Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра __Комп'ютерного моделювання та інтелектуальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
_____Розробка веб-застосунку чат-боту на базі Open AI API_____
(тема)

Виконав:
здобувач ____2_____ року навчання,
групи _____СПР-24-1_____
_____Домніч Д.В._____
(власне ім'я, прізвище)

Спеціальність ___122 Комп'ютерні науки___
(код і повна назва спеціальності)
Тип програми ___освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)
Освітня програма __Системне проєктування_
(повна назва освітньої програми)

Керівник _доцент каф. КМІТ Урняєва І.А.____
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри КМІТ _____ проф. Гребеннік І.В.
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Комп'ютерного моделювання та інтелектуальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне проєктування
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Домнічу Дмитру Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-застосунку чат-боту на базі Open AI API
затверджена наказом університету від 06.04.2026 р. № 268 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 15.05.2026 р.
3. Вихідні дані до роботи: розробити веб-застосунок на основі великих мовних моделей. Перелік використаних програмних засобів та технологій: Visual Studio Code, Next.js (Pages Router), OpenAI API, MongoDB Atlas, Auth0, Tailwind CSS, Vercel.
4. Перелік питань, що потрібно опрацювати в роботі: аналіз предметної області та принципів роботи великих мовних моделей, огляд та порівняльний аналіз існуючих рішень для побудови чат-інтерфейсів на базі мовних моделей, порівняльний аналіз підходів до потокової передачі даних (REST API, Server-Sent Events, WebSocket), обґрунтування вибору технологічного стеку для розробки веб-застосунку, визначення функціональних та нефункціональних вимог до системи, проєктування архітектури системи та розробка UML-діаграм, реалізація серверної частини застосунку з використанням Edge Functions та SSE, реалізація клієнтської частини застосунку на базі React та Next.js, реалізація системи автентифікації та авторизації користувачів, реалізація розширеного функціоналу управління розмовами, налаштування параметрів моделі та експорту даних, тестування продуктивності системи та оцінка якості відповідей моделі, забезпечення захисту інформації у веб-застосунку.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів,

комп’ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ___порівняльна схема підходів до потокової передачі даних (REST API, SSE, WebSocket), діаграма варіантів використання системи, діаграма компонентів системи, схема загальної архітектури системи, діаграма послідовності відправки повідомлення, схема колекцій бази даних MongoDB, порівняльна діаграма показників продуктивності.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання кваліфікаційної роботи	26.01.2026	виконано
2	Аналіз предметної області та огляд існуючих рішень	27.01.2026 – 06.02.2026	виконано
3	Порівняльний аналіз підходів до потокової передачі даних	07.02.2026 – 13.02.2026	виконано
4	Визначення вимог та обґрунтування технологічного стеку	14.02.2026 – 20.02.2026	виконано
5	Проєктування архітектури системи та розробка UML-діаграм	21.02.2026 – 28.02.2026	виконано
6	Розробка серверної частини та інтеграція з OpenAI API	01.03.2026 – 14.03.2026	виконано
7	Розробка клієнтської частини та системи автентифікації	15.03.2026 – 28.03.2026	виконано
8	Реалізація розширеного функціоналу	29.03.2026 – 08.04.2026	виконано
9	Розгортання застосунку на платформі Vercel	09.04.2026 – 11.04.2026	виконано
10	Тестування та оцінка якості застосунку	12.04.2026 – 22.04.2026	виконано
11	Оформлення пояснювальної записки	23.04.2026 – 03.05.2026	виконано
12	Оформлення презентаційних матеріалів до захисту	04.05.2026 – 08.05.2026	виконано
13	Представлення на рецензування	11.05.2026	виконано
14	Захист	15.05.2026	виконано

Дата видачі завдання _____26.01.2026_____

Здобувач _____

(підпис)

Керівник роботи _____ _____ ___доцент каф. Урняєва І.А.__

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи: 93 сторінки, 1 таблиця, 7 рисунків, 3 додатки, 25 джерел інформації.

ЧАТ-БОТ, ВЕЛИКІ МОВНІ МОДЕЛІ, ВЕБ-ЗАСТОСУНОК, ШТУЧНИЙ ІНТЕЛЕКТ, ПОТОКОВА ПЕРЕДАЧА ДАНИХ, ПЕРСОНАЛІЗАЦІЯ, API ІНТЕГРАЦІЯ, SERVER-SENT EVENTS, АУТЕНТИФІКАЦІЯ

Об'єктом дослідження є клієнт-серверна архітектура AI-powered веб-застосунку для взаємодії користувачів з великими мовними моделями.

Предметом дослідження є архітектурні рішення, що забезпечують потокову передачу відповідей, управління контекстом розмов та персоналізацію поведінки моделі.

Метою роботи є розробка архітектурних рішень та повнофункціонального веб-застосунку для взаємодії користувачів з великими мовними моделями через інтерфейс чат-бота, що забезпечує потокову передачу відповідей з низькою затримкою, управління історією розмов та персоналізацію поведінки моделі.

У роботі проведено порівняльний аналіз трьох підходів до потокової передачі даних (REST API, SSE, WebSocket) та обґрунтовано вибір Server-Sent Events як оптимального рішення з мінімальною затримкою 100–200 мс. Розроблено повнофункціональний веб-застосунок з управлінням розмовами на базі NoSQL бази даних, налаштуванням параметрів генерації, відстеженням токенів та експортом даних. Автентифікація реалізована через OAuth 2.0, серверний рівень побудовано на edge functions. Результати тестування підтвердили стабільність часових показників незалежно від довжини відповіді.

ABSTRACT

Master's thesis: 93 pages, 1 tables, 7 figures, 3 appendices, 25 references.

CHATBOT, LARGE LANGUAGE MODELS, WEB APPLICATION, ARTIFICIAL INTELLIGENCE, STREAMING DATA DELIVERY, PERSONALIZATION, API INTEGRATION, SERVER-SENT EVENTS, AUTHENTICATION

The object of research is the client-server architecture of an AI-powered web application for user interaction with large language models. The subject of research is the architectural solutions ensuring streaming response delivery, conversation context management, and model behavior personalization.

The aim of the work is to develop architectural solutions and a fully functional web application for user interaction with large language models through a chatbot interface, providing low-latency streaming response delivery, conversation history management, and model behavior personalization.

A comparative analysis of three streaming approaches (REST API, SSE, WebSocket) was conducted, justifying the selection of Server-Sent Events as the optimal solution with minimal latency of 100–200 ms. A fully functional web application was developed featuring conversation management on a NoSQL database, flexible generation parameter configuration, token usage tracking, and data export. Authentication is implemented via OAuth 2.0, and the server layer is built on edge functions. Performance testing confirmed the stability of time metrics regardless of response length.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AI – штучний інтелект

БД – база даних

ІС – інформаційна система

API – Application Programming Interface (інтерфейс програмування застосунків)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

DOM – Document Object Model (об'єктна модель документа)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (формат обміну даними)

LLM – Large Language Model (велика мовна модель)

NoSQL – Non-relational SQL (нереляційна база даних)

ODM – Object-Document Mapping (об'єктно-документне відображення)

REST – Representational State Transfer (архітектурний стиль передачі даних)

RLHF – Reinforcement Learning from Human Feedback (навчання з підкріпленням на основі зворотного зв'язку від людини)

SSE – Server-Sent Events (події, що надсилаються сервером)

SSR – Server-Side Rendering (серверний рендеринг)

TCP – Transmission Control Protocol (протокол керування передачею)

TTFT – Time To First Token (час до першого токена)

UI – User Interface (користувацький інтерфейс)

UML – Unified Modeling Language (уніфікована мова моделювання)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

XSS – Cross-Site Scripting (міжсайтовий скриптинг)

CSRF – Cross-Site Request Forgery (міжсайтова підробка запиту)

ВСТУП

Стрімкий розвиток великих мовних моделей (Large Language Models, LLM) у останні роки став одним із найважливіших проривів у галузі штучного інтелекту. Поява публічно доступних API для взаємодії з моделями GPT-4, Claude, Gemini та іншими відкрила можливості для створення інтелектуальних застосунків, здатних генерувати природномовні відповіді, допомагати у написанні коду, аналізі даних та вирішенні широкого спектра завдань. Однак масове впровадження таких технологій висуває нові виклики щодо створення зручних, ефективних та безпечних інтерфейсів для взаємодії кінцевих користувачів з AI-моделями.

Традиційні підходи до розробки чат-ботів зазвичай базувалися на обмежених rule-based системах або простих ML-моделях з фіксованим набором відповідей. Сучасні LLM надають принципово інші можливості – генерацію контекстно-залежних відповідей, підтримку складних діалогів та адаптацію до специфічних потреб користувача. Проте інтеграція таких моделей у веб-застосунки потребує вирішення низки технічних проблем: забезпечення потокової передачі відповідей для покращення користувацького досвіду, ефективного управління контекстом розмови, персистентного збереження історії діалогів, контролю використання токенів та можливості налаштування параметрів моделі.

Актуальність роботи обумовлена зростаючим попитом на AI-powered застосунки в різних сферах діяльності: від корпоративних асистентів до освітніх платформ. Розробка ефективних архітектурних рішень для інтеграції LLM у веб-застосунки є важливим завданням для забезпечення доступності та зручності використання технологій штучного інтелекту широким колом користувачів.

Об'єктом дослідження є процес взаємодії користувачів з великими мовними моделями через веб-інтерфейс чат-бота. Предметом дослідження виступають методи та технології створення веб-застосунків для інтеграції з API великих

мовних моделей, включаючи архітектурні рішення, підходи до потокової передачі даних та управління станом розмов.

Дослідження спрямоване на розробку повнофункціонального веб-застосунку для взаємодії користувачів з великими мовними моделями через інтерфейс чат-бота з персоналізованою роботою, збереженням історії розмов та гнучким налаштуванням параметрів генерації відповідей. Для досягнення цієї мети необхідно вирішити такі завдання: провести аналіз існуючих рішень для взаємодії з LLM та виявити їх переваги і недоліки; обґрунтувати вибір технологічного стеку для розробки веб-застосунку; розробити архітектуру системи з урахуванням вимог масштабованості та продуктивності; реалізувати механізм потокової передачі відповідей від API до клієнта; розробити систему управління історією розмов з використанням NoSQL бази даних; впровадити механізм персоналізації поведінки AI-моделі через налаштування параметрів; провести порівняльний аналіз різних підходів до передачі даних між сервером та клієнтом; виконати тестування продуктивності та якості роботи системи.

У роботі використано методи системного аналізу, об'єктно-орієнтованого проектування, веб-розробки, експериментального тестування та порівняльного аналізу архітектурних рішень.

Наукова новизна отриманих результатів полягає в розробці архітектури веб-застосунку, що поєднує потокову передачу відповідей через серверні функції з ефективним управлінням станом розмов та налаштуванням поведінки AI-моделі. Проведено порівняльний аналіз трьох підходів до інтеграції з API великих мовних моделей (REST API, Server-Sent Events, WebSocket) та обґрунтовано вибір оптимального рішення для задачі потокової передачі текстових відповідей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Великі мовні моделі: архітектура та програмні інтерфейси

1.1.1 Принципи роботи та покоління LLM

Великі мовні моделі (Large Language Models, LLM) являють собою нейронні мережі з мільярдами параметрів, навчені на величезних обсягах текстових даних для розуміння та генерації природної мови. Їх поява стала результатом еволюції архітектури Transformer, запропонованої Vaswani та співавторами у 2017 році [1]. Ключовою інновацією цієї архітектури став механізм уваги (attention mechanism), який дозволяє моделі ефективно обробляти послідовності довільної довжини та враховувати контекст при генерації відповідей.

Розвиток LLM можна умовно поділити на кілька поколінь. Перше покоління охоплює моделі на основі рекурентних нейронних мереж (RNN) та LSTM, які мали суттєві обмеження щодо обробки довгих послідовностей через проблему зникаючого градієнта. Друге покоління пов'язане з появою архітектури Transformer та моделей BERT і GPT-2, що продемонстрували значний прогрес у задачах розуміння та генерації тексту. Третє, сучасне покоління представлено масштабованими моделями GPT-3, GPT-4, Claude та LLaMA, які завдяки збільшенню кількості параметрів та обсягів навчальних даних досягли якісно нових можливостей.

Архітектура GPT (Generative Pre-trained Transformer) базується на decoder-only трансформері, який генерує текст послідовно, передбачаючи наступний токен на основі попередніх. Модель GPT-3, представлена у 2020 році, містила 175 мільярдів параметрів і продемонструвала вражаючі можливості у генерації тексту, перекладі та написанні коду [2]. Наступна ітерація, GPT-4, значно покращила якість відповідей та розширила контекстне вікно до 128 тисяч токенів.

Модель Claude, розроблена компанією Anthropic, використовує схожу архітектуру з додатковим акцентом на безпеку та вирівнювання з людськими

цінностями через методику Constitutional AI. Відкриті моделі сімейства LLaMA від Meta надають дослідникам та розробникам можливість експериментувати без обмежень закритих API.

Сучасні LLM демонструють здатність виконувати широкий спектр завдань без додаткового навчання. Ця властивість, відома як few-shot learning або zero-shot learning, дозволяє моделям адаптуватися до нових задач лише на основі текстового опису або кількох прикладів у промпті [3]. Важливим обмеженням залишаються так звані «галюцинації» — генерація фактично некоректної інформації з високою впевненістю, що потребує врахування при проектуванні застосунків.

1.1.2 OpenAI API: параметри, токени та режими взаємодії

Взаємодія з комерційними LLM здійснюється через API, які надають стандартизований інтерфейс для відправки запитів та отримання відповідей. OpenAI API є на сьогодні найбільш поширеним рішенням завдяки якості моделей, детальній документації та стабільності сервісу.

Ключовою одиницею тарифікації та обробки є токен. Токен приблизно відповідає 0,75 слова в англійській мові або одному символу в більшості інших мов, включаючи українську. Кожен запит до API витрачає токени як на вхідні дані (prompt_tokens), так і на згенеровану відповідь (completion_tokens). Загальна сума фіксується у полі total_tokens і безпосередньо визначає вартість запиту.

API дозволяє гнучко налаштовувати параметри генерації. Параметр temperature (діапазон 0,0–1,0) контролює ступінь випадковості відповідей: низькі значення забезпечують детерміновані та точні відповіді для технічних задач, тоді як високі значення збільшують різноманітність та креативність генерації. Параметр max_tokens обмежує максимальну довжину відповіді. Параметри top_p, presence_penalty та frequency_penalty додатково впливають на різноманітність та повторюваність тексту.

Важливим аспектом є управління контекстом розмови. Оскільки API є stateless, кожен запит повинен містити повну історію діалогу для збереження

контексту. Типовий запит включає масив повідомлень з ролями: system (визначає базову поведінку моделі), user (запити користувача) та assistant (попередні відповіді моделі) [4]. Це архітектурне обмеження безпосередньо впливає на проектування серверної логіки застосунку.

Однією з ключових особливостей сучасних API є підтримка streaming режиму, коли відповідь передається частинами у міру генерації, а не одним блоком після завершення. Це критично важливо для створення інтерактивних застосунків, оскільки дозволяє користувачу бачити початок відповіді через долі секунди після відправки запиту, а не очікувати кілька секунд до повного завершення генерації.

1.2 Огляд існуючих чат-бот рішень

1.2.1 Комерційні веб-інтерфейси

Сучасний ринок рішень для взаємодії з великими мовними моделями можна умовно поділити на три категорії: комерційні веб-інтерфейси провайдерів моделей, корпоративні платформи та open-source альтернативи.

ChatGPT від OpenAI є найбільш відомим та широко використовуваним веб-інтерфейсом для взаємодії з моделями GPT. Запущений у листопаді 2022 року, він встановив фактичний стандарт користувацького досвіду для таких застосунків. Ключові особливості включають потокову передачу відповідей, збереження історії розмов, можливість редагування попередніх повідомлень та організацію діалогів у окремі треди [5].

Інтерфейс Claude.ai від Anthropic пропонує схожий користувацький досвід із додатковими функціями: завантаження документів для аналізу, Artifacts для створення інтерактивного контенту та проекти з власною базою знань. Особливістю є більше контекстне вікно — до 200 тисяч токенів, що дозволяє обробляти цілі книги або великі кодові бази в одному запиті.

Google Gemini інтегрований з екосистемою Google-сервісів, що дозволяє йому отримувати доступ до Gmail, Google Drive та Maps для надання

персоналізованих відповідей. Gemini також підтримує мультимодальність — обробку не лише тексту, але й зображень.

Порівняльний аналіз комерційних інтерфейсів виявляє спільні характеристики, що стали де-факто стандартом: потокова передача відповідей, персистентна історія розмов, можливість налаштування поведінки моделі та адаптивний інтерфейс. Разом з тим ці рішення мають суттєві обмеження щодо кастомізації та інтеграції у власні продукти.

1.2.2 Корпоративні платформи

Корпоративні платформи для створення чат-ботів, такі як Microsoft Copilot Studio, IBM watsonx Assistant та Amazon Lex, орієнтовані на бізнес-застосування з можливістю інтеграції у існуючі корпоративні системи. Ці рішення зазвичай надають розширені можливості для налаштування, включаючи підключення до внутрішніх баз даних, інтеграцію з CRM-системами та контроль доступу на рівні організації [6].

Microsoft Copilot Studio дозволяє створювати ботів з використанням низькокодового підходу, де діалоги проектуються візуально через графічний інтерфейс. Платформа підтримує інтеграцію з Azure OpenAI Service, що дає змогу використовувати GPT-моделі з дотриманням корпоративних політик безпеки.

IBM watsonx Assistant фокусується на створенні голосових та текстових асистентів для обслуговування клієнтів, поєднуючи власні LLM з класичними NLP-підходами для розпізнавання намірів та видобування сутностей. Це дозволяє будувати більш детерміновані діалоги для специфічних бізнес-процесів.

Основним недоліком корпоративних платформ є висока вартість та складність початкового налаштування, що робить їх доцільними лише для великих організацій з відповідними ресурсами.

1.2.3 Open-source альтернативи

Серед open-source рішень виділяються BetterChatGPT, Jan.ai та Open WebUI. Ці проекти надають базову функціональність для взаємодії з різними LLM API та можуть бути розгорнуті на власній інфраструктурі, що важливо для організацій з жорсткими вимогами до конфіденційності даних.

BetterChatGPT є веб-застосунком, що працює повністю у браузері користувача, зберігаючи API-ключі та історію розмов локально. Це забезпечує приватність, але обмежує можливості синхронізації між пристроями.

Jan.ai позиціонується як open-source альтернатива ChatGPT з можливістю запуску моделей локально без необхідності підключення до зовнішніх API. Застосунок підтримує різні формати моделей, включаючи GGUF для квантованих версій LLaMA.

Open WebUI надає розширений веб-інтерфейс для роботи з локальними моделями через Ollama, а також підтримує підключення до зовнішніх API. Особливості включають RAG (Retrieval-Augmented Generation) для роботи з документами, управління промптами та голосове введення [7].

Аналіз показує, що жодне з існуючих рішень не є універсальним. Комерційні інтерфейси надають найкращий користувацький досвід, але мають обмеження у кастомізації. Корпоративні платформи забезпечують необхідні можливості для бізнесу, але вимагають значних інвестицій. Open-source альтернативи дають повний контроль, але потребують технічної експертизи для розгортання та підтримки.

1.3 Технологічний стек для AI-орієнтованих веб-застосунків

1.3.1 Клієнтські фреймворки та мета-фреймворки

Для розробки клієнтської частини AI-застосунків найбільш поширеними є JavaScript-фреймворки React, Vue.js та Angular. React, розроблений та підтримуваний Meta, базується на компонентному підході та використовує віртуальний DOM для ефективного оновлення лише змінених частин сторінки.

Його екосистема включає велику кількість готових бібліотек для роботи з формами, маршрутизацією та управлінням станом [8].

Для створення повноцінних веб-застосунків з серверним рендерингом широко використовуються мета-фреймворки: Next.js (для React), Nuxt.js (для Vue) та SvelteKit. Next.js надає серверний рендеринг (SSR), генерацію статичних сторінок (SSG), API Routes для створення backend-ендпоінтів та автоматичне розбиття коду для оптимізації розміру бандлів [9].

Ключовою перевагою Next.js для AI-застосунків є підтримка Edge Runtime та Edge Functions. Edge-функції виконуються на серверах, географічно наближених до користувача, що зменшує латентність запитів до API. Це особливо важливо для потокової передачі відповідей від LLM, де кожна додаткова мілісекунда затримки впливає на користувацький досвід.

Для управління станом застосунку використовуються бібліотеки Redux, Zustand або вбудовані механізми React — Context API та hooks. Zustand надає мінімалістичний API для створення глобального стану з автоматичним перерендерингом компонентів при зміні даних та не вимагає написання boilerplate-коду, характерного для Redux.

1.3.2 Бази даних, автентифікація та хостинг

Для зберігання даних в AI-застосунках вибір типу бази даних залежить від специфіки задачі. Реляційні бази даних PostgreSQL та MySQL підходять для структурованих даних зі складними зв'язками. NoSQL бази даних MongoDB, Firebase Firestore та DynamoDB краще підходять для зберігання розмов у форматі документів [10].

MongoDB зберігає дані у BSON-форматі, що природно відображає структуру JavaScript-об'єктів. Документо-орієнтована модель дозволяє зберігати повну розмову як один документ з вкладеним масивом повідомлень, що спрощує отримання історії діалогу одним запитом. Для роботи з MongoDB у Node.js

застосунках використовується ODM-бібліотека Mongoose, що надає схеми для валідації даних та middleware для додаткової логіки.

Автентифікація користувачів у сучасних застосунках реалізується переважно через OAuth 2.0 / OpenID Connect провайдерів: Auth0, Firebase Authentication, Clerk або Supabase Auth. Auth0 надає готове рішення з підтримкою social login (Google, GitHub), multi-factor authentication та управлінням користувачами через адміністративну панель. Інтеграція з Next.js здійснюється через бібліотеку @auth0/nextjs-auth0 [11].

Для хостингу Next.js застосунків платформа Vercel є природним вибором завдяки нативній інтеграції та можливості автоматичного масштабування serverless-функцій. Edge Network Vercel забезпечує глобальну доступність із низькою латентністю через CDN для статичних ресурсів та edge-функції для динамічного контенту.

1.4 Порівняльний аналіз підходів до потокової передачі даних

Ефективна передача даних між сервером та клієнтом є критично важливою для створення responsive-інтерфейсів AI-застосунків. Традиційна модель HTTP базується на циклі request-response, де клієнт ініціює запит, сервер обробляє його та повертає повну відповідь. Для задач генерації тексту LLM, де процес може тривати від кількох секунд до хвилини, такий підхід створює негативний користувацький досвід через тривале очікування без жодного зворотного зв'язку.

Потокова передача даних дозволяє серверу відправляти частини відповіді клієнту у міру їх готовності, забезпечуючи миттєвий зворотний зв'язок та створюючи природний ефект послідовного формулювання відповіді.

1.4.1 Server-Sent Events

Технологія Server-Sent Events (SSE) є стандартом W3C для однонаправленої потокової передачі подій від сервера до клієнта через HTTP [12]. SSE

використовує звичайне HTTP-з'єднання з Content-Type: text/event-stream та передає текстові повідомлення у форматі, де кожна подія відокремлюється двома символами нового рядка.

На клієнтській стороні SSE реалізується через вбудований у всі сучасні браузери API EventSource. EventSource автоматично встановлює з'єднання, обробляє reconnection у разі розриву та надає події для отримання даних.

Переваги SSE: простота реалізації на основі стандартного HTTP, автоматичне відновлення з'єднання, ефективне використання ресурсів та нативна підтримка у браузерах. Час до першого токена при SSE становить 100–500 мс. Обмеження: однонаправлений характер комунікації та можливі проблеми з корпоративними proxy.

1.4.2 WebSocket

WebSocket є протоколом повнодуплексної комунікації поверх TCP, що забезпечує двосторонній обмін даними через одне тривале з'єднання [13]. Встановлення з'єднання починається з HTTP upgrade-запиту, після якого протокол переключається з HTTP на WebSocket і обидві сторони можуть відправляти повідомлення у будь-який момент.

Переваги WebSocket: двостороння комунікація в реальному часі, низька латентність через відсутність HTTP-заголовків для кожного повідомлення та можливість передачі бінарних даних. WebSocket ідеально підходить для застосунків з високою інтерактивністю — онлайн-ігор, collaborative editing [14].

Недоліки: висока складність реалізації, необхідність управління станом з'єднання на сервері, що ускладнює горизонтальне масштабування, та відсутність автоматичного reconnect у базовому API.

1.4.3 Традиційний REST API

Підхід REST API без streaming передбачає відправку запиту та очікування повної відповіді перед її відображенням. Для AI-застосунків це означає, що користувач не бачить жодного тексту протягом усього часу генерації, що може становити 10–30 секунд для довгих відповідей.

Переваги цього підходу: максимальна простота реалізації, повна сумісність з будь-якою інфраструктурою та зручність кешування. Однак для інтерактивних AI-застосунків відсутність проміжного зворотного зв'язку є критичним недоліком.

На рисунку 1.1 наведено порівняльну схему трьох підходів за ключовими характеристиками.

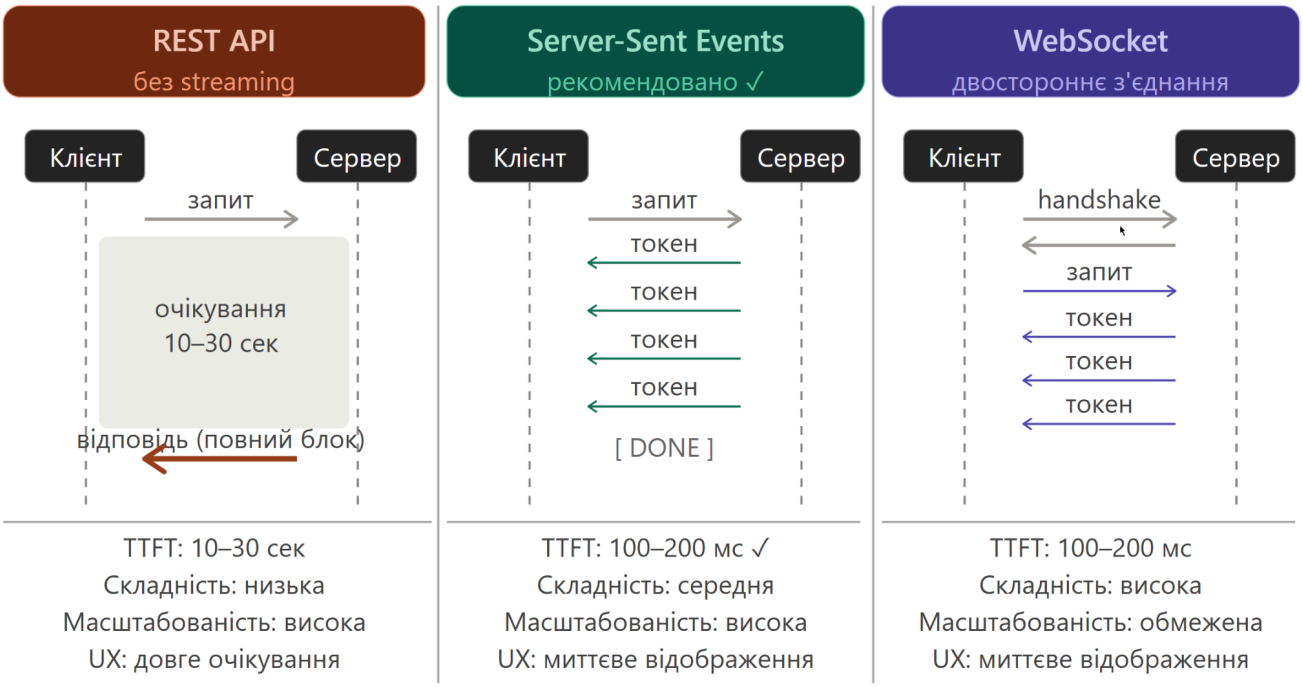


Рисунок 1.1 — Порівняльна схема підходів до потокової передачі даних

Порівняльний аналіз показує, що SSE є найбільш збалансованим вибором для AI-застосунків з фокусом на генерації текстових відповідей. Він забезпечує необхідну потокову передачу з мінімальною складністю реалізації та має хорошу сумісність із serverless і edge functions.

Роблячи підсумок, у розділі проведено аналіз предметної області та огляд існуючих технологій для створення веб-застосунків на базі великих мовних моделей.

Розглянуто еволюцію LLM від моделей на основі RNN до сучасних масштабованих трансформерів, а також принципи взаємодії з ними через API — управління токенами, параметрами генерації та контекстом розмови. Встановлено, що stateless-природа API є ключовим архітектурним обмеженням, яке необхідно враховувати при проектуванні серверної логіки.

Проаналізовано три категорії існуючих рішень для побудови чат-інтерфейсів: комерційні веб-інтерфейси, корпоративні платформи та open-source альтернативи. Виявлено, що жодне з них не є універсальним, а розробка власного застосунку дозволяє поєднати якість користувацького досвіду комерційних рішень з гнучкістю open-source підходу.

Досліджено веб-технології для розробки клієнтської та серверної частини, включаючи React, Next.js, MongoDB та Auth0. Особливу увагу приділено edge computing та serverless-архітектурам, що забезпечують низьку латентність та автоматичне масштабування.

Проведено порівняльний аналіз трьох підходів до потокової передачі даних: SSE, WebSocket та REST API. Встановлено, що Server-Sent Events надає оптимальний баланс між простотою реалізації, ефективністю використання ресурсів та якістю користувацького досвіду для задачі передачі відповідей від LLM.

2 РОЗРОБКА ВИМОГ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Постановка задачі

2.1.1 Функціональні вимоги до системи

На основі аналізу предметної області та огляду існуючих рішень сформульовано задачу розробки веб-застосунку для взаємодії користувачів з великими мовними моделями через інтерфейс чат-бота. Застосунок повинен забезпечувати зручну взаємодію з AI-моделлю, персоналізацію поведінки системи та збереження історії діалогів для подальшого використання.

Функціональні вимоги визначають конкретні можливості системи, що безпосередньо надаються кінцевому користувачу. Систему вимог структуровано за функціональними модулями:

1) Управління розмовами. Система має забезпечувати створення нових розмов та управління існуючими діалогами через інтерфейс користувача. Кожна розмова повинна зберігати повну історію обміну повідомленнями між користувачем та моделлю з можливістю перегляду в будь-який момент. Передбачено можливість перейменування розмов, їх видалення, а також пошук серед збережених діалогів за назвою.

2) Генерація відповідей. Система має підтримувати потокову передачу відповідей від моделі, відображаючи текст у міру його генерації без очікування повного завершення процесу. Користувач повинен мати можливість перервати генерацію в будь-який момент.

3) Налаштування параметрів моделі. Користувач повинен мати можливість налаштовувати параметри генерації відповідей, включаючи temperature для контролю креативності, максимальну кількість токенів у відповіді та системний промпт для визначення базової поведінки моделі. Передбачено збереження готових шаблонів налаштувань для типових сценаріїв використання.

4) Відстеження токенів. Система має відстежувати використання токенів для кожного запиту та в розрізі всієї розмови, надаючи користувачу інформацію про кількість prompt_tokens, completion_tokens та total_tokens.

5) Експорт даних. Користувач повинен мати можливість експортувати історію розмов у форматах JSON, Markdown та TXT для зберігання або подальшої обробки.

6) Автентифікація. Система підтримує реєстрацію та вхід через email або OAuth-провайдерів. Кожен користувач має доступ виключно до власних розмов та налаштувань.

2.1.2 Нефункціональні вимоги до системи

Нефункціональні вимоги визначають якісні характеристики системи та обмеження на її реалізацію. Включають в себе:

1) Продуктивність. Час до першого токена відповіді (TTFT) має не перевищувати 200 мілісекунд. Швидкість генерації повинна становити не менше 40 токенів на секунду. Інтерфейс повинен залишатися responsive під час генерації відповіді.

2) Масштабованість. Архітектура системи повинна забезпечувати можливість обробки зростаючої кількості користувачів без деградації продуктивності. Використання serverless та edge computing дозволяє автоматично масштабувати ресурси відповідно до поточного навантаження.

3) Надійність. Система повинна коректно обробляти помилки на всіх рівнях з інформативними повідомленнями для користувача. У разі збою під час генерації відповіді — відновлюватися без втрати попередньої історії розмови.

4) Безпека. API-ключі не повинні передаватися на клієнтську сторону — всі запити до LLM API проходять виключно через серверний рівень. Система повинна бути захищена від типових веб-вразливостей: XSS та CSRF атак. Усі запити до захищених ендпоінтів проходять перевірку автентифікації.

5) Зручність використання. Інтерфейс повинен не вимагати технічних знань для базових операцій. Система надає підказки для налаштування параметрів моделі та шаблони промптів для різних сценаріїв використання.

6) Управління контекстом. Оскільки контекстне вікно моделі є обмеженим, система повинна автоматично обмежувати кількість повідомлень, що передаються в одному запиті, зберігаючи 10–15 останніх повідомлень для підтримки природного діалогу.

2.2 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку базується на аналізі вимог, технічних обмежень та сучасних практик веб-розробки. Рішення приймалися з урахуванням балансу між продуктивністю, зручністю розробки та можливостями для подальшого розширення функціональності.

2.2.1 Фронтенд: React та Next.js Pages Router

Для клієнтської частини обрано React як основну бібліотеку для побудови користувацького інтерфейсу. React забезпечує ефективне оновлення DOM через віртуальний DOM та reconciliation-алгоритм, що критично важливо для відображення потокових відповідей з високою частотою оновлень. Компонентна архітектура дозволяє створювати модульні інтерфейсні елементи з чіткою ізоляцією відповідальності та можливістю повторного використання.

Next.js обрано як мета-фреймворк для React, що надає серверний рендеринг, file-based routing, API Routes та вбудовану оптимізацію продуктивності. Використовується Pages Router — система маршрутизації на основі файлової структури директорії pages, де кожен файл автоматично перетворюється на URL-маршрут. Це спрощує структуру проєкту та зменшує обсяг boilerplate-коду.

API Routes у Next.js дозволяють створювати serverless-функції безпосередньо у структурі проєкту без необхідності розгортання окремого

backend-сервера. Edge Functions використовуються для ендпоінтів, що обробляють streaming відповіді від OpenAI API, забезпечуючи низьку латентність через виконання на географічно розподілених серверах.

Для управління глобальним станом застосунку обрано Zustand — бібліотеку з мінімалістичним API, що не потребує boilerplate-коду характерного для Redux. Store містить масив розмов, активну розмову та методи для CRUD операцій.

Альтернативами розглядалися Vue.js з Nuxt.js та SvelteKit. Обрано Next.js завдяки найбільшій екосистемі, нативній підтримці Edge Functions, тісній інтеграції з платформою Vercel та TypeScript підтримці з коробки.

2.2.2 База даних: MongoDB та Mongoose

MongoDB обрано як основну базу даних для зберігання розмов та налаштувань користувачів. Документо-орієнтована модель MongoDB природно відповідає структурі даних чат-застосунку: розмова представлена як документ з вкладеним масивом повідомлень. Це дозволяє отримати повну історію діалогу одним запитом без складних JOIN-операцій, характерних для реляційних баз даних.

Гнучка схема MongoDB дозволяє додавати нові поля до документів без міграцій, що прискорює ітеративну розробку. MongoDB Atlas використовується як managed-сервіс, що забезпечує автоматичні резервні копії, моніторинг продуктивності та масштабування без простоїв.

Mongoose використовується як ODM для взаємодії з MongoDB, надаючи схеми для валідації даних, middleware для додаткової логіки та зручні методи для типових операцій. TypeScript-інтеграція Mongoose забезпечує типобезпеку на етапі компіляції.

Альтернативами розглядалися PostgreSQL з Prisma та Firebase Firestore. Обрано MongoDB через більш природну відповідність структури документів структурі чат-розмов та простоту горизонтального масштабування.

2.2.3 Автентифікація: Auth0

Auth0 обрано як провайдер автентифікації користувачів. Auth0 надає готове рішення з підтримкою email/password реєстрації, social login через Google та GitHub, управлінням сесіями та захистом від типових атак. Це дозволяє уникнути реалізації власної системи автентифікації з усіма пов'язаними безпековими ризиками.

Інтеграція здійснюється через бібліотеку @auth0/nextjs-auth0, що надає готові API Routes для login, logout та callback-ендпоінтів, а також hook useUser для доступу до даних поточного користувача в компонентах. Auth0 автоматично обробляє хешування паролів, ротацію refresh-токенів та захист від brute-force атак.

Альтернативами розглядалися Firebase Authentication, Clerk та NextAuth.js. Auth0 обрано через зрілість платформи, розширені можливості адміністрування та надійну документацію для інтеграції з Next.js.

2.2.4 Стилізація та хостинг: Tailwind CSS та Vercel

Tailwind CSS обрано для стилізації інтерфейсу завдяки utility-first підходу, що дозволяє швидко створювати адаптивні дизайни безпосередньо в JSX-розмітці без написання власного CSS. Tailwind забезпечує консистентність дизайну через визначену систему відступів, кольорів та типографіки. Purge-функція автоматично видаляє невикористані класи з продакшн-білду, мінімізуючи розмір CSS-файлу.

Vercel обрано як платформу для розгортання завдяки нативній інтеграції з Next.js та автоматичному масштабуванню serverless-функцій. Vercel автоматично створює preview deployment для кожного pull request, що спрощує процес code review. Edge Network забезпечує глобальну доступність застосунку з низькою латентністю через CDN для статичних ресурсів та edge-функції для динамічного контенту.

TypeScript використовується для всього кодбейзу, забезпечуючи статичну типізацію та покращений developer experience. Це зменшує кількість

runtime-помилок та полегшує рефакторинг завдяки виявленню помилок на етапі компіляції.

У таблиці 2.1 представлено зведене обґрунтування вибору технологічного стеку.

Таблиця 2.1 — Обґрунтування вибору технологічного стеку

Компонент	Обране рішення	Альтернативи	Обґрунтування
Frontend	React	Vue.js, Angular	Велика екосистема, ефективний virtual DOM, компонентна архітектура
Мета-фреймворк	Next.js (Pages Router)	Remix, SvelteKit, Nuxt	SSR, API Routes, Edge Functions, Vercel-інтеграція
База даних	MongoDB	PostgreSQL, Firebase	Документна модель, гнучка схема, managed сервіс Atlas
ODM	Mongoose	Prisma, TypeORM	Нативна MongoDB-підтримка, middleware, схеми валідації
Автентифікація	Auth0	Firebase Auth, Clerk, NextAuth	Готове рішення, social login, безпека, managed сервіс
Стилізація	Tailwind CSS	CSS Modules, Styled Components	Швидкість розробки, utility-first, мінімальний bundle
LLM API	OpenAI API	Anthropic, Google AI	Найкраща документація, streaming підтримка, стабільність
Хостинг	Vercel	Netlify, AWS, Railway	Next.js оптимізація, Edge Functions, автоматичне масштабування
Мова	TypeScript	JavaScript	Типобезпека, autocomplete, виявлення помилок на етапі компіляції

2.3 Проєктування системи засобами UML

Для формального опису архітектури та поведінки системи розроблено комплект UML-діаграм, що охоплюють три ключових аспекти: функціональні можливості системи з точки зору користувача, структурну організацію компонентів та динаміку взаємодії між ними під час виконання типових сценаріїв.

2.3.1 Розробка діаграми варіантів використання

Діаграма варіантів використання описує функціональні можливості системи з точки зору акторів, що з нею взаємодіють. В системі визначено двох основних акторів: Користувач (автентифікований) та Система автентифікації (Auth0).

Неавтентифікований відвідувач має доступ лише до сторінки входу та реєстрації. Після успішної автентифікації користувач отримує доступ до повного набору функцій застосунку.

Автентифікований користувач може виконувати такі дії:

– керувати розмовами: створювати нові, переглядати список існуючих, перейменовувати та видаляти;

– надсилати повідомлення та отримувати відповіді від AI-моделі у потоковому режимі; – налаштовувати параметри моделі: змінювати temperature, max_tokens та системний промпт;

– переглядати статистику використання токенів для поточної розмови;

– виконувати пошук серед збережених розмов;

– експортувати розмови у форматах JSON, Markdown та TXT;

– виходити з системи.

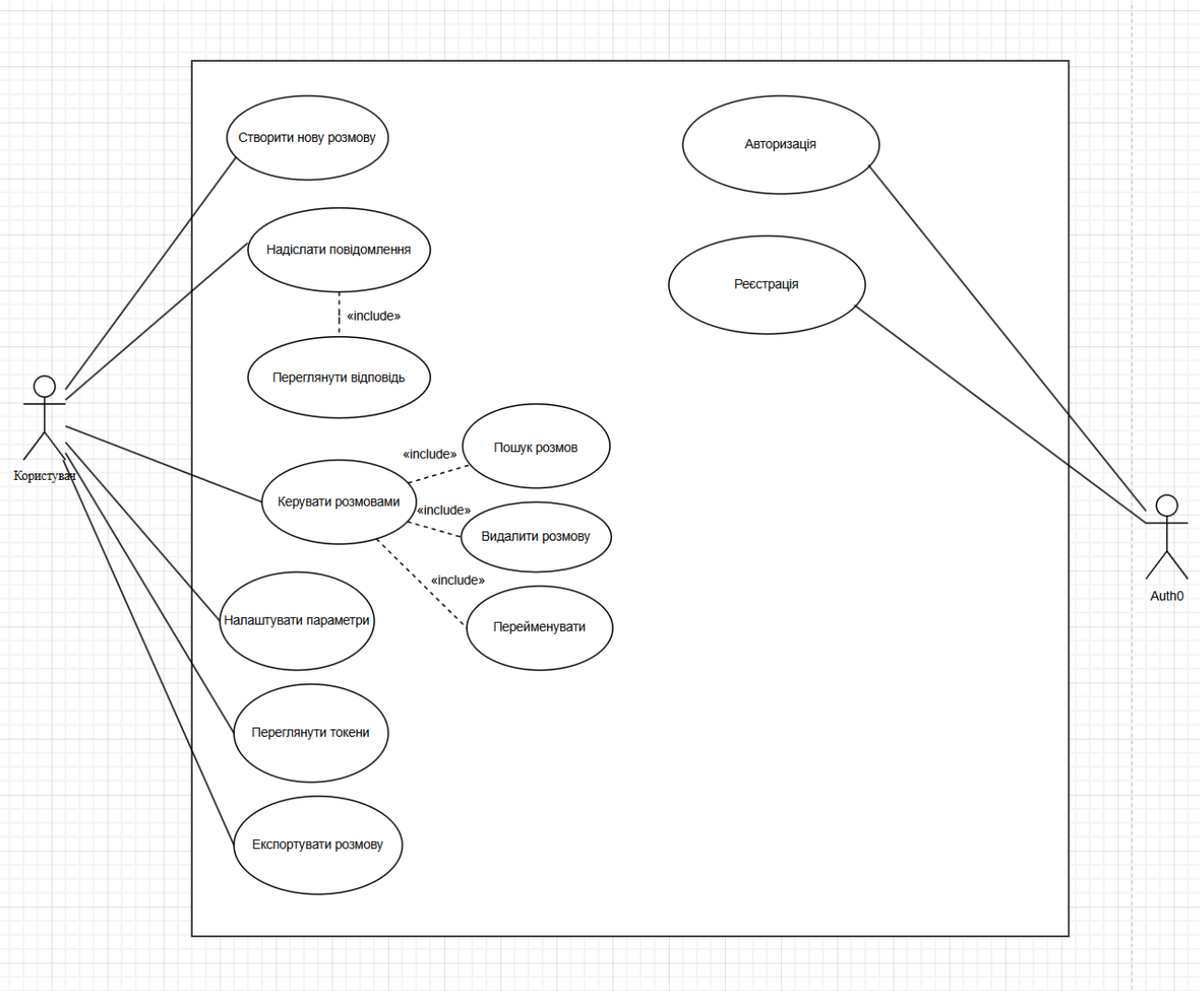


Рисунок 2.1 — Діаграма варіантів використання системи

2.3.2 Розробка діаграми компонентів

Діаграма компонентів відображає структурну організацію системи та залежності між її складовими. Система поділена на три основні рівні: клієнтський, серверний та рівень зовнішніх сервісів.

Клієнтський рівень містить такі компоненти: ChatPage (головна сторінка застосунку), Sidebar (навігація та список розмов), ChatWindow (область відображення повідомлень), MessageInput (поле введення та відправки), SettingsPanel (налаштування параметрів моделі) та TokenCounter (відображення статистики токенів). Компоненти взаємодіють через Zustand Store, що є централізованим сховищем стану застосунку.

Серверний рівень реалізований через Next.js API Routes та Edge Functions. Ключові ендпоінти: /api/auth/* (обробка автентифікації через Auth0), /api/chat/sendMessage (Edge Function для streaming відповідей), /api/chat/* (CRUD операції з розмовами), /api/settings/* (управління налаштуваннями). Middleware автентифікації захищає всі ендпоінти, окрім публічних сторінок.

Рівень зовнішніх сервісів включає: OpenAI API (генерація відповідей), Auth0 (автентифікація та авторизація) та MongoDB Atlas (персистентне зберігання даних).

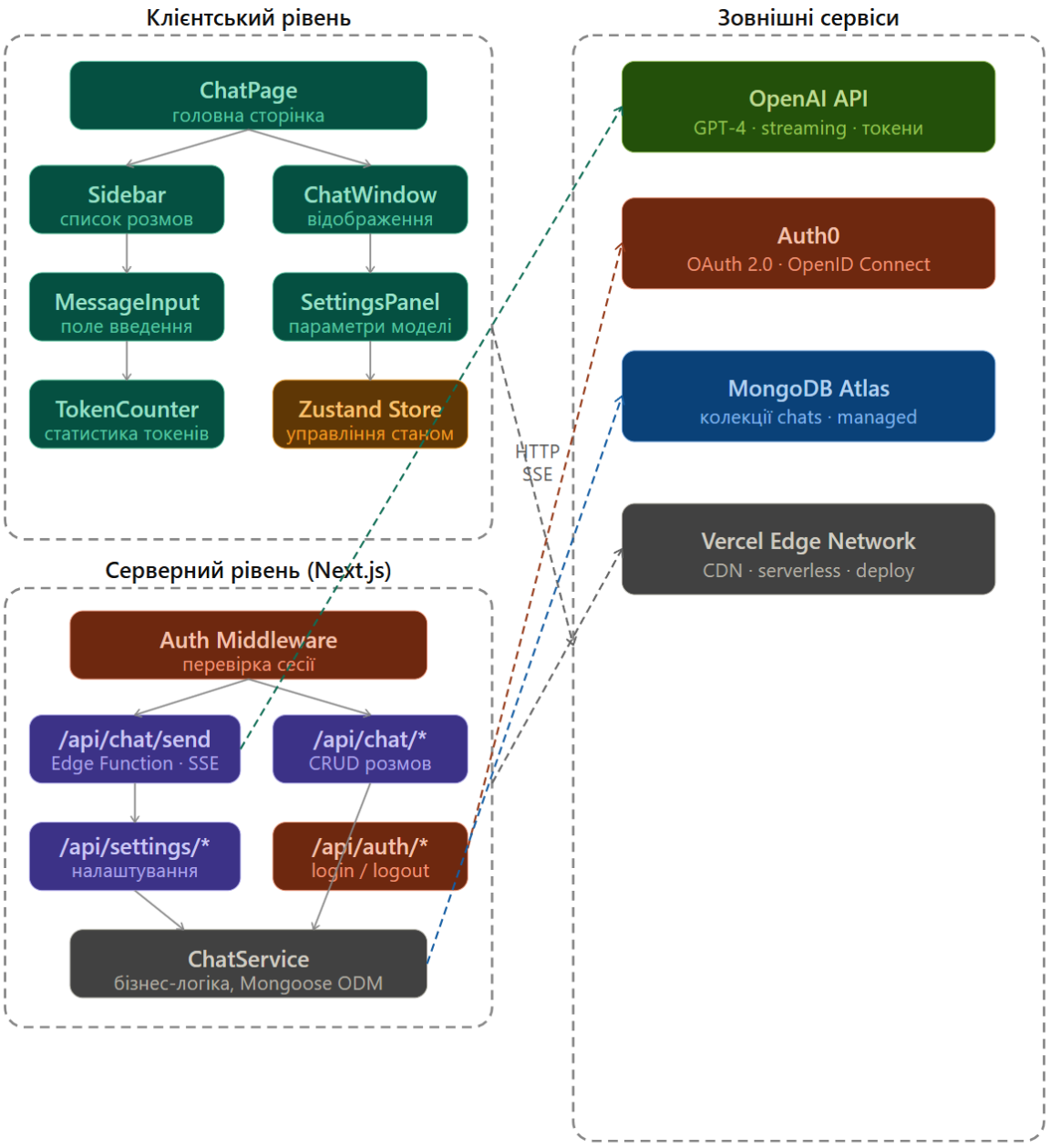


Рисунок 2.2 — Діаграма компонентів системи

2.3.3 Розробка діаграми послідовності дій

Діаграма послідовності описує динаміку взаємодії між компонентами системи під час виконання основного сценарію — відправки повідомлення та отримання streaming відповіді.

Послідовність дій включає такі кроки:

1) Користувач вводить повідомлення у компонент MessageInput та натискає кнопку відправки.

2) ChatPage викликає метод sendMessage зі Zustand Store, що встановлює SSE-з'єднання з ендпоінтом /api/chat/sendMessage.

3) Middleware автентифікації перевіряє валідність сесії через Auth0. У разі відсутності сесії повертається помилка 401.

4) Edge Function завантажує історію розмови з MongoDB та формує масив повідомлень для API.

5) Edge Function ініціює streaming запит до OpenAI API з параметрами генерації.

6) Кожен згенерований токен одразу передається клієнту через SSE-потік.

7) ChatWindow отримує токени та оновлює відображення повідомлення в реальному часі.

8) Після отримання сигналу [DONE] Edge Function зберігає повне повідомлення та метадані токенів у MongoDB.

9) ChatPage оновлює стан розмови через Zustand Store.

Весь процес можна побачити на рисунку 2.3 у відповідній діаграмі послідовності.

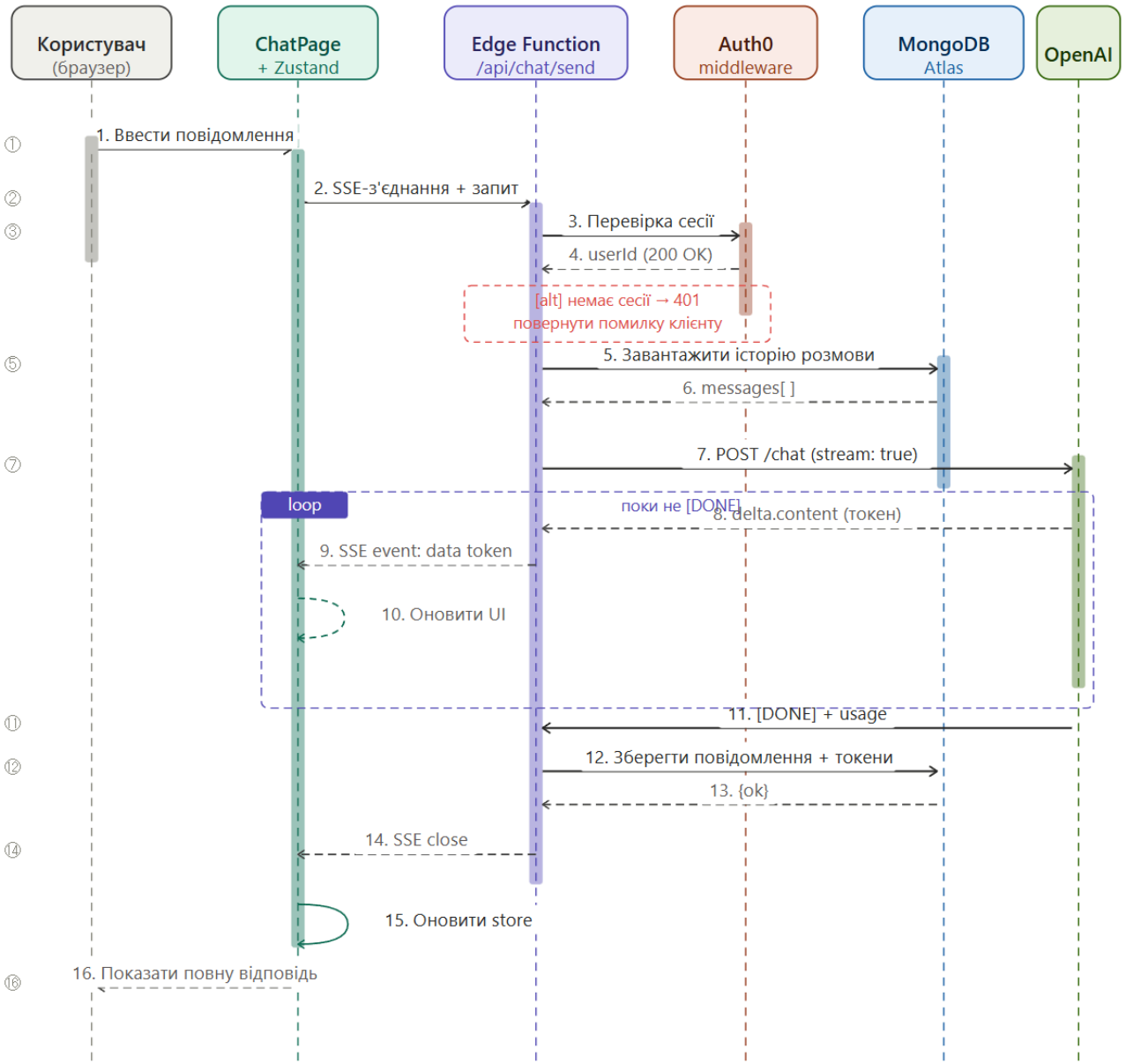


Рисунок 2.3 — Діаграма послідовності відправки повідомлення

2.4 Архітектура розробленої системи

2.4.1 Загальна архітектура та рівні застосунку

Архітектура веб-застосунку побудована за принципом розділення відповідальності (Separation of Concerns) між трьома рівнями: клієнтським, серверним та рівнем збереження даних. Це забезпечує модульність, тестованість та можливість незалежного розвитку кожного компонента системи.

Клієнтський рівень реалізує презентаційну логіку та безпосередню взаємодію з користувачем через React-компоненти. Компоненти організовані за функціональними модулями: автентифікація, управління розмовами, інтерфейс чату, налаштування параметрів моделі. Клієнтський рівень не містить бізнес-логіки та не взаємодіє з зовнішніми API безпосередньо — всі запити проходять через серверний рівень.

Серверний рівень реалізований через Next.js API Routes та Edge Functions. Edge Functions використовуються для ендпоінтів, критичних до латентності — передусім для обробки streaming відповідей від OpenAI API. Звичайні API Routes обробляють CRUD операції з базою даних. Middleware автентифікації виконується перед кожним захищеним ендпоінтом.

Рівень збереження даних забезпечується MongoDB Atlas — хмарним managed-сервісом. Mongoose ODM надає об'єктно-документне відображення між JavaScript-об'єктами та документами MongoDB. Зовнішні сервіси (OpenAI API, Auth0) взаємодіють із системою виключно через серверний рівень.

2.4.2 Схема взаємодії між компонентами

На рисунку 2.4 зображено загальну архітектуру розробленого веб-застосунку з основними компонентами та потоками даних.

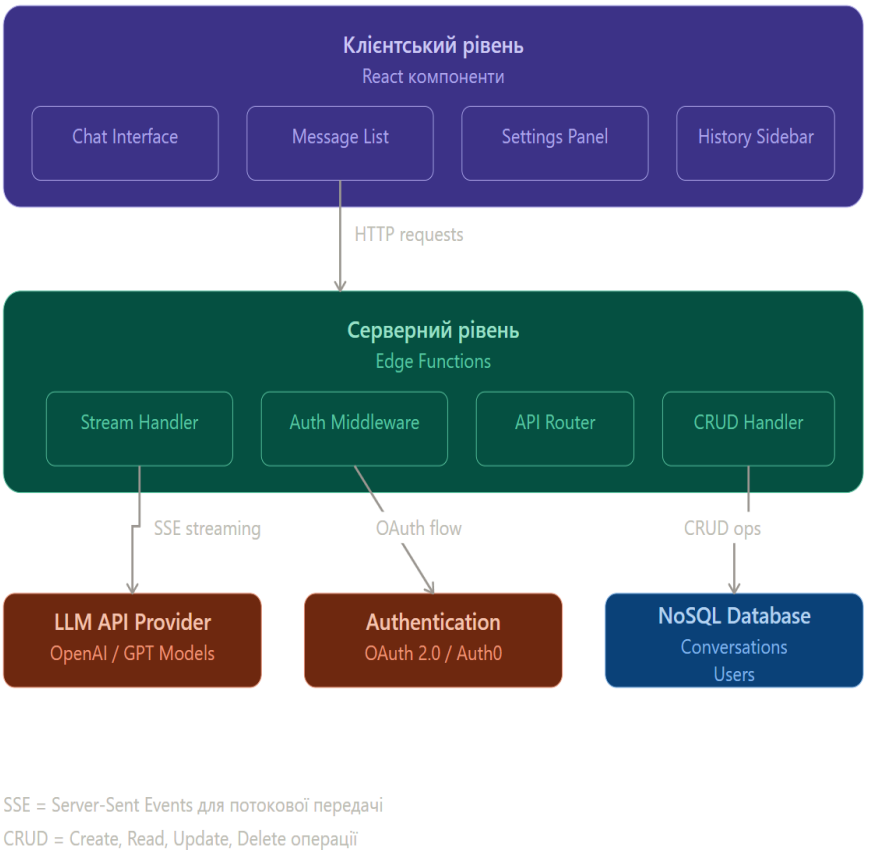


Рисунок 2.4 — Архітектура системи

Клієнтський рівень комунікує з серверним через HTTPS, відправляючи REST-запити для CRUD операцій та встановлюючи SSE-з'єднання для streaming відповідей. Серверний рівень авторизує запити через Auth0, виконує бізнес-логіку, взаємодіє з MongoDB для персистентності даних та викликає OpenAI API для генерації відповідей.

Потік даних для типового запиту охоплює такі кроки: компонент встановлює SSE-з'єднання з /api/chat/sendMessage; middleware перевіряє автентифікацію; Edge Function завантажує контекст розмови з MongoDB; формується запит до OpenAI API; streaming відповідь передається клієнту по токену; після завершення повне повідомлення зберігається у MongoDB.

2.4.3 Схема колекцій бази даних

База даних містить дві основні колекції: chats та users. Колекція users управляється Auth0 і зберігає базову інформацію про акаунт. Колекція chats є центральною для застосунку.

Схема документа колекції chats містить такі поля: _id (унікальний ідентифікатор MongoDB), userId (ідентифікатор користувача з Auth0 для ізоляції даних), title (назва розмови), messages (масив вкладених документів), settings (об'єкт з параметрами генерації), createdAt та updatedAt (мітки часу).

Кожен елемент масиву messages містить: role (system | user | assistant), content (текст повідомлення), timestamp та tokenUsage (об'єкт з полями promptTokens, completionTokens, totalTokens).

Об'єкт settings зберігає: temperature (число від 0.0 до 1.0), maxTokens (ціле число), systemPrompt (рядок).

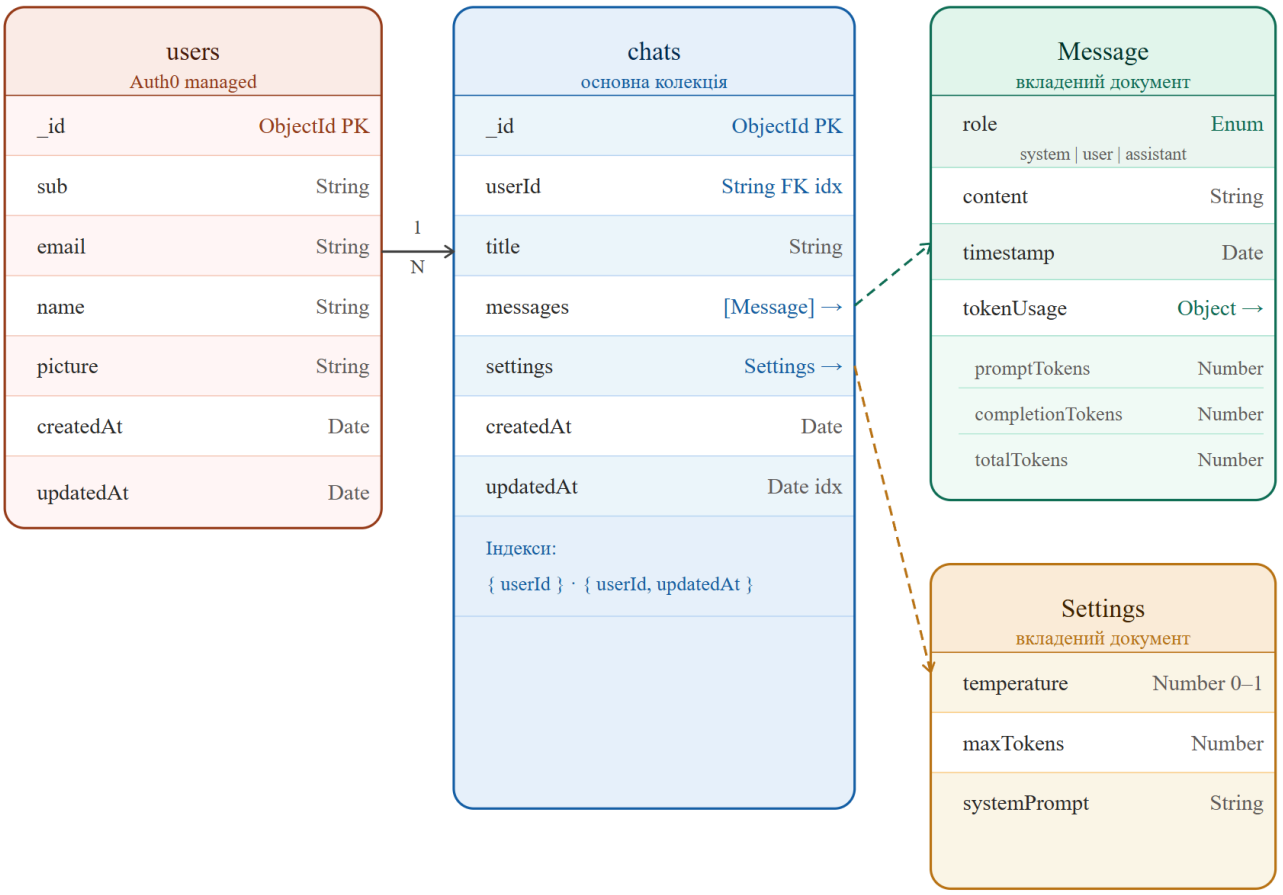


Рисунок 2.5 — Схема колекцій бази даних

Індекси створено на полі userId для швидкого пошуку розмов конкретного користувача та на парі (userId, updatedAt) для ефективного сортування списку розмов у хронологічному порядку. Compound-індекс оптимізує найбільш частий запит — отримання списку розмов користувача, відсортованого за часом останнього оновлення.

Роблячи підсумок, у розділі проведено постановку задачі, обґрунтовано вибір технологічного стеку та виконано проєктування системи.

Визначено функціональні вимоги до системи, що охоплюють управління розмовами, потокову генерацію відповідей, налаштування параметрів моделі, відстеження токенів та експорт даних. Сформульовано нефункціональні вимоги щодо продуктивності (TTFT ≤ 200 мс), масштабованості, надійності, безпеки та зручності використання.

Обґрунтовано вибір технологічного стеку: React та Next.js (Pages Router) для клієнтської частини та serverless-бекенду, MongoDB з Mongoose для зберігання даних, Auth0 для автентифікації, Tailwind CSS для стилізації та Vercel для хостингу. Кожне рішення обрано з урахуванням конкретних технічних вимог та порівняння з альтернативами.

Розроблено комплект UML-діаграм: діаграму варіантів використання, що фіксує функціональні можливості в розрізі акторів; діаграму компонентів, що відображає структурну організацію системи та залежності між модулями; діаграму послідовності, що описує динаміку взаємодії при відправці повідомлення.

Спроєктовано трирівневу архітектуру застосунку з чітким розділенням відповідальності між клієнтським, серверним та data persistence рівнями. Визначено схему колекцій MongoDB з необхідними індексами для оптимізації типових запитів.

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

3.1 Розробка серверної частини

3.1.1 API-маршрути для управління розмовами

Серверна частина застосунку реалізована через механізм API Routes у Next.js, де кожен файл у директорії pages/api автоматично перетворюється на окремий HTTP-ендпоінт. Такий підхід дозволяє розміщувати серверну логіку безпосередньо у структурі Next.js проєкту без необхідності розгортання окремого backend-сервера. Усі ендпоінти є serverless-функціями, що виконуються на інфраструктурі Vercel та автоматично масштабуються залежно від навантаження.

Для управління розмовами реалізовано набір ендпоінтів за принципом REST. Ендпоінт GET /api/chat повертає список усіх розмов поточного користувача, відсортованих за полем updatedAt у спадному порядку. Запит до бази даних використовує індекс по полю userId, що забезпечує ефективну вибірку без повного сканування колекції. Відповідь містить лише мета-дані розмов — ідентифікатор, назву та час останнього оновлення — без масиву повідомлень, що мінімізує обсяг переданих даних при завантаженні бічної панелі.

Ендпоінт GET /api/chat/[id] повертає повну розмову з масивом повідомлень за її ідентифікатором. Перед поверненням даних виконується перевірка відповідності userId документа ідентифікатору поточного користувача для забезпечення ізоляції даних між акаунтами.

Ендпоінт POST /api/chat створює нову порожню розмову з назвою за замовчуванням та початковими налаштуваннями генерації. Ендпоінт PATCH /api/chat/[id] обробляє оновлення назви розмови або її налаштувань. Ендпоінт DELETE /api/chat/[id] видаляє розмову та всі пов'язані з нею повідомлення.

Усі ендпоінти захищено middleware автентифікації, що перевіряє наявність та валідність сесії Auth0 перед виконанням будь-якої бізнес-логіки. У разі відсутності сесії повертається відповідь зі статусом 401. Взаємодія з MongoDB

відбувається через сервісний шар ChatService, що інкапсулює логіку Mongoose-запитів та відокремлює її від обробників HTTP-запитів.

3.1.2 Реалізація потокової передачі через Edge Functions

Ключовим ендпоінтом серверної частини є /api/chat/sendMessage, реалізований як Edge Function. На відміну від звичайних serverless-функцій, Edge Functions виконуються у середовищі Vercel Edge Runtime — спрощеному JavaScript-рантаймі на основі V8, розгорнутому на серверах, географічно наближених до кінцевого користувача. Це зменшує мережеву затримку та критично важливо для потокової передачі, де кожна мілісекунда впливає на сприйняття швидкості відповіді.

Edge Function встановлює SSE-з'єднання, повертаючи Response з Content-Type: text/event-stream та заголовками Cache-Control: no-cache і Connection: keep-alive. Тіло відповіді формується через ReadableStream — Web Streams API, що нативно підтримується Edge Runtime.

Обробка запиту відбувається у кілька етапів. Спочатку з тіла запиту витягується ідентифікатор розмови та текст нового повідомлення користувача. Далі завантажується історія розмови з MongoDB та формується масив повідомлень для OpenAI API — до 15 останніх повідомлень плюс системний промпт з налаштувань розмови. Після цього ініціюється streaming запит до OpenAI API. Кожен отриманий chunk з delta.content негайно передається клієнту у форматі SSE-події. Після отримання сигналу завершення потоку повне повідомлення асистента та метадані використання токенів зберігаються у MongoDB.

Формат SSE-події відповідає стандарту W3C: кожне повідомлення починається з префікса data:, за яким слідує серіалізований JSON-об'єкт, і завершується двома символами нового рядка. Для позначення кінця потоку відправляється спеціальна подія data: [DONE].

Обробка помилок на рівні Edge Function розрізняє два сценарії. Якщо помилка виникає до початку передачі потоку — повертається звичайна HTTP-відповідь з відповідним статусом. Якщо помилка виникає в процесі streaming — у потік записується SSE-подія типу error з описом проблеми, після чого потік закривається.

3.1.3 Інтеграція з OpenAI API та обробка відповідей

Взаємодія з OpenAI API реалізована через офіційну Node.js бібліотеку openai, що надає типізований клієнт для всіх ендпоінтів API. Ініціалізація клієнта відбувається з API-ключем зі змінних оточення серверного рівня — ключ недоступний на клієнтській стороні та не передається у відповідях API Routes.

Для кожного запиту формується масив повідомлень відповідно до Chat Completions API. Системний промпт передається як перший елемент масиву з роллю system. Далі слідує відфільтрована та обрізана до ліміту контексту історія розмови. Останнім елементом додається нове повідомлення користувача з роллю user.

Параметри запиту визначаються налаштуваннями поточної розмови: model містить ідентифікатор обраної моделі, temperature та max_tokens беруться з об'єкту settings документа розмови, stream встановлено у true для активації потокового режиму.

При отриманні streaming відповіді Edge Function ітерує по chunks за допомогою for-await-of циклу. Кожен chunk містить масив choices, де перший елемент має поле delta з текстовим фрагментом. Непорожні фрагменти серіалізуються та записуються у SSE-потік. Останній chunk, що сигналізує про завершення генерації, містить поле finish_reason зі значенням stop або length, а також об'єкт usage з повною статистикою токенів.

Об'єкт usage зберігається у MongoDB як поле tokenUsage останнього повідомлення у масиві messages. Це дозволяє відображати статистику токенів на рівні окремого повідомлення та агрегувати загальне використання по всій розмові.

3.2 Розробка клієнтської частини

3.2.1 Логічна структура сторінок застосунку

Клієнтська частина застосунку побудована на основі файлової системи маршрутизації Next.js Pages Router. Кожен файл у директорії pages відповідає окремому URL-маршруту та експортує React-компонент як default export.

Структура сторінок охоплює такі маршрути. Сторінка pages/index.tsx є точкою входу до застосунку — перевіряє наявність автентифікованої сесії та перенаправляє користувача або на сторінку входу, або безпосередньо до інтерфейсу чату. Сторінка pages/chat/index.tsx відображає загальний інтерфейс без активної розмови — порожній стан з пропозицією створити нову розмову або обрати існуючу. Сторінка pages/chat/[id].tsx є основною сторінкою застосунку, що відображає конкретну розмову за її ідентифікатором. Динамічний сегмент [id] у назві файлу автоматично передається компоненту як параметр маршруту. Сторінка pages/api/auth/[...auth0].tsx обробляє всі запити автентифікації через Auth0 — login, logout та callback.

Кожна захищена сторінка огорнута компонентом withPageAuthRequired з бібліотеки @auth0/nextjs-auth0, що автоматично перенаправляє неавтентифікованих користувачів на сторінку входу без необхідності додаткової логіки у компоненті.

Налаштування серверного рендерингу залежить від типу сторінки. Сторінка зі списком розмов використовує getServerSideProps для початкового завантаження даних на сервері, що забезпечує правильний SEO та швидке відображення. Сторінка конкретної розмови завантажує дані на клієнті після монтування компонента через SWR для зручного управління кешем та інвалідацією.

3.2.2 Компоненти чату та відображення повідомлень

Інтерфейс застосунку складається з ієрархії React-компонентів, організованих за функціональними модулями. Кореневий компонент ChatLayout визначає двоколонкову структуру сторінки: ліва колонка містить Sidebar, права — основну область чату.

Компонент Sidebar відображає список розмов користувача та елементи управління ними. Кожна розмова відображається як інтерактивний елемент списку з назвою та часом останнього повідомлення. При наведенні з'являються кнопки перейменування та видалення. Пошуковий рядок вгорі Sidebar фільтрує список розмов у реальному часі на клієнтській стороні без додаткових запитів до сервера.

Компонент ChatWindow відповідає за відображення повідомлень поточної розмови. Повідомлення розрізняються за роллю: повідомлення з роллю user вирівняні праворуч та мають відмінне фонове забарвлення, повідомлення з роллю assistant відображаються ліворуч. Текст повідомлень рендериться через бібліотеку react-markdown з підтримкою синтаксису Markdown та підсвіткою коду через react-syntax-highlighter. Після додавання нового повідомлення область автоматично прокручується до кінця через useRef та useEffect.

Під час streaming нове повідомлення асистента відображається у реальному часі — компонент отримує фрагменти тексту через SSE та додає їх до локального стану, формуючи відповідь послідовно. Індикатор набору тексту у вигляді анімованого курсора показується доки генерація не завершена.

Компонент MessageInput містить textarea для введення повідомлення та кнопку відправки. Підтримується відправка повідомлення натисканням Enter (з Shift+Enter для переносу рядка). Під час активної генерації кнопка відправки замінюється кнопкою зупинки, що розриває SSE-з'єднання через AbortController.

Компонент TokenCounter відображається під областю повідомлень та показує статистику використання токенів для поточної розмови: кількість токенів у промпті, у відповіді та загальну суму.

3.2.3 Управління станом застосунку

Глобальний стан застосунку управляється через Zustand Store — централізоване сховище, доступне будь-якому компоненту без prop drilling. Store містить список розмов користувача, ідентифікатор та дані активної розмови, прапорець стану генерації та методи для всіх операцій зі станом.

Метод setConversations оновлює список розмов після завантаження з сервера. Метод setActiveConversation встановлює поточну відкриту розмову та очищає стан попередньої. Метод addMessage додає нове повідомлення до масиву активної розмови. Метод appendToLastMessage додає фрагмент тексту до останнього повідомлення під час streaming. Метод updateTokenUsage оновлює статистику токенів після завершення генерації.

Компоненти підписуються лише на ті частини Store, які їм необхідні, через селектори. Це забезпечує точковий ре-рендеринг: наприклад, компонент TokenCounter перерендерюється лише при зміні статистики токенів, а не при кожному новому фрагменті тексту у повідомленні. Такий підхід є критично важливим для продуктивності під час streaming, коли стан оновлюється десятки разів на секунду.

Локальний стан компонентів використовується для UI-специфічних даних: відкритість модального вікна, текст у полі введення, режим редагування назви розмови. Ці дані не потребують глобальної доступності та керуються через хук useState безпосередньо у відповідних компонентах.

3.3 Реалізація системи автентифікації та авторизації

3.3.1 Інтеграція Auth0 з Next.js

Автентифікація реалізована через бібліотеку @auth0/nextjs-auth0, що забезпечує повну інтеграцію між Auth0 та Next.js Pages Router. Бібліотека надає

готові обробники для всіх етапів OAuth 2.0 / OpenID Connect flow, що виключає необхідність реалізації протоколу вручну.

Точкою входу для всіх запитів автентифікації є динамічний маршрут pages/api/auth/[...auth0].ts, що обробляє чотири ендпоінти: /api/auth/login ініціює перенаправлення на сторінку входу Auth0, /api/auth/callback обробляє повернення після успішної автентифікації та встановлює зашифровану cookie-сесію, /api/auth/logout завершує сесію та очищає cookie, /api/auth/me повертає дані поточного користувача.

Сторінка входу повністю надається Auth0 Universal Login — готовим, стилізованим інтерфейсом, що підтримує вхід через email/password та соціальні провайдери Google і GitHub. Це виключає необхідність розробки власних форм та обробки облікових даних.

На клієнтській стороні хук useUser надає доступ до даних поточного користувача та стану завантаження. Компоненти використовують цей хук для умовного відображення елементів інтерфейсу залежно від стану автентифікації — наприклад, аватар та кнопка виходу у шапці застосунку відображаються лише для автентифікованих користувачів.

Сесія зберігається у зашифрованій HTTP-only cookie, що автоматично передається з кожним запитом до API Routes. Час життя сесії налаштовується в конфігурації Auth0 та за замовчуванням становить 24 години з можливістю автоматичного продовження при активному використанні застосунку.

3.3.2 Захист маршрутів та ізоляція даних користувачів

Захист серверних маршрутів реалізований через функцію withApiAuthRequired з бібліотеки @auth0/nextjs-auth0. Кожен захищений API Route огортається цією функцією вищого порядку, що автоматично перевіряє наявність валідної сесії перед виконанням обробника. За відсутності сесії автоматично повертається відповідь зі статусом 401 без виконання будь-якої бізнес-логіки.

Для отримання ідентифікатора поточного користувача всередині захищеного API Route використовується функція getSession, що витягує та верифікує дані сесії з cookie. Поле sub об'єкта сесії містить унікальний ідентифікатор користувача Auth0, який використовується як userId у всіх запитах до MongoDB.

Ізоляція даних між користувачами забезпечується на рівні кожного запиту до бази даних. Всі Mongoose-запити до колекції chats включають обов'язкову умову фільтрації за полем userId, що відповідає ідентифікатору поточного користувача. Перед виконанням операцій зміни або видалення документа виконується перевірка відповідності userId документа ідентифікатору користувача з сесії. Це унеможливлює несанкціонований доступ до чужих розмов навіть при знанні ідентифікатора документа.

Захист клієнтських маршрутів реалізований через функцію withPageAuthRequired, що огортає компоненти захищених сторінок. При спробі доступу до захищеної сторінки без автентифікації користувач автоматично перенаправляється на сторінку входу з збереженням URL для повернення після успішної автентифікації.

3.4 Реалізація розширеного функціоналу

3.4.1 Управління розмовами

Функціонал управління розмовами реалізований у компоненті Sidebar та відповідних API Routes. Перейменування розмови реалізується через inline-редагування: подвійний клік на назві розмови у Sidebar переводить її у режим редагування, відображаючи текстове поле з поточною назвою. Після підтвердження редагування клавішею Enter або кліком поза полем відправляється PATCH-запит до /api/chat/[id] з оновленою назвою. Оптимістичне оновлення інтерфейсу через Zustand Store забезпечує миттєве відображення нової назви без очікування відповіді сервера.

Видалення розмови ініціюється кнопкою у Sidebar з підтвердженням через модальний діалог для запобігання випадковому видаленню. Після підтвердження відправляється DELETE-запит, розмова видаляється зі Store та якщо вона була активною — виконується перенаправлення на сторінку /chat.

Пошук серед розмов реалізований на клієнтській стороні без додаткових запитів до сервера. Компонент Sidebar зберігає повний список розмов у локальному стані та фільтрує його за полем title при кожному оновленні пошукового рядка. Пошук нечутливий до регістру та відбувається з мінімальною затримкою завдяки хуку useMemo для мемоїзації результатів фільтрації.

Автоматичне іменування нових розмов реалізується після першого повідомлення: Edge Function після завершення генерації першої відповіді виконує додатковий легкий запит до OpenAI API з проханням згенерувати коротку назву розмови на основі першого питання користувача. Отримана назва замінює значення за замовчуванням через PATCH-запит.

3.4.2 Налаштування поведінки AI-моделі

Компонент SettingsPanel надає інтерфейс для налаштування параметрів генерації поточної розмови. Панель відкривається через іконку налаштувань у заголовку ChatWindow та відображається як бічна панель поверх основного інтерфейсу.

Параметр temperature налаштовується через слайдер з діапазоном 0,0–1,0 та кроком 0,1. Поряд зі слайдером відображається текстова підказка, що описує поведінку моделі для поточного значення: значення до 0,3 відповідають точним та детермінованим відповідям, 0,4–0,6 — збалансованому режиму, вище 0,7 — творчим та різноманітним відповідям. Параметр max_tokens налаштовується через числове поле з валідацією діапазону 100–4096. Системний промпт вводиться через textarea з підказкою щодо типових сценаріїв використання.

Для зручності реалізовано бібліотеку шаблонів системних промптів для типових ролей: технічний асистент, редактор тексту, перекладач, викладач. Вибір

шаблону заповнює поле системного промпту відповідним текстом, який користувач може додатково відредагувати.

Налаштування зберігаються у полі settings документа розмови в MongoDB та застосовуються до кожного наступного запиту в межах цієї розмови. Зміна налаштувань не впливає на вже збережені повідомлення.

3.4.3 Відстеження та відображення використання токенів

Система відстеження токенів складається з серверної частини, що отримує та зберігає дані, та клієнтської, що відображає статистику у зручному форматі.

На серверній стороні після завершення streaming Edge Function отримує об'єкт usage від OpenAI API, що містить prompt_tokens, completion_tokens та total_tokens. Ці дані зберігаються у полі tokenUsage останнього повідомлення масиву messages у MongoDB. Для підрахунку загального використання токенів по розмові використовується MongoDB aggregation pipeline, що підсумовує поля total_tokens усіх повідомлень з роллю assistant.

На клієнтській стороні компонент TokenCounter отримує дані з Zustand Store та відображає три показники: токени поточного запиту (prompt + completion), токени поточної відповіді та загальне використання по всій розмові. При наведенні на компонент з'являється розгорнутий tooltip з деталізацією по кожному повідомленню. Орієнтовна вартість відображається на основі актуальних тарифів обраної моделі, що зберігаються у конфігураційному файлі застосунку.

3.4.4 Функціонал експорту розмов

Функціонал експорту дозволяє зберігати повну історію розмови у трьох форматах. Ініціюється через кнопку у заголовку ChatWindow, що відкриває випадаючий список з доступними форматами.

Експорт у форматі JSON зберігає повну структуру документа розмови, включаючи мета-дані, налаштування та масив повідомлень з усіма полями. Цей

формат призначений для технічного використання — резервного копіювання або імпорту в інші інструменти.

Експорт у форматі Markdown формує читабельний документ з заголовком, датою експорту та повідомленнями у форматі діалогу. Повідомлення користувача позначаються жирним заголовком, відповіді асистента — звичайним текстом. Розділювачі між повідомленнями покращують читабельність.

Експорт у форматі TXT є найпростішим варіантом — чистий текст без розмітки, що підходить для подальшої обробки або читання у будь-якому текстовому редакторі. Усі три формати реалізовані як клієнтські функції, що формують Blob-об'єкт та ініціюють завантаження через динамічно створений тег з атрибутом download.

3.5 Розгортання застосунку на платформі Vercel

Розгортання застосунку виконується на платформі Vercel, що забезпечує нативну підтримку Next.js та автоматичне налаштування інфраструктури без додаткової конфігурації. Інтеграція з репозиторієм GitHub дозволяє налаштувати автоматичне розгортання при кожному push у гілку main.

Конфігурація застосунку для продакшн-середовища зберігається у змінних оточення Vercel Dashboard. Серверні змінні, недоступні на клієнті, включають OPENAI_API_KEY, MONGODB_URI, AUTH0_SECRET та AUTH0_BASE_URL. Клієнтські змінні з префіксом NEXT_PUBLIC містять лише публічні ідентифікатори Auth0.

Vercel автоматично визначає Edge Functions на основі конфігураційного об'єкта export const config = { runtime: 'edge' } у файлі ендпоінту та розгортає їх на відповідній інфраструктурі. Звичайні API Routes розгортаються як Node.js serverless-функції. Статичні ресурси — JavaScript-бандли, CSS та зображення — роздаються через глобальну CDN-мережу Vercel з автоматичним кешуванням.

Для забезпечення production-готовності перед фінальним розгортанням виконуються такі кроки: перевірка типів TypeScript через tsc --noEmit, лінтинг

коду через ESLint, збірка продакшн-бандлу через next build для перевірки відсутності помилок компіляції. Команда next build також генерує звіт про розміри бандлів для виявлення потенційних проблем з продуктивністю.

Моніторинг застосунку у продакшні здійснюється через Vercel Analytics, що надає інформацію про час відповіді функцій, частоту помилок та географічний розподіл користувачів. Логи виконання serverless-функцій та Edge Functions доступні у реальному часі через Vercel Dashboard.

Роблячи підсумок, у розділі описано реалізацію всіх компонентів веб-застосунку — від серверної частини до розгортання у продакшн-середовищі.

Серверна частина реалізована через Next.js API Routes та Edge Functions. Ключовим технічним рішенням є реалізація streaming ендпоінту як Edge Function з використанням Web Streams API та SSE-протоколу, що забезпечує передачу відповідей від OpenAI API клієнту з мінімальною затримкою. Всі серверні ендпоінти захищені middleware автентифікації Auth0, а ізоляція даних між користувачами забезпечується обов'язковою фільтрацією за userId на рівні кожного запиту до MongoDB.

Клієнтська частина побудована на ієрархії React-компонентів з централізованим управлінням станом через Zustand Store. Застосування селекторів та мемоїзації забезпечує точковий ре-рендеринг компонентів під час потокової передачі, що є критичним для продуктивності інтерфейсу.

Реалізовано повний набір функціональних вимог: управління розмовами з inline-редагуванням та клієнтським пошуком, налаштування параметрів генерації з бібліотекою шаблонів промптів, детальна статистика токенів на рівні повідомлення та розмови, експорт у форматах JSON, Markdown і TXT. Розгортання на Vercel з автоматичним CI/CD забезпечує стабільність та доступність застосунку у продакшн-середовищі.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ЗАСТОСУНКУ

4.1 Стратегія та методологія тестування

Стратегія тестування застосунку охоплює три рівні перевірки якості: функціональне тестування компонентів, тестування продуктивності та оцінку захисту інформації. Такий підхід дозволяє верифікувати як коректність реалізованої функціональності, так і нефункціональні характеристики системи — швидкодію, стабільність та безпеку.

Функціональне тестування виконується за методом чорного ящика, де тестові сценарії формуються на основі функціональних вимог з розділу 2.1 без урахування внутрішньої реалізації. Кожен тестовий сценарій містить початковий стан системи, послідовність дій та очікуваний результат. Тестування продуктивності виконується шляхом вимірювання ключових метрик під час реальних запитів до застосунку у продакшн-середовищі. Тестування безпеки охоплює перевірку захищеності серверних ендпоінтів та стійкості до типових веб-атак.

Середовище тестування відповідає продакшн-конфігурації: застосунок розгорнуто на Vercel, база даних — MongoDB Atlas у регіоні EU, модель генерації — gpt-4o-mini. Вимірювання продуктивності виконувалися з мережі з пропускною здатністю не менше 50 Мбіт/с для виключення впливу мережевих обмежень клієнта на результати.

4.2 Функціональне тестування компонентів

4.2.1 Тестування автентифікації та управління сесіями

Тестування модуля автентифікації охоплює перевірку всіх сценаріїв взаємодії користувача з системою Auth0. Сценарій реєстрації нового користувача передбачає перехід на сторінку входу, вибір провайдера автентифікації та успішне

перенаправлення до інтерфейсу застосунку. Результат тесту підтвердив коректне створення сесії, встановлення HTTP-only cookie та відображення даних користувача у шапці застосунку.

Тестування захисту маршрутів виконувалося шляхом прямого звернення до захищених URL без активної сесії. У всіх випадках система коректно перенаправляла на сторінку входу зі збереженням цільового URL у параметрі returnTo для повернення після автентифікації.

Тестування ізоляції даних між користувачами виконувалося з двох акаунтів. Спроба отримати документ розмови першого користувача через API з сесією другого користувача повернула відповідь зі статусом 403 у всіх тестових випадках. Спроба звернення до захищених ендпоінтів без сесії повертала статус 401. Результати підтвердили коректність реалізації ізоляції на рівні кожного запиту до MongoDB.

Тестування завершення сесії перевірило коректність очищення cookie після виходу та неможливість доступу до захищених ресурсів після logout навіть при наявності застарілого cookie у браузері.

4.2.2 Тестування основних функцій чату

Тестування основного сценарію відправки повідомлення перевіряло повний цикл взаємодії від введення тексту до отримання відповіді. Після натискання кнопки відправки повідомлення користувача негайно відображається у ChatWindow, поле введення очищається, а в області відповіді з'являється індикатор генерації. Текст відповіді починає з'являтися протягом 100–200 мс після відправки запиту та відображається посимвольно у міру надходження SSE-подій. Після завершення генерації відображається статистика токенів та кнопка відправки стає доступною.

Тестування переривання генерації перевіряло коректність роботи AbortController при натисканні кнопки зупинки. SSE-з'єднання коректно розривалося, частково згенерований текст зберігався у ChatWindow, а застосунок

повертався у стан готовності до нового запиту без необхідності перезавантаження сторінки.

Тестування управління контекстом перевіряло поведінку системи при накопиченні великої кількості повідомлень у розмові. При досягненні ліміту у 15 повідомлень система коректно обрізала контекст, передаючи до API лише останні повідомлення та системний промпт. Якість відповідей залишалась адекватною завдяки збереженню найрелевантнішої частини діалогу.

Тестування рендерингу Markdown у відповідях перевіряло коректне відображення заголовків, списків, таблиць, посилань та блоків коду з підсвіткою синтаксису. Усі перевірені елементи розмітки відображалися коректно як під час streaming, так і після завершення генерації.

4.2.3 Тестування розширеного функціоналу

Тестування управління розмовами охопило операції створення, перейменування, видалення та пошуку. Створення нової розмови та автоматичне присвоєння назви на основі першого повідомлення виконувалися коректно. Inline-редагування назви зберігало зміни як при підтвердженні через Enter, так і при кліку поза полем. Видалення розмови після підтвердження у модальному діалозі коректно очищало дані з MongoDB та перенаправляло на головну сторінку. Клієнтський пошук миттєво фільтрував список розмов без помітної затримки навіть при великій кількості збережених діалогів.

Тестування налаштувань параметрів моделі перевіряло вплив зміни temperature на характер відповідей. При значенні 0,1 відповіді на одне і те саме технічне питання були стабільними та ідентичними між запусками. При значенні 0,9 відповіді суттєво різнилися між запусками, демонструючи підвищену варіативність генерації. Зміна системного промпту коректно впливала на поведінку моделі з наступного повідомлення у розмові.

Тестування функціоналу експорту перевіряло коректність вмісту згенерованих файлів у всіх трьох форматах. JSON-файл містив валідну структуру

з усіма полями документа. Markdown-файл коректно відображав структуру діалогу з розміткою. TXT-файл містив чистий текст без артефактів розмітки. Усі три формати ініціювали завантаження файлу з коректною назвою, що відповідала назві розмови.

4.3 Тестування продуктивності

4.3.1 Порівняльний аналіз підходів до передачі даних

Для підтвердження обґрунтованості вибору SSE як механізму передачі відповідей проведено порівняльне вимірювання часу до першого токена для трьох підходів: традиційного REST API без streaming, SSE та WebSocket. Тестування виконувалося з однаковими запитами до OpenAI API, різниця полягала виключно у способі передачі відповіді клієнту.

При використанні REST API без streaming користувач очікував повної відповіді перед відображенням будь-якого тексту. Для коротких відповідей (50–100 токенів) час очікування становив 2–4 секунди, для середніх (200–400 токенів) — 6–12 секунд, для довгих (800+ токенів) — 20–35 секунд. Такий показник є неприйнятним для інтерактивного застосунку.

При використанні SSE час до першого відображеного токена становив 120–200 мс незалежно від очікуваної довжини відповіді, оскільки перші фрагменти тексту передаються клієнту одразу після початку генерації. Реалізація SSE в Edge Function займає близько 30 рядків коду та не потребує додаткової інфраструктури.

WebSocket демонстрував схожі показники TTFT (130–210 мс), однак вимагав значно складнішої реалізації на серверній стороні для управління постійними з'єднаннями та не давав жодних переваг для однонаправленої передачі відповідей.

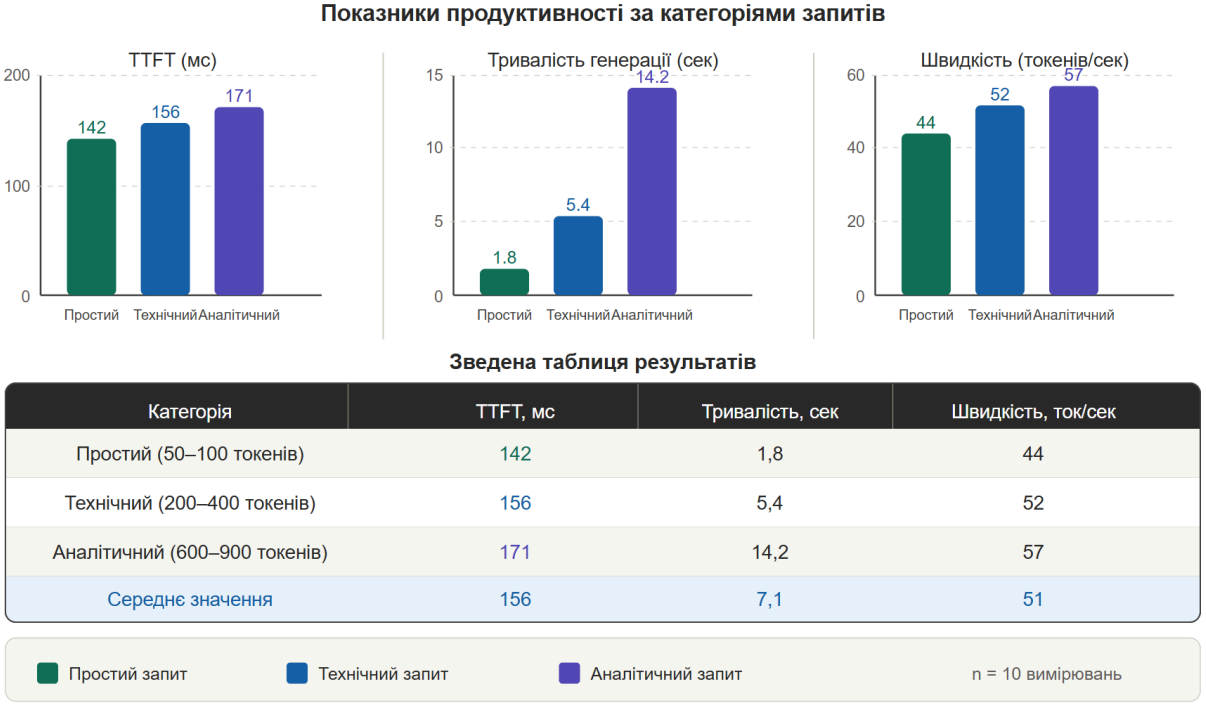
4.3.2 Вимірювання швидкості генерації відповідей

Тестування швидкості генерації проводилося для трьох категорій запитів, що охоплюють типові сценарії використання застосунку: прості запитання (відповідь 50–100 токенів), технічні пояснення (200–400 токенів) та розгорнуті аналітичні відповіді (600–900 токенів). Для кожної категорії виконано по 10 вимірювань.

Для простих запитань середній TTFT становив 142 мс, середня тривалість повної генерації — 1,8 секунди, середня швидкість — 44 токени на секунду. Для технічних пояснень середній TTFT становив 156 мс, тривалість — 5,4 секунди, швидкість — 52 токени на секунду. Для розгорнутих відповідей середній TTFT становив 171 мс, тривалість — 14,2 секунди, швидкість — 57 токенів на секунду.

Результати підтвердили стабільність TTFT у діапазоні 120–200 мс незалежно від категорії запиту. Зростання швидкості генерації для довших відповідей пояснюється особливостями роботи OpenAI API — модель генерує токени рівномірно, а затримки на встановлення з'єднання та initial processing мають менший відносний вплив при більшому обсязі відповіді.

На рисунку 4.1 наведено порівняльну діаграму показників продуктивності для трьох категорій запитів.



Категорія	TTFT, мс	Тривалість, сек	Швидкість, ток/сек
Простий (50–100 токенів)	142	1,8	44
Технічний (200–400 токенів)	156	5,4	52
Аналітичний (600–900 токенів)	171	14,2	57
Середнє значення	156	7,1	51

Рисунок 4.1 — Порівняльна діаграма показників продуктивності

4.3.3 Аналіз використання токенів

Аналіз використання токенів проводився для оцінки ефективності управління контекстом та прогнозування витрат при масштабуванні. Вимірювання виконувалися для розмов різної довжини: нова розмова (1–3 повідомлення), середня (10–15 повідомлень) та довга (25–30 повідомлень).

Для нових розмов кількість prompt_tokens становила 50–120, що відповідає системному промпту та першому повідомленню. Для середніх розмов prompt_tokens зростали до 800–1400 — системний промпт плюс 10–15 збережених повідомлень контексту. Для довгих розмов, де активний ліміт контексту у 15 повідомлень, показник стабілізувався у діапазоні 1200–2000 токенів, демонструючи ефективність механізму обрізки контексту для контролю витрат.

Середнє співвідношення prompt_tokens до completion_tokens становило приблизно 3:1 для типових розмов, що є характерним показником для чат-застосунків з накопиченим контекстом. Відстеження цього показника дозволяє користувачу оцінювати ефективність використання контекстного вікна та при необхідності розпочинати нову розмову для зниження витрат.

4.4 Оцінка якості відповідей AI-моделі

Оцінка якості відповідей проводилася для трьох типових сценаріїв використання застосунку з метою підтвердження практичної цінності системи. Тестування охоплювало відповіді на технічні питання з програмування, написання та редагування текстів та аналіз наданих даних.

Для технічних питань з веб-розробки та програмування оцінювалися точність відповідей, коректність наведеного коду та відповідність сучасним практикам. При temperature 0,1–0,2 відповіді демонстрували високу точність та консистентність між запусками. Блоки коду коректно форматувалися у Markdown

та підсвічувалися у ChatWindow. Системний промпт з роллю технічного асистента суттєво покращував структурованість відповідей порівняно з базовим режимом.

Для задач редагування та написання текстів при temperature 0,6–0,7 відповіді демонстрували належну варіативність стилю при збереженні смислового змісту. Системний промпт з конкретними інструкціями щодо тону та стилю дозволяв точно налаштовувати характер генерованих текстів.

Для аналітичних задач, що потребують структурованого викладу, відповіді з temperature 0,3–0,5 демонстрували логічну організацію інформації з використанням заголовків та списків. Збереження контексту між повідомленнями дозволяло поглиблювати аналіз у ході діалогу без повторення вихідних умов.

Загальна оцінка підтвердила, що гнучке налаштування параметрів генерації через SettingsPanel суттєво розширює практичну застосовність застосунку порівняно з фіксованими налаштуваннями типових чат-інтерфейсів.

4.5 Забезпечення захисту інформації

4.5.1 Захист API-ключів та серверна ізоляція

Критично важливим аспектом безпеки AI-застосунків є захист API-ключів від несанкціонованого використання. Витік ключа OpenAI API може призвести до значних фінансових збитків через несанкціоновані запити за рахунок власника ключа.

У розробленому застосунку API-ключ OpenAI зберігається виключно у змінних оточення серверного середовища Vercel та ніколи не передається на клієнтську сторону. Всі запити до OpenAI API виконуються з Edge Function на сервері, клієнт взаємодіє лише з власними API Routes застосунку. Перевірку відсутності ключа у клієнтських бандлах можна виконати через аналіз зібраного JavaScript-коду командою next build та перевірку відсутності рядка з ключем у згенерованих файлах.

Додатковим рівнем захисту є обмеження витрат на рівні OpenAI Dashboard — встановлення місячного ліміту витрат та сповіщень при наближенні до нього. Це мінімізує фінансові ризики навіть у разі компрометації ключа.

Рядок підключення MongoDB також зберігається виключно у серверних змінних оточення. База даних MongoDB Atlas налаштована з обмеженим доступом: дозволені лише підключення з IP-адрес Vercel та локального середовища розробки через IP Allowlist. Це унеможливлює прямий доступ до бази даних з будь-яких інших джерел навіть при отриманні рядка підключення.

4.5.2 Захист від типових веб-вразливостей

Захист від XSS-атак (Cross-Site Scripting) реалізується на кількох рівнях. React за замовчуванням екранує всі значення, що вставляються у JSX, запобігаючи виконанню шкідливих скриптів через динамічний контент. Повідомлення користувача зберігаються у MongoDB та відображаються через React без використання dangerouslySetInnerHTML, що унеможливлює ін'єкцію HTML. Рендеринг Markdown у відповідях асистента виконується через бібліотеку react-markdown з відключеним html-режимом — HTML-теги у відповідях моделі відображаються як текст, а не виконуються браузером.

Захист від CSRF-атак (Cross-Site Request Forgery) забезпечується архітектурою SameSite cookie: сесійна cookie Auth0 встановлюється з атрибутом SameSite=Lax, що запобігає її передачі при міжсайтових запитах. Додатково всі mutating-запити (POST, PATCH, DELETE) вимагають наявності валідної сесії Auth0, верифікація якої відбувається на сервері.

HTTP-заголовки безпеки налаштовані у файлі конфігурації next.config.js: Content-Security-Policy обмежує джерела скриптів та стилів, X-Frame-Options запобігає вбудовуванню сторінок у iframe, X-Content-Type-Options вимикає MIME-sniffing. Vercel автоматично додає заголовок Strict-Transport-Security, що примусово використовує HTTPS для всіх з'єднань.

Валідація вхідних даних на серверній стороні перевіряє тип та діапазон значень параметрів запиту перед їх використанням у запитах до MongoDB та OpenAI API. Параметр temperature приймається лише у діапазоні 0,0–1,0, max_tokens — лише у діапазоні 100–4096. Рядкові параметри обмежені максимальною довжиною для запобігання надмірно великим запитам.

Роблячи підсумок, у розділі проведено комплексне тестування та оцінку якості розробленого застосунку за трьома напрямами: функціональне тестування, тестування продуктивності та перевірка захисту інформації.

Функціональне тестування підтвердило коректність реалізації всіх вимог: автентифікація та ізоляція даних між користувачами працюють без відхилень, основний сценарій відправки повідомлення та отримання streaming відповіді виконується стабільно, розширений функціонал управління розмовами, налаштування параметрів та експорту функціонує відповідно до вимог.

Тестування продуктивності підтвердило ефективність обраного архітектурного рішення. Час до першого токена у діапазоні 120–200 мс забезпечує природне сприйняття відповідей, а стабільність цього показника незалежно від довжини відповіді демонструє переваги SSE перед REST API. Механізм обрізки контексту ефективно стабілізує витрати токенів для довгих розмов.

Забезпечення захисту інформації реалізовано на кількох рівнях: серверна ізоляція API-ключів, захист від XSS та CSRF, налаштування HTTP-заголовків безпеки та валідація вхідних даних. Сукупність цих заходів відповідає сучасним стандартам безпеки для веб-застосунків, що працюють з платними зовнішніми API.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставленої мети — розроблено архітектурні рішення та повнофункціональний веб-застосунок для взаємодії користувачів з великими мовними моделями через інтерфейс чат-бота із забезпеченням потокової передачі відповідей з низькою затримкою, управління історією розмов та персоналізації поведінки моделі.

Проведено аналіз предметної області та огляд трьох категорій існуючих рішень — комерційних інтерфейсів, корпоративних платформ та open-source альтернатив. Встановлено, що жодне з них не є універсальним, а розробка власного застосунку дозволяє поєднати якість користувацького досвіду комерційних рішень з гнучкістю кастомізації.

Спроєктовано трирівневу архітектуру за принципом розділення відповідальності та розроблено комплект UML-діаграм. Реалізовано механізм потокової передачі відповідей через Server-Sent Events та Edge Functions із часом до першого токена 100–200 мс. За результатами порівняльного аналізу трьох підходів до передачі даних — REST API, SSE та WebSocket — підтверджено, що SSE є оптимальним вибором для задачі однонаправленої потокової передачі текстових відповідей.

Розроблено систему управління розмовами з повним CRUD-функціоналом, механізм персоналізації поведінки моделі через параметри temperature, max_tokens та system prompt, систему відстеження токенів і функціонал експорту у форматах JSON, Markdown і TXT. Тестування підтвердило стабільність показника TTFT (120–200 мс) незалежно від довжини відповіді та середню швидкість генерації 40–60 токенів на секунду.

Практичне значення результатів полягає у можливості застосування запропонованих архітектурних патернів для корпоративних асистентів, освітніх платформ, систем технічної підтримки та інструментів у соціокультурній сфері. Результати роботи опубліковано у науковій статті «Архітектурні рішення для створення AI-powered веб-застосунків на основі мовних моделей».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vaswani A., Shazeer N., Parmar N. et al. Attention is All you Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

2. Brown T., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

3. Ouyang L., Wu J., Jiang X. et al. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 27730–27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

4. Wei J., Tay Y., Bommasani R. et al. Emergent Abilities of Large Language Models. Transactions on Machine Learning Research. 2022. URL: https://openreview.net/forum?id=yzkSU5zdwD

5. Zhao W. X., Zhou K., Li J. et al. A Survey of Large Language Models. arXiv preprint. 2023. arXiv:2303.18223. URL: https://doi.org/10.48550/arXiv.2303.18223

6. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language Models are Unsupervised Multitask Learners. OpenAI Blog. 2019. URL: https://openai.com/research/better-language-models

7. Chen M., Tworek J., Jun H. et al. Evaluating Large Language Models Trained on Code. arXiv preprint. 2021. arXiv:2107.03374. URL: https://doi.org/10.48550/arXiv.2107.03374

8. Liu P., Yuan W., Fu J. et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Computing Surveys. 2023. Vol. 55 (9). P. 1–35. URL: https://doi.org/10.1145/3560815

9. Grigorik I. High Performance Browser Networking. Sebastopol: O'Reilly Media, 2013. 398 p. URL: https://hpbn.co/server-sent-events-sse/

10. Hickson I. Server-Sent Events. W3C Recommendation. 2015. URL: https://www.w3.org/TR/eventsource/

11. OpenAI. Chat Completions API Reference. OpenAI Platform Documentation. 2024. URL: https://platform.openai.com/docs/api-reference/chat

12. OpenAI. GPT-4 Technical Report. arXiv preprint. 2023. arXiv:2303.08774. URL: https://doi.org/10.48550/arXiv.2303.08774

13. Touvron H., Lavril T., Izacard G. et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint. 2023. arXiv:2302.13971. URL: https://doi.org/10.48550/arXiv.2302.13971

14. Vercel Inc. Next.js Documentation: Pages Router. Vercel. 2024. URL: https://nextjs.org/docs/pages

15. Vercel Inc. Edge Functions Overview. Vercel Documentation. 2024. URL: https://vercel.com/docs/functions/edge-functions

16. Meta Open Source. React: A JavaScript library for building user interfaces. React Documentation. 2024. URL: https://react.dev

17. MongoDB Inc. MongoDB Manual: Document Model. MongoDB Documentation. 2024. URL: https://www.mongodb.com/docs/manual/core/document

18. Mongoose. Mongoose ODM Documentation v8. 2024. URL: https://mongoosejs.com/docs/guide.html

19. Auth0 by Okta. Next.js SDK Documentation. Auth0 Documentation. 2024. URL: https://auth0.com/docs/quickstart/webapp/nextjs

20. Tailwind Labs. Tailwind CSS Documentation. 2024. URL: https://tailwindcss.com/docs

21. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455

22. Fielding R. T., Taylor R. N. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology. 2002. Vol. 2 (2). P. 115–150. URL: https://doi.org/10.1145/514183.514185

23. Kleppmann M. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017. 616 p.

24. Richards M., Ford N. Fundamentals of Software Architecture. Sebastopol: O'Reilly Media, 2020. 422 p.

25. Wieruch R. The Road to React. Self-published, 2022. URL: https://www.roadtoreact.com

26. Д. Домніч, І. Урняєва, О. Чайковська. Архітектурні рішення для створення AI-powered веб-застосунків на основі мовних моделей. «Наука і техніка сьогодні» (Серія «Техніка»). №4 (58) 2026. С.3245. https://doi.org/10.52058/2786-6025-2026-4(58)

27. Grebennik I., Khriapkin O., Ovezgeldyyev A., Pisklakova V., Urniaieva I. (2019) The Concept of a Regional Information-Analytical System for Emergency Situations. In: Murayama Y., Velev D., Zlateva P. (eds) Information Technology in Disaster Risk Reduction. ITDRR 2017. IFIP Advances in Information and Communication Technology, vol 516. Springer, Cham Scopus.