
              <div key={c.id} className={`conv-item${activeConvId === c.id ? "
active" : ""}`}
                onClick={() => loadConversation(c)}
                style={{ display: "flex", alignItems: "center", justifyContent:
"space-between", padding: "9px 10px", borderRadius: 10, border: "1px solid
transparent", cursor: "pointer", marginBottom: 3, transition: "all 0.2s" }}>
                <span style={{ fontSize: 13, overflow: "hidden", textOverflow:
"ellipsis", whiteSpace: "nowrap", flex: 1, color: "#c4b5fd" }}>💬 {c.title}</span>
                <button className="del-btn" onClick={e => deleteConversation(c.id,
e)} style={{ background: "none", border: "none", color: "#ef4444", cursor: "pointer",
fontSize: 15, padding: "0 0 0 6px", flexShrink: 0 }}>×</button>
              </div>
            ))}
          </div>

          <div style={{ padding: "12px", borderTop: "1px solid
rgba(255,255,255,0.06)" }}>
            <button onClick={onQuiz} style={{ width: "100%", background:
"rgba(139,92,246,0.12)", border: "1px solid rgba(139,92,246,0.3)", borderRadius: 10,
color: "#a78bfa", fontSize: 13, fontWeight: 600, padding: "9px", cursor: "pointer",
marginBottom: 10, transition: "background 0.2s" }}>
              🎓 Тести
            </button>
            <div style={{ display: "flex", alignItems: "center", gap: 8 }}>
              <div style={{ width: 28, height: 28, borderRadius: "50%", background:
"linear-gradient(135deg,#8b5cf6,#6d28d9)", display: "flex", alignItems: "center",
justifyContent: "center", fontSize: 12, flexShrink: 0 }}>👤</div>
              <span style={{ fontSize: 12, color: "#64748b", flex: 1, overflow:
"hidden", textOverflow: "ellipsis" }}>{user.email}</span>
              <button onClick={onLogout} style={{ background: "none", border:
"none", color: "#475569", cursor: "pointer", fontSize: 12, flexShrink: 0
}}>Вийти</button>
            </div>
          </div>
        </div>
      )}

      {/* Chat area */}
      <div style={{ flex: 1, display: "flex", flexDirection: "column", overflow:
"hidden" }}>
        {/* Messages */}
        <div ref={chatBoxRef} style={{ flex: 1, overflowY: "auto", padding: "20px"
}}>
          <div style={{ maxWidth: 780, margin: "0 auto" }}>

            {messages.length <= 1 && (
              <div style={{ marginBottom: 20 }}>
```

```
                <p style={{ color: "#475569", fontSize: 11, textTransform:
"uppercase", letterSpacing: 1.2, margin: "0 0 10px" }}>Спробуй запитати</p>
                <div style={{ display: "flex", flexWrap: "wrap", gap: 8 }}>
                  {SUGGESTIONS.map(s => (
                    <button key={s} className="suggest-btn" onClick={() => send(s)}
style={{ background: "rgba(255,255,255,0.04)", border: "1px solid
rgba(255,255,255,0.1)", color: "#c4b5fd", borderRadius: 20, padding: "7px 16px",
fontSize: 13, cursor: "pointer", transition: "all 0.2s" }}>{s}</button>
                  ))}
                </div>
              </div>
            )}

            {messages.map((m, i) => (
              <div key={i} className="msg" style={{ display: "flex",
justifyContent: m.role === "user" ? "flex-end" : "flex-start", marginBottom: 14 }}>
                {m.role === "assistant" && (
                  <div style={{ width: 30, height: 30, borderRadius: 9, flexShrink:
0, background: "linear-gradient(135deg,#8b5cf6,#6d28d9)", display: "flex",
alignItems: "center", justifyContent: "center", fontSize: 13, marginRight: 10,
alignSelf: "flex-end" }}>✦</div>
                )}
                <div style={{
                  maxWidth: "78%",
                  background: m.role === "user" ? "linear-
gradient(135deg,#8b5cf6,#7c3aed)" : "rgba(255,255,255,0.05)",
                  border: m.role === "user" ? "none" : "1px solid
rgba(255,255,255,0.08)",
                  borderRadius: m.role === "user" ? "18px 18px 4px 18px" : "18px
18px 18px 4px",
                  padding: "12px 16px", color: "#e2e8f0", fontSize: 14, lineHeight:
1.65,
                }}>
                  <MessageContent text={m.content} />
                </div>
              </div>
            ))}

            {/* Streaming response */}
            {streaming && (
              <div className="msg" style={{ display: "flex", marginBottom: 14 }}>
                <div style={{ width: 30, height: 30, borderRadius: 9, flexShrink:
0, background: "linear-gradient(135deg,#8b5cf6,#6d28d9)", display: "flex",
alignItems: "center", justifyContent: "center", fontSize: 13, marginRight: 10,
alignSelf: "flex-end" }}>✦</div>
                <div style={{ maxWidth: "78%", background:
"rgba(255,255,255,0.05)", border: "1px solid rgba(255,255,255,0.08)", borderRadius:
"18px 18px 18px 4px", padding: "12px 16px", color: "#e2e8f0", fontSize: 14,
lineHeight: 1.65 }}>
                  <MessageContent text={streaming} />
```

```
                    <span style={{ display: "inline-block", width: 2, height: 15,
background: "#8b5cf6", marginLeft: 2, verticalAlign: "middle", animation: "blink 0.8s
infinite" }} />
                  </div>
                </div>
              )}

              {/* Loading dots (before streaming starts) */}
              {loading && !streaming && (
                <div style={{ display: "flex", marginBottom: 14 }}>
                  <div style={{ width: 30, height: 30, borderRadius: 9, background:
"linear-gradient(135deg,#8b5cf6,#6d28d9)", display: "flex", alignItems: "center",
justifyContent: "center", fontSize: 13, marginRight: 10 }}>✦</div>
                  <div style={{ background: "rgba(255,255,255,0.05)", border: "1px
solid rgba(255,255,255,0.08)", borderRadius: "18px 18px 18px 4px" }}><TypingDots
/></div>
                </div>
              )}

              <div ref={bottomRef} />
            </div>
          </div>

          {/* Input */}
          <div style={{ padding: "14px 20px", borderTop: "1px solid
rgba(255,255,255,0.06)", background: "rgba(15,15,19,0.97)", flexShrink: 0 }}>
            <div style={{ maxWidth: 780, margin: "0 auto", display: "flex", gap: 10,
alignItems: "flex-end" }}>
              <textarea value={input} onChange={e => setInput(e.target.value)}
                onKeyDown={e => { if (e.key === "Enter" && !e.shiftKey) {
e.preventDefault(); send(); } }}
                placeholder="Задай питання... (Enter — надіслати, Shift+Enter — новий
рядок)"
                rows={1}
                style={{ flex: 1, background: "rgba(255,255,255,0.05)", border: "1px
solid rgba(255,255,255,0.1)", borderRadius: 14, color: "#e2e8f0", fontSize: 14,
padding: "12px 16px", resize: "none", lineHeight: 1.5, fontFamily: "inherit",
minHeight: 46, maxHeight: 140, overflowY: "auto", transition: "border-color .2s, box-
shadow .2s" }}
                onInput={e => { e.target.style.height = "auto"; e.target.style.height
= Math.min(e.target.scrollHeight, 140) + "px"; }}
              />
              <button className="send-btn" onClick={() => send()}
disabled={!input.trim() || loading}
                style={{ background: input.trim() && !loading ? "#8b5cf6" :
"rgba(255,255,255,0.07)", border: "none", borderRadius: 12, width: 46, height: 46,
color: input.trim() && !loading ? "#fff" : "#475569", fontSize: 20, cursor:
input.trim() && !loading ? "pointer" : "default", transition: "all .2s", flexShrink:
0, display: "flex", alignItems: "center", justifyContent: "center" }}>
                {loading ? "⏳" : "↑"}
```

```
              </button>
            </div>
            <p style={{ textAlign: "center", color: "#334155", fontSize: 11, margin:
"8px 0 0" }}>Дипломна робота · Комп'ютерні науки 4 курс · Powered by Groq AI</p>
          </div>
        </div>
      </div>
    </div>
  );
}
```

Лістинг А.5 – Головна сторінка та маршрутизація

```
// ── LANDING ──────────────────────────────────────────────────────
function LandingPage({ onStart }) {
  return (
    <div style={{ minHeight: "100vh", background: "#0f0f13", color: "#fff",
fontFamily: "'Segoe UI',system-ui,sans-serif" }}>
      <style>{`
        @keyframes
fadeUp{from{opacity:0;transform:translateY(20px)}to{opacity:1;transform:translateY(0)
}}
        .f1{animation:fadeUp .6s ease forwards}
        .f2{animation:fadeUp .6s .15s ease both}
        .f3{animation:fadeUp .6s .3s ease both}
        .f4{animation:fadeUp .6s .45s ease both}
        .feat:hover{border-
color:rgba(139,92,246,0.5)!important;transform:translateY(-
3px);background:rgba(139,92,246,0.07)!important}
        .btn-p:hover{background:#7c3aed!important;transform:translateY(-1px)}
        .btn-
s:hover{background:rgba(255,255,255,0.1)!important;transform:translateY(-1px)}
        .nav-start:hover{background:#7c3aed!important}
      `}</style>

      <nav style={{ display: "flex", justifyContent: "space-between", alignItems:
"center", padding: "16px 40px", borderBottom: "1px solid rgba(255,255,255,0.06)",
position: "sticky", top: 0, background: "rgba(15,15,19,0.95)", backdropFilter:
"blur(12px)", zIndex: 50 }}>
        <Logo onClick={() => {}} />
        <div style={{ display: "flex", gap: 10 }}>
          <button onClick={onStart} style={{ background: "transparent", border:
"none", color: "#c4b5fd", fontSize: 14, cursor: "pointer", padding: "8px 16px"
}}>Увійти</button>
          <button className="nav-start" onClick={onStart} style={{ background:
"#8b5cf6", border: "none", color: "#fff", fontSize: 14, fontWeight: 600, cursor:
"pointer", padding: "9px 20px", borderRadius: 10, transition: "all .2s"
}}>Почати</button>
        </div>
      </nav>
```

```
      <div style={{ textAlign: "center", padding: "90px 24px 70px" }}>
        <div className="f1" style={{ display: "inline-flex", alignItems: "center",
gap: 8, background: "rgba(139,92,246,0.12)", border: "1px solid
rgba(139,92,246,0.3)", borderRadius: 30, padding: "6px 16px", marginBottom: 28,
fontSize: 13, color: "#c4b5fd" }}>
          ✦ Інформаційна система інтелектуальної підтримки навчання
        </div>
        <h1 className="f2" style={{ fontSize: "clamp(34px,6vw,66px)", fontWeight:
800, lineHeight: 1.15, margin: "0 0 22px" }}>
          AI-помічник для студентів<br /><span style={{ color: "#8b5cf6"
}}>комп'ютерних наук</span>
        </h1>
        <p className="f3" style={{ fontSize: 17, color: "#94a3b8", maxWidth: 540,
margin: "0 auto 36px", lineHeight: 1.7 }}>
          Пояснення тем, приклади коду та генерація тестів — усе в одному діалозі.
Навчайся у власному темпі з персональним AI-наставником.
        </p>
        <div className="f4" style={{ display: "flex", gap: 12, justifyContent:
"center", flexWrap: "wrap" }}>
          {/* ✅ ВИПРАВЛЕНО: кнопки ведуть одразу на реєстрацію */}
          <button className="btn-p" onClick={onStart} style={{ background: "#8b5cf6",
border: "none", color: "#fff", fontSize: 15, fontWeight: 600, padding: "13px 26px",
borderRadius: 12, cursor: "pointer", transition: "all .2s" }}>✦ Почати
безкоштовно</button>
          <button className="btn-s" onClick={onStart} style={{ background:
"rgba(255,255,255,0.06)", border: "1px solid rgba(255,255,255,0.12)", color: "#fff",
fontSize: 15, fontWeight: 600, padding: "13px 26px", borderRadius: 12, cursor:
"pointer", transition: "all .2s" }}>🎓 Спробувати тести</button>
        </div>
      </div>

      <div style={{ padding: "50px 40px 70px", maxWidth: 1100, margin: "0 auto" }}>
        <div style={{ textAlign: "center", marginBottom: 44 }}>
          <h2 style={{ fontSize: 34, fontWeight: 800, margin: "0 0 10px"
}}>Можливості системи</h2>
          <p style={{ color: "#94a3b8", fontSize: 15 }}>Усе, що потрібно для
ефективного самостійного навчання</p>
        </div>
        <div style={{ display: "grid", gridTemplateColumns: "repeat(auto-
fit,minmax(290px,1fr))", gap: 18 }}>
          {FEATURES.map((f, i) => (
            <div key={i} className="feat" style={{ background:
"rgba(255,255,255,0.03)", border: "1px solid rgba(255,255,255,0.08)", borderRadius:
16, padding: "26px 22px", transition: "all .25s", cursor: "default" }}>
              <div style={{ width: 42, height: 42, borderRadius: 11, background:
"rgba(139,92,246,0.15)", border: "1px solid rgba(139,92,246,0.2)", display: "flex",
alignItems: "center", justifyContent: "center", fontSize: 19, marginBottom: 14
}}>{f.icon}</div>
              <h3 style={{ margin: "0 0 8px", fontSize: 16, fontWeight: 700
}}>{f.title}</h3>
```

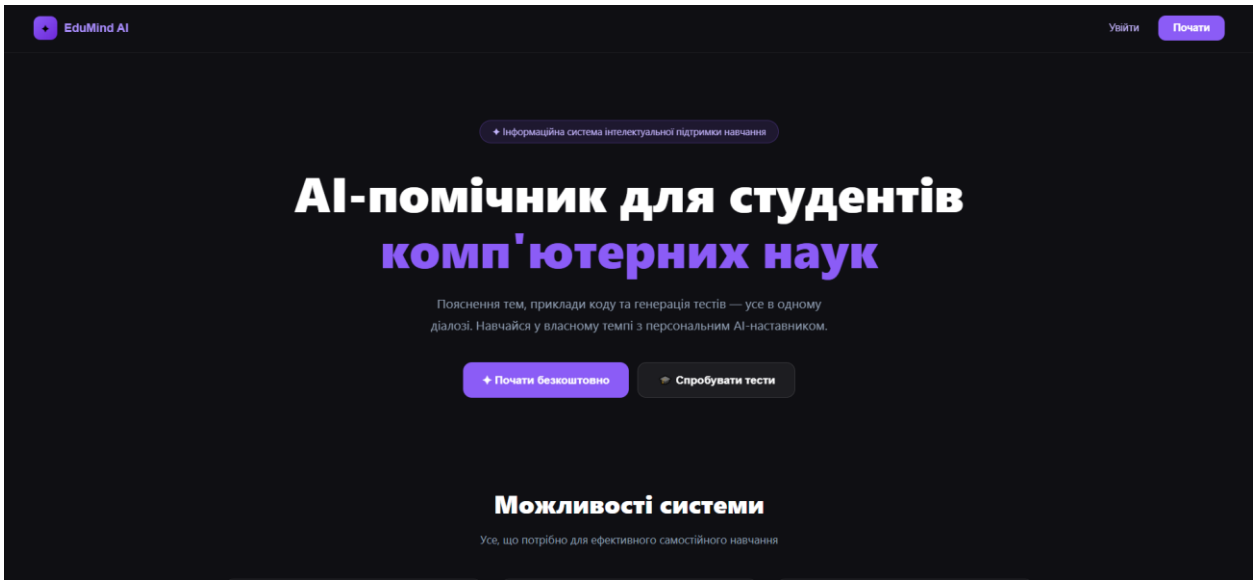
```
              <p style={{ margin: 0, color: "#94a3b8", fontSize: 13, lineHeight: 1.6
}}>{f.desc}</p>
            </div>
          ))}
        </div>
      </div>

      <div style={{ padding: "0 40px 70px" }}>
        <div style={{ background: "linear-gradient(135deg,#7c3aed,#8b5cf6)",
borderRadius: 20, padding: "56px 40px", textAlign: "center", maxWidth: 1100, margin:
"0 auto" }}>
          <h2 style={{ fontSize: 34, fontWeight: 800, margin: "0 0 12px" }}>Готові
навчатися розумніше?</h2>
          <p style={{ color: "rgba(255,255,255,0.85)", fontSize: 15, margin: "0 0
28px" }}>Зареєструйтесь і отримайте власного AI-наставника вже зараз.</p>
          <button onClick={onStart} style={{ background: "#0f0f13", border: "none",
color: "#fff", fontSize: 15, fontWeight: 700, padding: "13px 30px", borderRadius: 12,
cursor: "pointer" }}>Створити акаунт →</button>
        </div>
      </div>
    </div>
  );
}

// ── ROOT ──────────────────────────────────────────────────────────
export default function App() {
  const [page, setPage] = useState("landing");
  const [user, setUser] = useState(null);

  useEffect(() => {
    supabase.auth.getSession().then(({ data }) => {
      if (data.session?.user) { setUser(data.session.user); setPage("chat"); }
    });
    supabase.auth.onAuthStateChange((_, session) => {
      setUser(session?.user || null);
      if (!session?.user) setPage("landing");
    });
  }, []);

  const logout = async () => {
    await supabase.auth.signOut();
    setUser(null);
    setPage("landing");
  };

  const goHome = () => setPage(user ? "chat" : "landing");

  if (page === "landing") return <LandingPage onStart={() => setPage("auth")} />;
  if (page === "auth") return <AuthPage onAuth={u => { setUser(u); setPage("chat");
}} onBack={() => setPage("landing")} />;
```

```
  if (page === "chat" && user) return <ChatPage user={user} onLogout={logout}
onQuiz={() => setPage("quiz")} onHome={() => setPage("landing")} />;
  if (page === "quiz" && user) return <QuizPage user={user} onBack={() =>
setPage("chat")} onHome={() => setPage("landing")} />;
  return <LandingPage onStart={() => setPage("auth")} />;
}
```

ДОДАТОК Б

Графічний інтерфейс розробленої інформаційної системи

Рисунок Б.1 – Головна сторінка (Landing Page) вебзастосунку EduMind AI



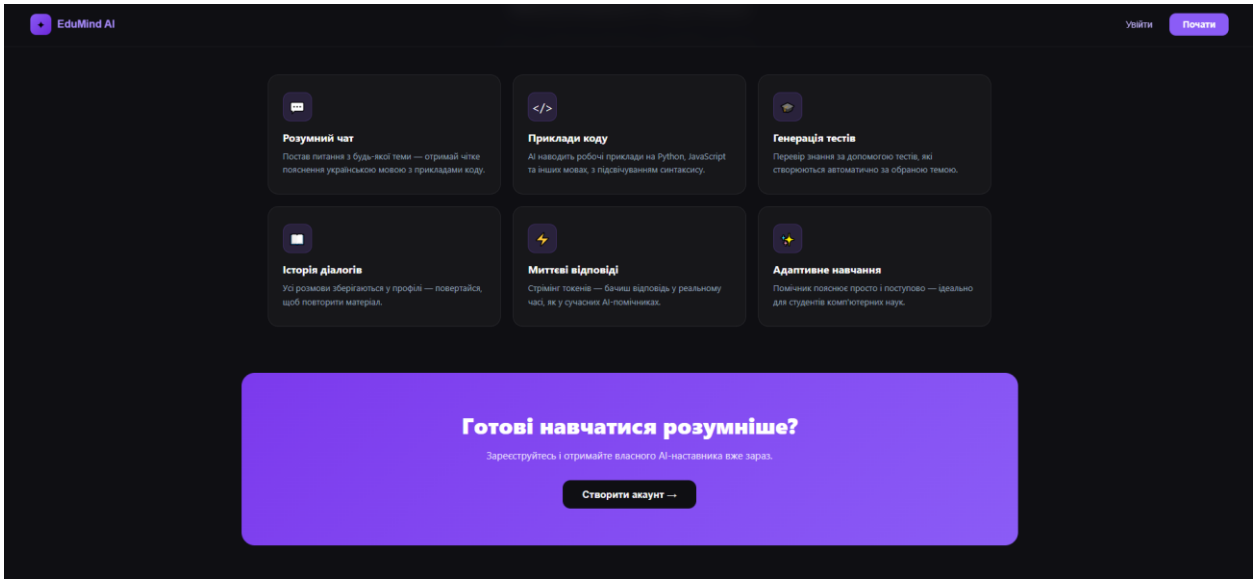


Рисунок Б.2 – Інтерфейс модуля авторизації користувачів

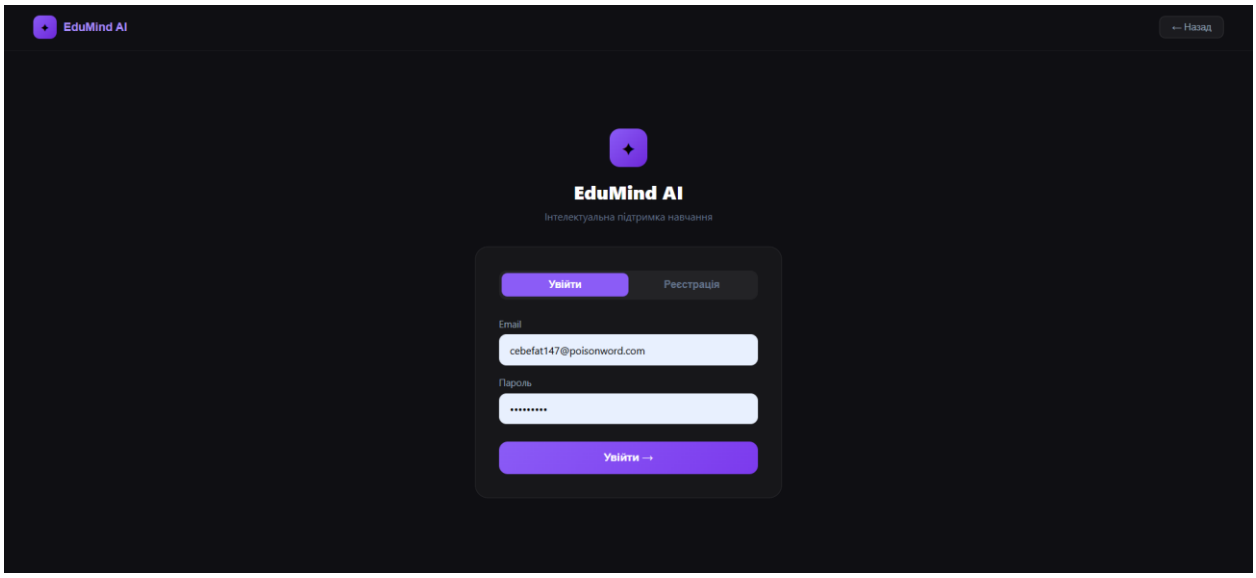
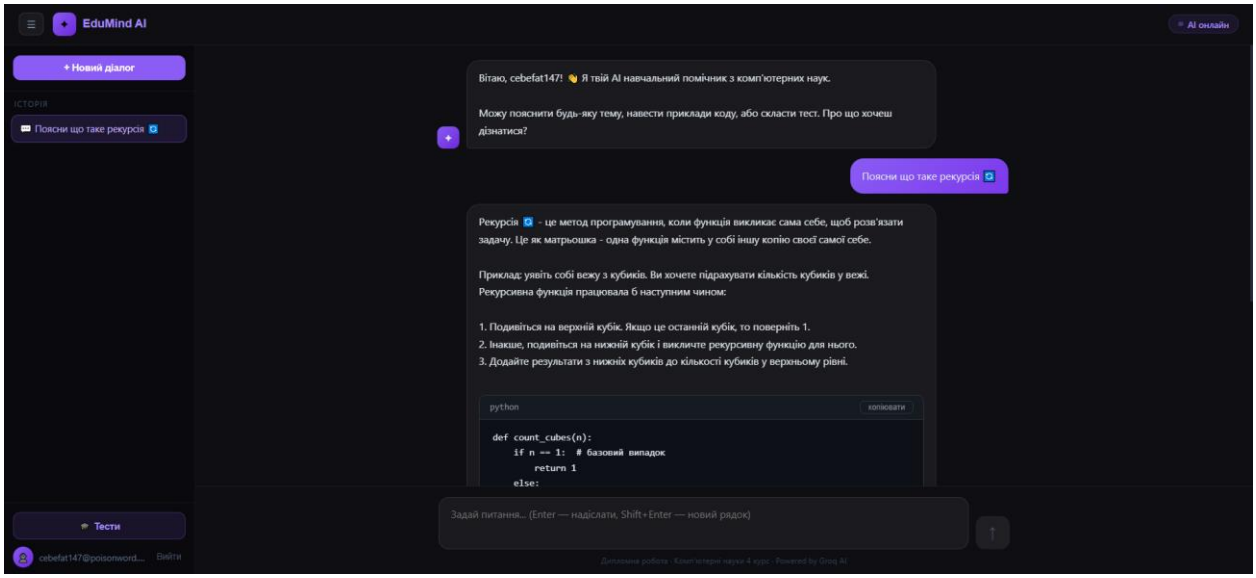


Рисунок Б.3 – Інтерфейс інтелектуального чату з підтримкою підсвічування синтаксису коду



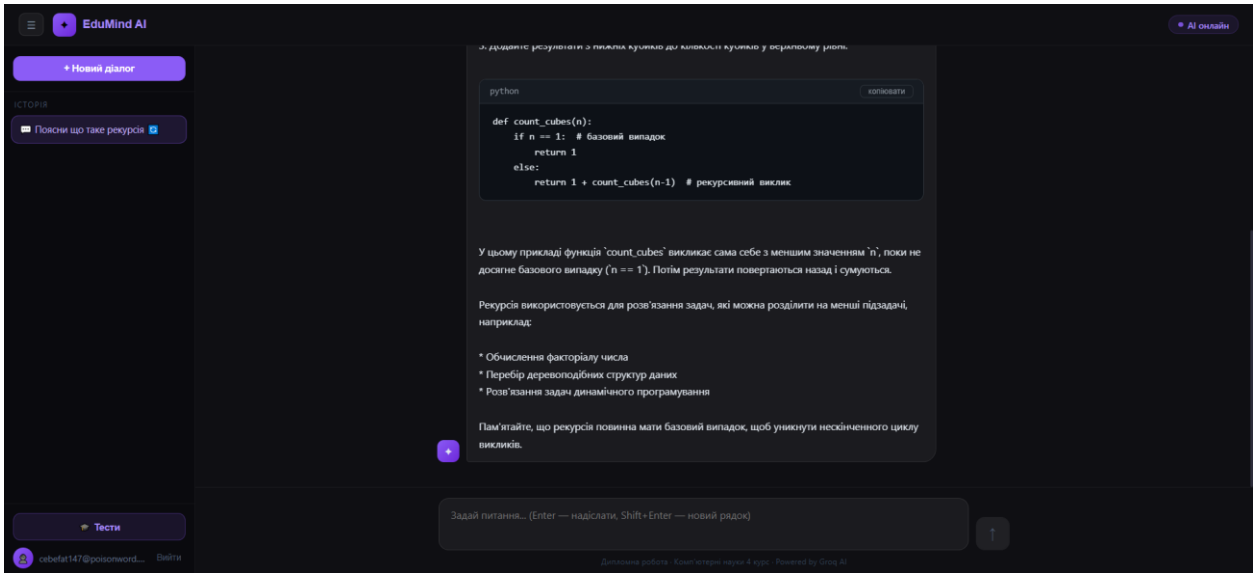


Рисунок Б.4 – Інтерфейс модуля проходження автоматично згенерованих тестів

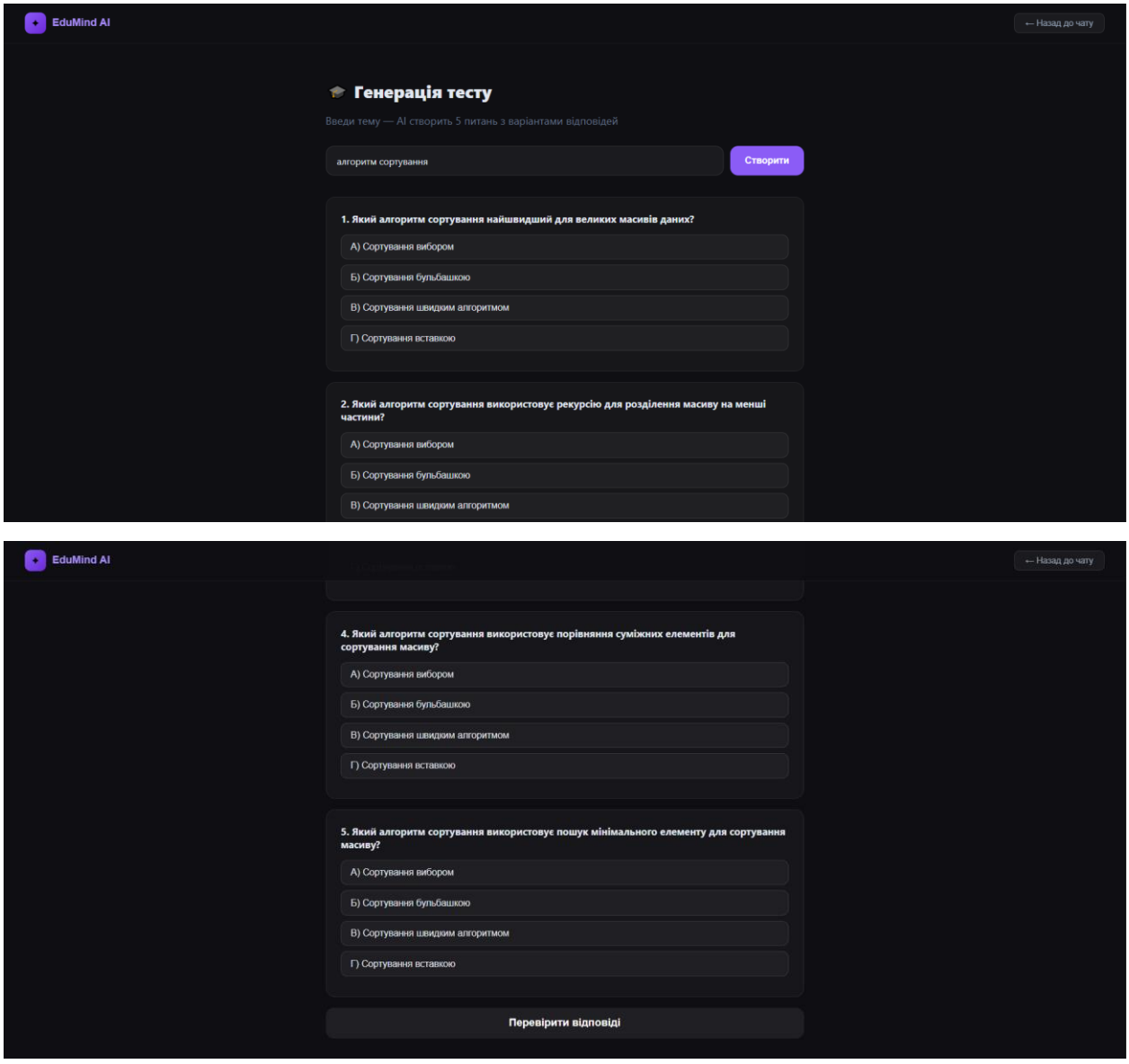


Рисунок Б.5 – Відображення результатів контролю знань користувача

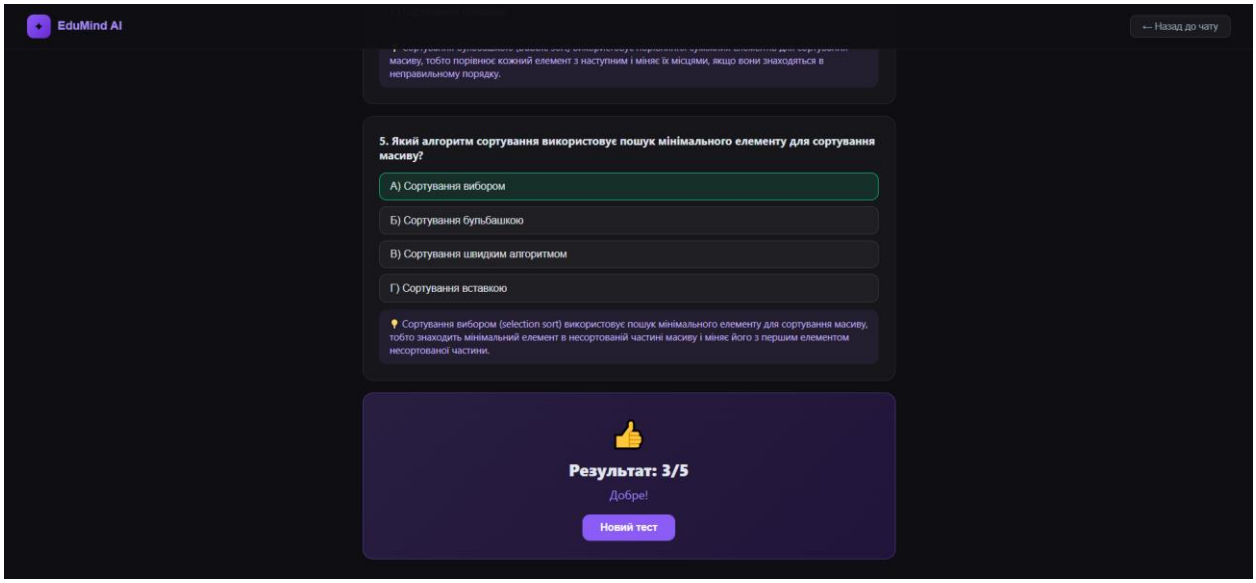
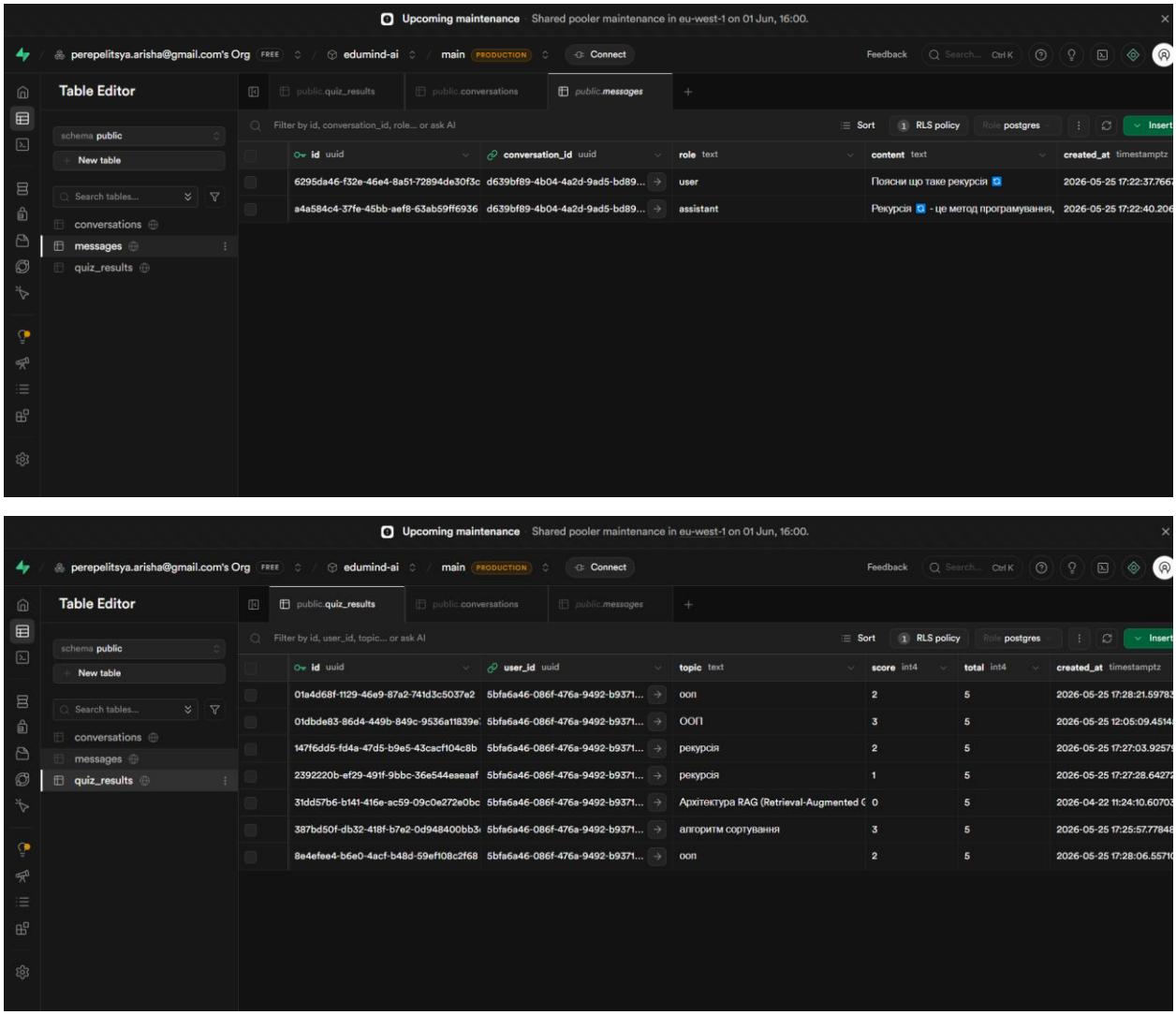
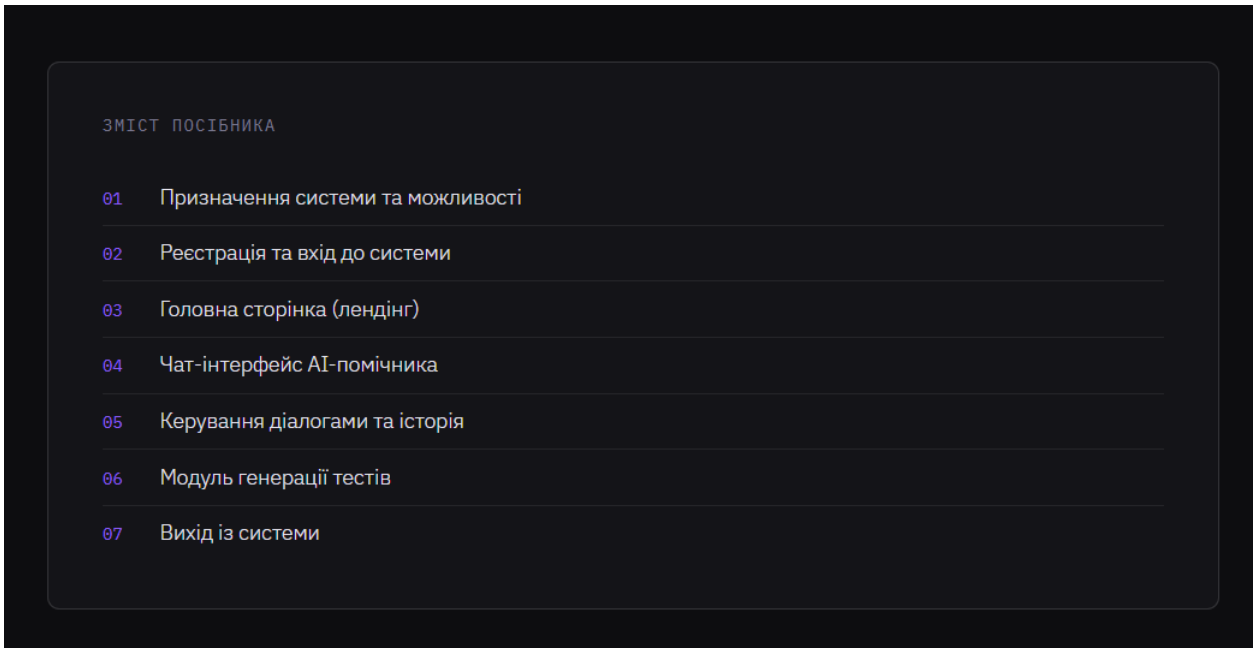


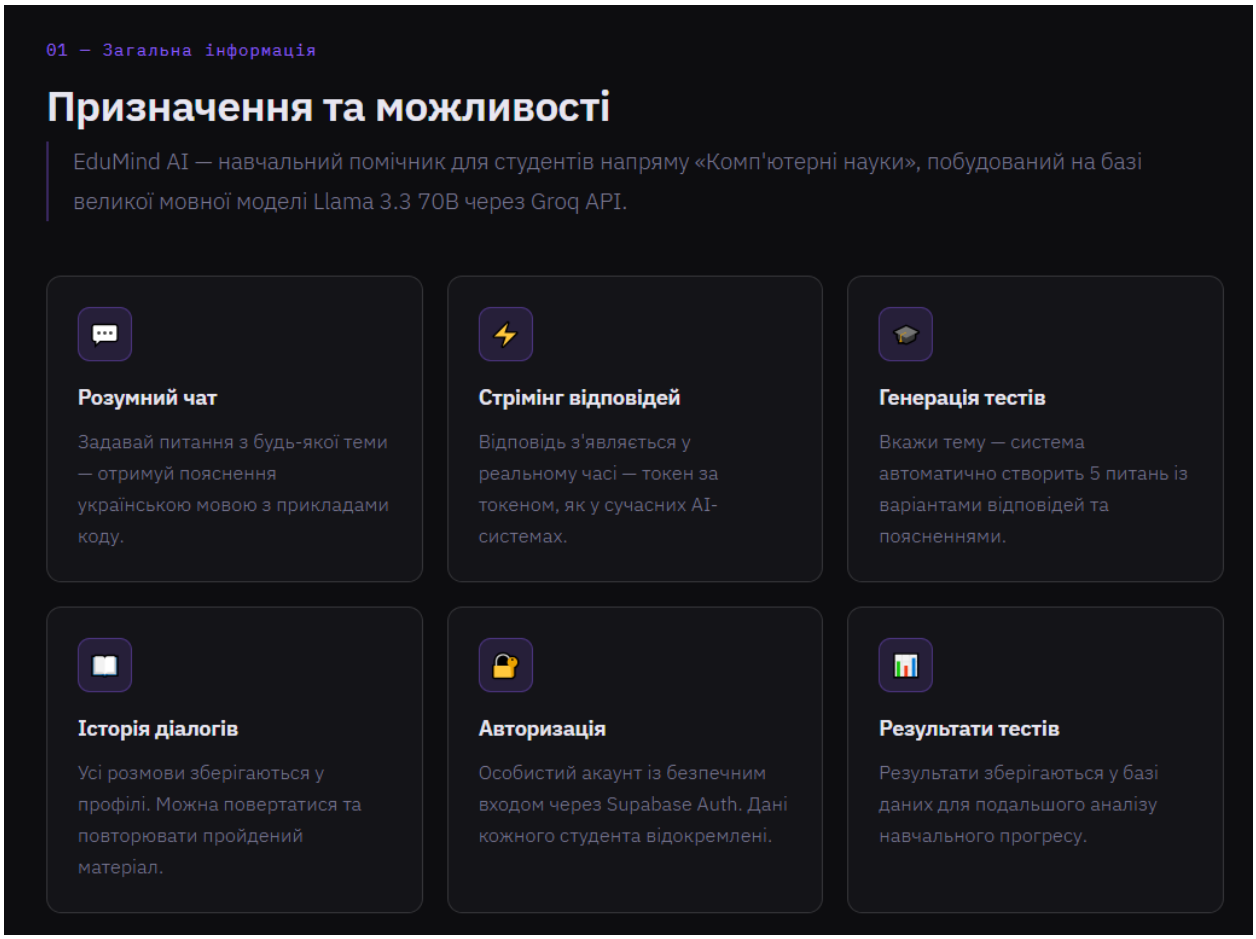
Рисунок Б.6 – Структура збережених даних у хмарній базі Supabase



ДОДАТОК В

Посібник користувача вебзастосунку EduMind AI.




01 — Загальна інформація
Призначення та можливості
EduMind AI — навчальний помічник для студентів напряму «Комп'ютерні науки», побудований на базі великої мовної моделі Llama 3.3 70B через Groq API.
Розумний чат
Задавай питання з будь-якої теми — отримуй пояснення українською мовою з прикладами коду.
Стрімінг відповідей
Відповідь з'являється у реальному часі — токен за токеном, як у сучасних AI-системах.
Генерація тестів
Вкажи тему — система автоматично створить 5 питань із варіантами відповідей та поясненнями.
Історія діалогів
Усі розмови зберігаються у профілі. Можна повертатися та повторювати пройдений матеріал.
Авторизація
Особистий акаунт із безпечним входом через Supabase Auth. Дані кожного студента відокремлені.
Результати тестів
Результати зберігаються у базі даних для подальшого аналізу навчального прогресу.

02 — Вхід у систему

Реєстрація та авторизація

Для роботи з системою необхідно мати особистий акаунт. Реєстрація займає менше хвилини.

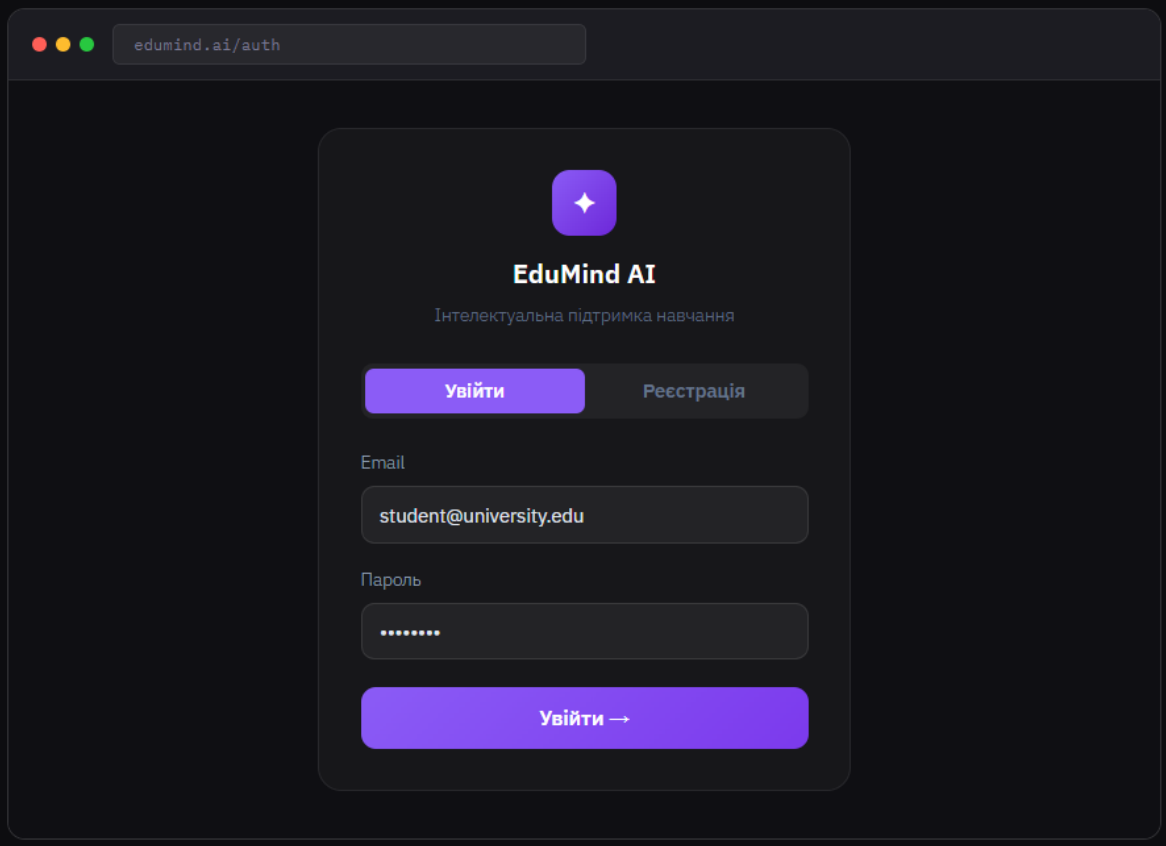


Рис. 1 — Сторінка авторизації EduMind AI

01 Перейдіть на сторінку авторизації

Натисніть кнопку «Почати» або «Увійти» на головній сторінці. Відкриється форма входу/реєстрації.

02 Оберіть вкладку «Реєстрація» (для нових користувачів)

Введіть email і пароль (мінімум 6 символів). Натисніть «Створити акаунт». Перевірте пошту — прийде лист підтвердження від Supabase.

03 Підтвердіть email і увійдіть

Перейдіть за посиланням у листі. Поверніться на вкладку «Увійти», введіть email і пароль — система автоматично направить вас до чату.

Після успішного входу система запам'ятовує сесію. При наступному відкритті сторінки авторизація відбудеться автоматично — вводити дані знову не потрібно.

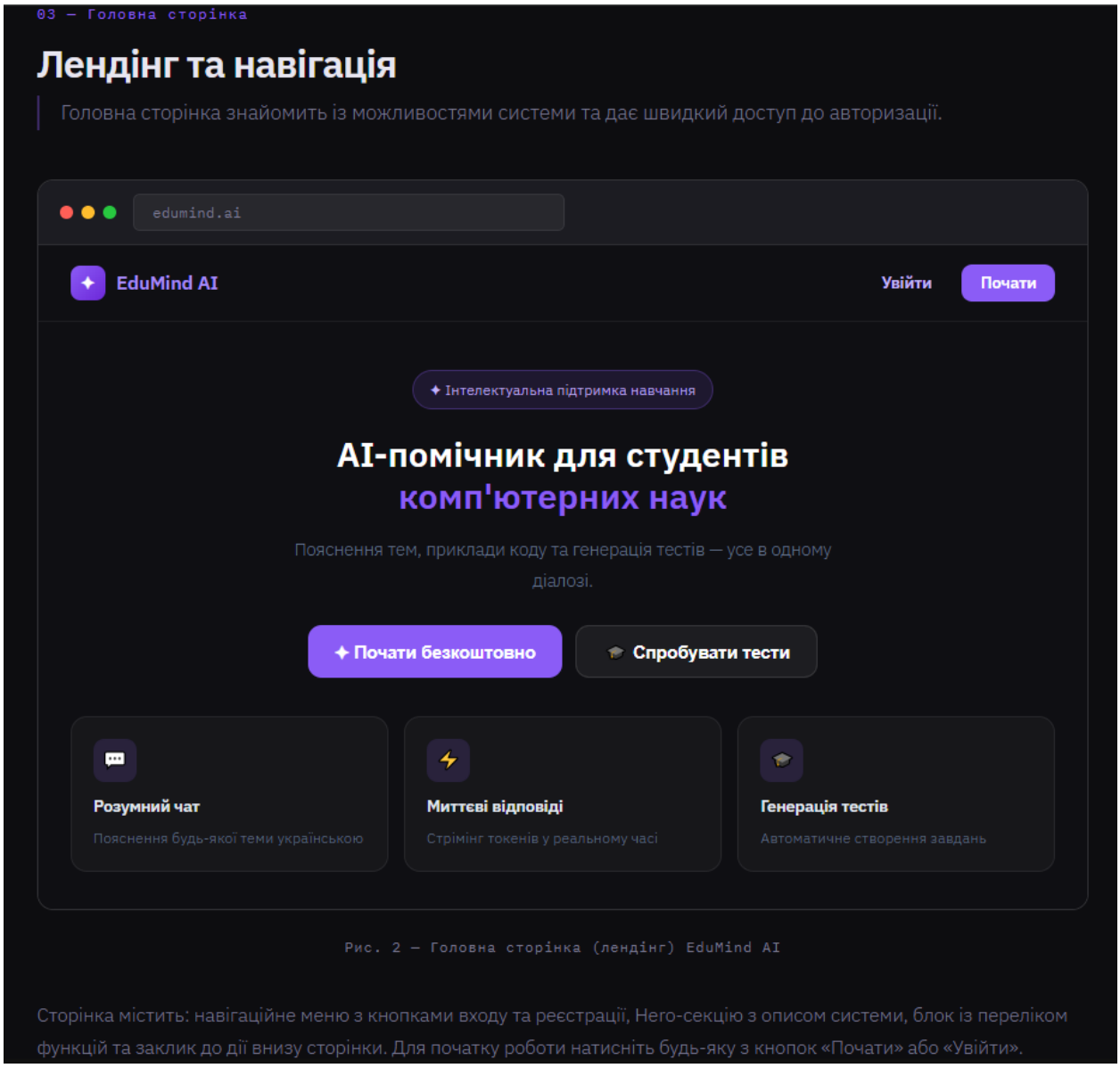


Рис. 2 — Головна сторінка (лендінг) EduMind AI

Сторінка містить: навігаційне меню з кнопками входу та реєстрації, Hero-секцію з описом системи, блок із переліком функцій та заклик до дії внизу сторінки. Для початку роботи натисніть будь-яку з кнопок «Почати» або «Увійти».

04 — Головний інструмент

Чат-інтерфейс AI-помічника

> Основна сторінка системи — чат із AI. Тут можна ставити запитання, отримувати пояснення та переглядати приклади коду.

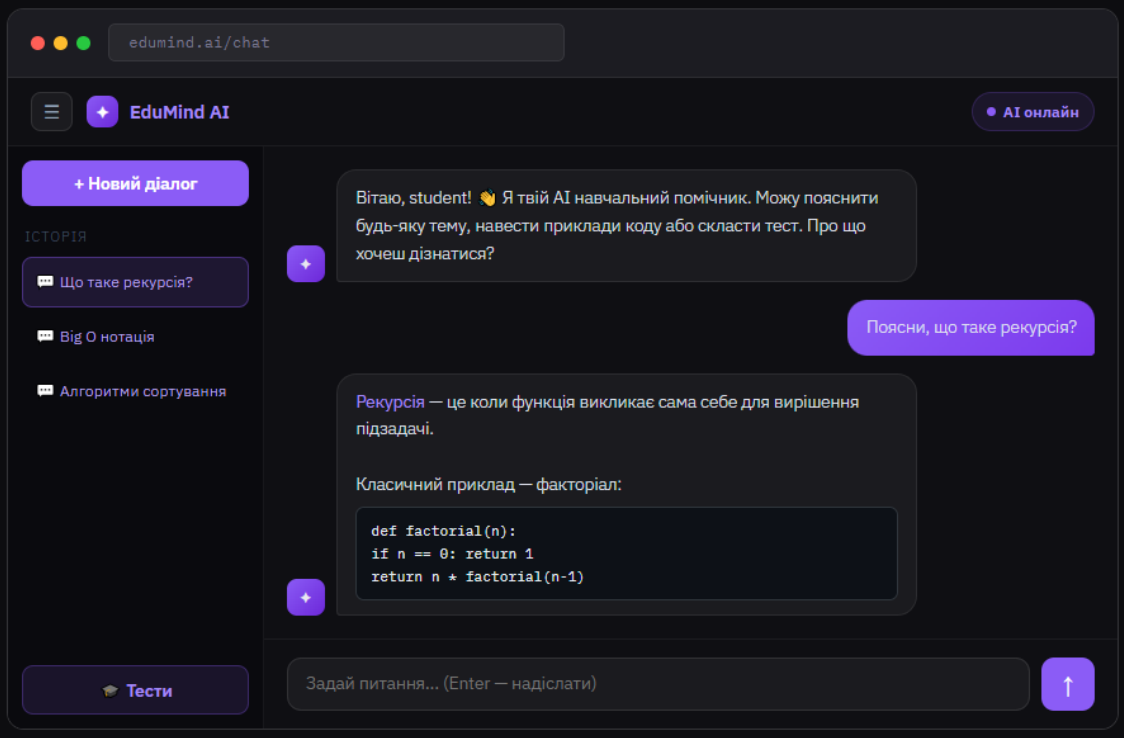
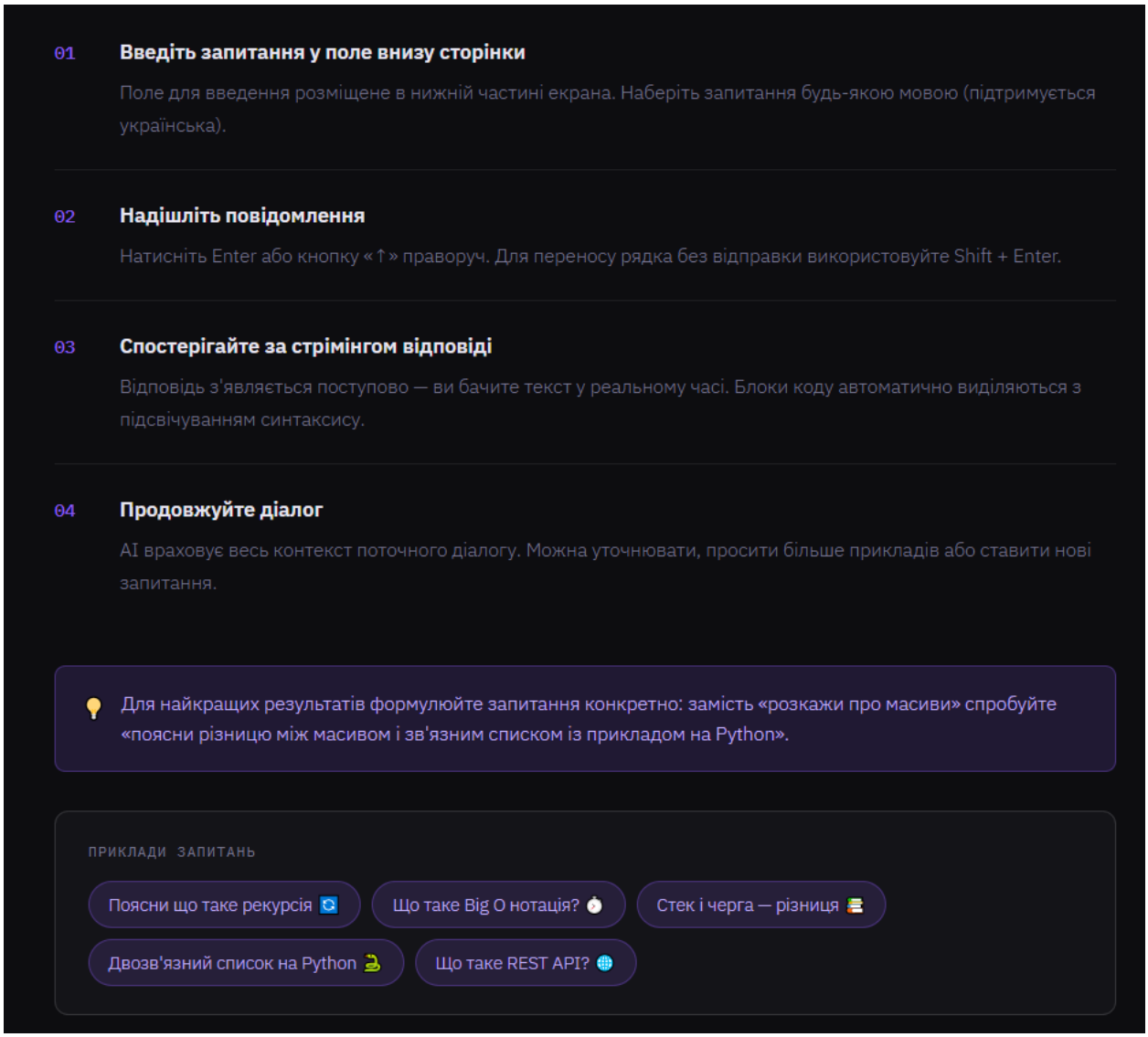


Рис. 3 — Інтерфейс чату з AI-помічником


01
Введіть запитання у поле внизу сторінки
Поле для введення розміщене в нижній частині екрана. Наберіть запитання будь-якою мовою (підтримується українська).
02
Надішліть повідомлення
Натисніть Enter або кнопку «↑» праворуч. Для переносу рядка без відправки використовуйте Shift + Enter.
03
Спостерігайте за стрімінгом відповіді
Відповідь з'являється поступово — ви бачите текст у реальному часі. Блоки коду автоматично виділяються з підсвічуванням синтаксису.
04
Продовжуйте діалог
AI враховує весь контекст поточного діалогу. Можна уточнювати, просити більше прикладів або ставити нові запитання.
Для найкращих результатів формулюйте запитання конкретно: замість «розкажи про масиви» спробуйте «поясни різницю між масивом і зв'язним списком із прикладом на Python».
ПРИКЛАДИ ЗАПИТАНЬ
Поясни що таке рекурсія
Що таке Big O нотація?
Стек і черга — різниця
Двозв'язний список на Python
Що таке REST API?

05 — Збереження даних

Керування діалогами та історія

> Усі розмови автоматично зберігаються у базі даних і відображаються у бічній панелі.

01 **Бічна панель із діалогами**

Ліворуч відображається список усіх попередніх розмов. Назва діалогу — перші 50 символів вашого першого запитання.

02 **Відкрити минулий діалог**

Натисніть на будь-який діалог у списку — всі повідомлення завантажаться з бази даних і відобразяться у полі чату.

03 **Новий діалог**

Натисніть «+ Новий діалог» у верхній частині панелі. Попередня розмова збережеться, а поле чату очиститься.

04 **Видалення діалогу**

При наведенні на назву діалогу з'являється кнопка «×». Натисніть її для видалення розмови. Дія незворотна.

05 **Приховати/показати панель**

Кнопка «☰» у верхньому лівому куті сховає бічну панель, збільшивши простір для чату.

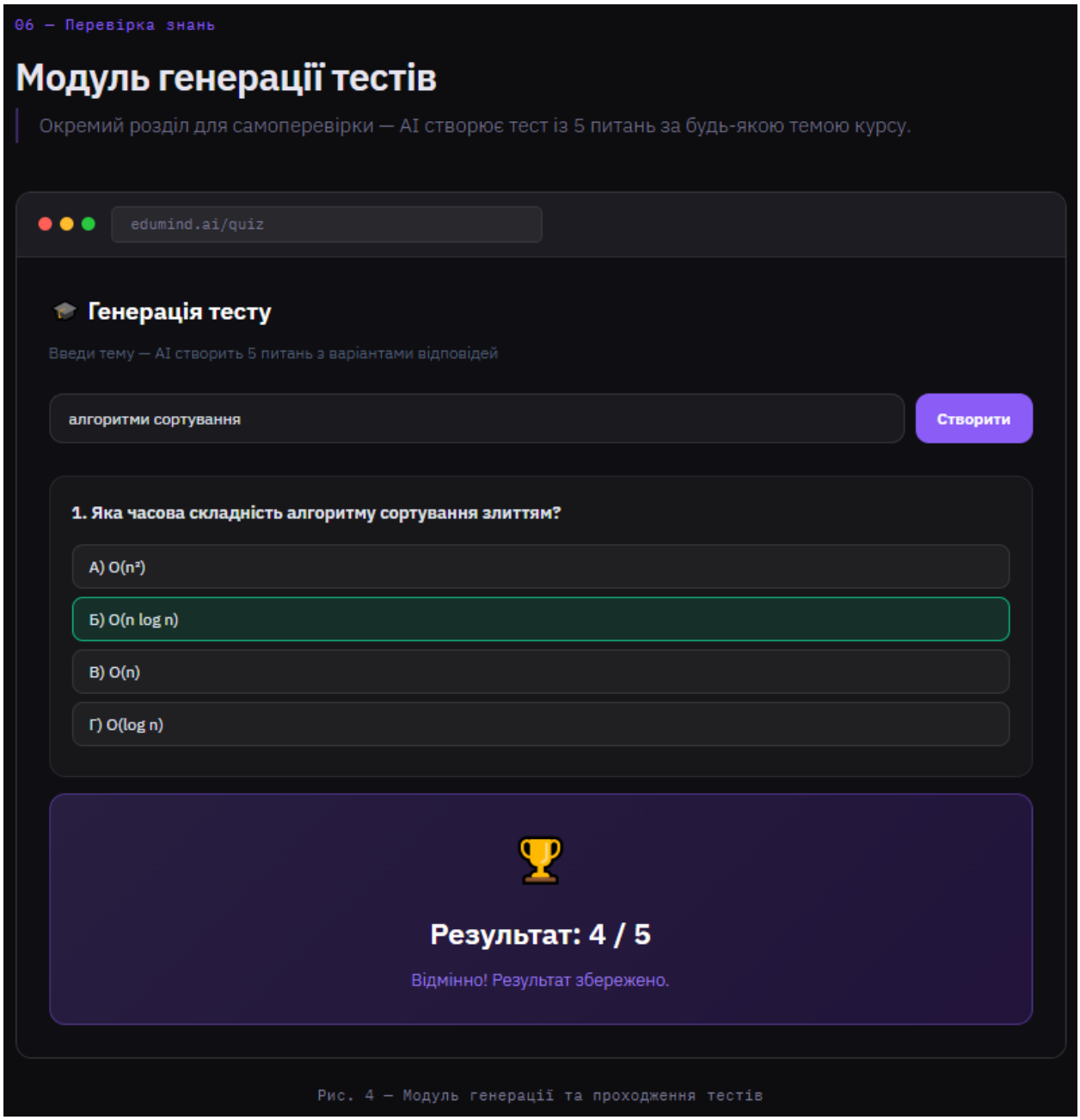


Рис. 4 — Модуль генерації та проходження тестів

01 **Перейдіть до розділу тестів**

Натисніть кнопку « 🎓 Тести» у нижній частині бічної панелі чату або виберіть «Спробувати тести» на головній сторінці.

02 **Введіть тему тесту**

У поле введення напишіть тему, наприклад: «алгоритми сортування», «рекурсія», «ООП», «REST API». Натисніть Enter або «Створити».

03 **Дочекайтесь генерації**

AI формує 5 питань із 4 варіантами відповідей кожне. Генерація займає 3–5 секунд.

04 **Дайте відповіді на всі питання**

Натисніть на один із варіантів для кожного питання. Обраний варіант підсвічується фіолетовим кольором.

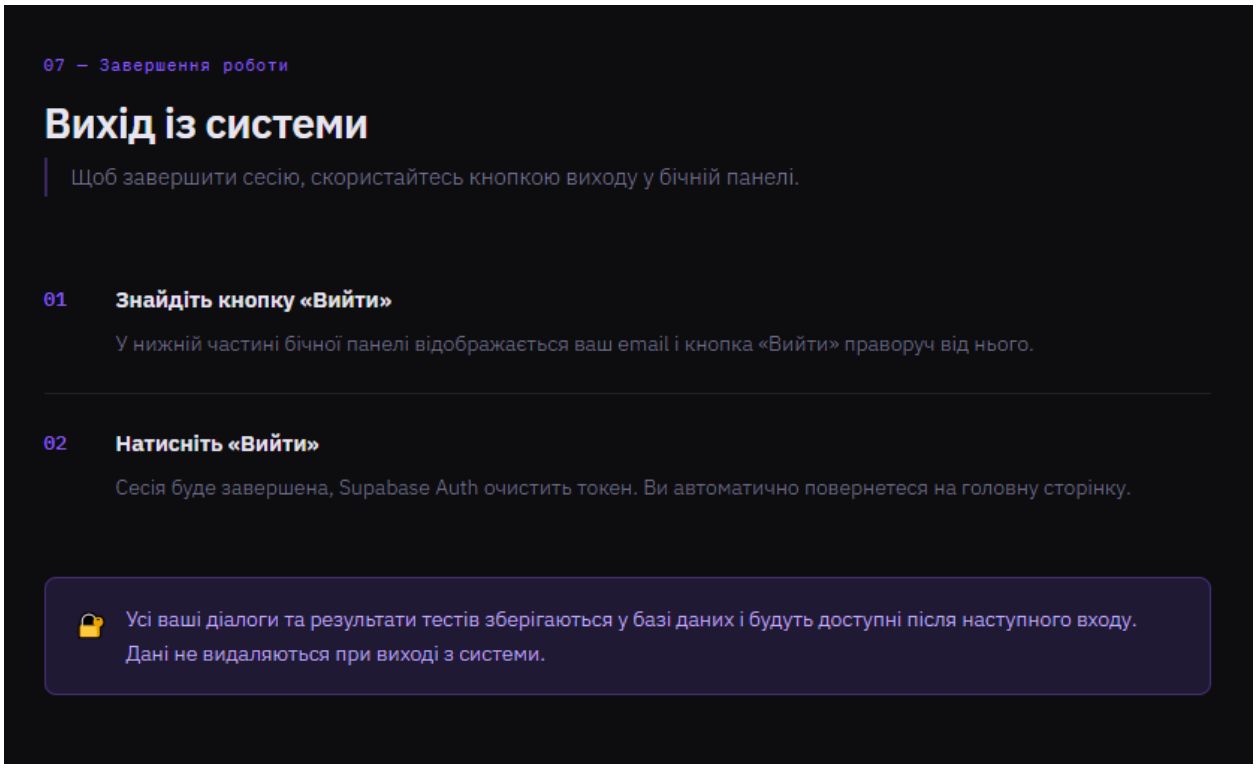
05 **Перевірте результат**

Після вибору всіх відповідей натисніть «Перевірити відповіді». Правильні відповіді позначаться зеленим, неправильні — червоним. Під кожним питанням з'явиться пояснення.

06 **Результат зберігається автоматично**

Бал (score/total) та тема тесту зберігаються у базі даних у вашому профілі. Натисніть «Новий тест» для повторного проходження.

Оцінка результату: 5/5 — «Відмінно! 🏆», 4/5 — «Добре! 👍», 3/5 і нижче — «Треба повторити матеріал 📚». Після перегляду пояснень поверніться до чату та попросіть AI детальніше розповісти про теми, де виникли помилки.


07 — Завершення роботи
Вихід із системи
Щоб завершити сесію, скористайтесь кнопкою виходу у бічній панелі.
01
Знайдіть кнопку «Вийти»
У нижній частині бічної панелі відображається ваш email і кнопка «Вийти» праворуч від нього.
02
Натисніть «Вийти»
Сесія буде завершена, Supabase Auth очистить токен. Ви автоматично повернетеся на головну сторінку.
Усі ваші діалоги та результати тестів зберігаються у базі даних і будуть доступні після наступного входу. Дані не видаляються при виході з системи.