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 системи підтримки супроводу клієнтів мовної школи з використанням AI-асистента»

Виконав студент гр. КН-22-2 _____ Табалов М. А.

Керівник _____ Харламенко В. Ю.

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.*

_____

« *26* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-2 *Табалову Максиму Андрійовичу*

1. **Тема кваліфікаційної роботи:** *«Розробка системи підтримки супроводу клієнтів мовної школи з використанням AI-асистента»*

затверджено наказом по університету № *173с* від *30.03.2026 р.*

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

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 80 с., презентація у Microsoft PowerPoint (12 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Харламенко В. Ю.

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: *28.01.2026 р.*

Керівник _____ / Харламенко В. Ю./

7. Запевнення: *Я, Табалов Максим Андрійович, 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.

Студент _____ / Табалов М. А. /

АНОТАЦІЯ

Табалов М. А. Розробка системи підтримки супроводу клієнтів мовної школи з використанням AI-асистента : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 70с.

У кваліфікаційній роботі розглянуто задачу розробки вебсистеми підтримки супроводу клієнтів мовної школи з використанням AI-асистента. Актуальність теми зумовлена потребою мовних шкіл у швидкій та якісній обробці звернень клієнтів, зменшенні навантаження на менеджерів, централізованому збереженні історії комунікації та автоматизації відповідей на типові запитання щодо курсів, вартості, формату навчання й запису на заняття.

Метою роботи є проєктування та програмна реалізація вебсистеми, яка забезпечує взаємодію клієнта з AI-асистентом, збереження історії діалогів, розпізнавання намірів користувача, ескалацію складних звернень до адміністратора та керування базою знань мовної школи через адміністративну панель.

У першому розділі проаналізовано предметну галузь, визначено основні проблеми клієнтського супроводу в мовних школах, розглянуто існуючі підходи до автоматизації комунікації та обґрунтовано доцільність створення власної вебсистеми з AI-модулем. Також сформульовано функціональні й нефункціональні вимоги до системи та обґрунтовано вибір технологічного стеку.

У другому розділі виконано проєктування та програмну реалізацію системи. Розроблено трирівневу архітектуру вебзастосунку, спроєктовано структуру бази даних, реалізовано серверну частину на Node.js та Express.js, клієнтський інтерфейс на основі HTML, CSS і JavaScript, інтеграцію з AI API, адміністративну панель, механізм авторизації, керування базою знань і журналом діалогів. Проведено тестування основних функцій системи.

Ключові слова:

AI-АСИСТЕНТ, КЛІЄНТСЬКИЙ СУПРОВІД, ВЕБСИСТЕМА, ЧАТ-БОТ, БАЗА ЗНАНЬ, NODE.JS, EXPRESS.JS, SQLITE, REST API

ЗМІСТ

ВСТУП

У сфері освітніх послуг важливе значення має якісна та своєчасна комунікація з клієнтами. Потенційні користувачі освітніх послуг звертаються із запитаннями щодо вартості навчання, доступних програм, формату занять, графіку, пробних уроків та умов запису. Від швидкості, повноти й зрозумілості відповіді часто залежить подальша зацікавленість клієнта та його рішення продовжити взаємодію з навчальним закладом.

У невеликих освітніх організаціях значна частина таких звернень обробляється вручну. Менеджер або адміністратор змушений багаторазово відповідати на однотипні запитання, уточнювати контактні дані, переглядати попередні повідомлення та передавати складні звернення відповідальній особі. За великої кількості повідомлень це може призводити до затримок у відповідях, втрати окремих звернень або неповного інформування клієнта.

Одним із підходів до розв’язання цієї задачі є використання AI-асистента. Сучасні мовні моделі здатні обробляти природну мову, розуміти різні формулювання запитань і формувати відповіді у зручному для користувача стилі. У системах клієнтського супроводу це дає змогу автоматизувати первинну консультацію, зменшити навантаження на менеджера та забезпечити швидке інформування клієнтів. При цьому AI-асистент не повинен повністю замінювати людину: у складних або нестандартних випадках звернення має передаватися адміністратору.

Актуальність теми дослідження визначається потребою в підвищенні якості та оперативності клієнтського супроводу в освітній сфері. Клієнти очікують швидкої відповіді та зрозумілої інформації, а працівники організації потребують інструменту, який зменшує кількість рутинних операцій, зберігає історію звернень і допомагає контролювати складні запити. У межах цієї роботи така задача розглядається на прикладі мовної школи, для якої важливими є консультації щодо курсів, формату навчання, вартості, пробних занять і запису клієнтів.

Метою кваліфікаційної роботи є проєктування та програмна реалізація вебсистеми підтримки супроводу клієнтів мовної школи з інтегрованим AI-асистентом, яка забезпечує автоматизовану первинну комунікацію з клієнтами, збереження історії діалогів, роботу з базою знань, розпізнавання намірів користувача та адресацію складних звернень до адміністратора.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

− проаналізувати специфіку клієнтського супроводу в освітній сфері та визначити основні проблеми предметної галузі;

− розглянути існуючі підходи й сучасні сервіси автоматизації клієнтської підтримки;

− сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;

− обґрунтувати вибір технологічного стеку для реалізації вебзастосунку;

− спроєктувати архітектуру системи та логіку взаємодії її компонентів;

− розробити структуру бази даних для збереження чат-сесій, історії діалогів, бази знань, користувачів і налаштувань;

− реалізувати серверну частину системи на основі Node.js та Express.js;

− реалізувати клієнтський інтерфейс на основі HTML, CSS, TailwindCSS та JavaScript;

− інтегрувати AI API для генерації відповідей на повідомлення клієнтів;

− реалізувати адміністративну панель для перегляду звернень, статистики, керування базою знань і налаштуваннями;

− провести тестування основних функцій системи та перевірити коректність її роботи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Характеристика предметної галузі

Сфера освітніх послуг активно змінюється під впливом цифрових технологій. Мовні школи, навчальні центри та курси додаткової освіти дедалі частіше використовують онлайн-канали для залучення клієнтів, консультування, запису на навчання та підтримки постійної комунікації з учнями. Для таких організацій важливим є не лише якісний навчальний процес, а й швидкість реакції на звернення, коректність наданої інформації, зручність запису на заняття та здатність зберігати історію взаємодії з клієнтом.

Мовна школа як бізнес-об’єкт поєднує освітні та сервісні процеси. З одного боку, вона надає навчальні послуги з вивчення іноземних мов, організовує заняття, підбирає викладачів і навчальні програми. З іншого боку, школа функціонує як комерційна організація, що працює з потенційними клієнтами, обробляє заявки, консультує щодо вартості та формату навчання, підтримує комунікацію з учнями та їхніми батьками, а також прагне утримувати клієнтів у довгостроковій перспективі.

Типовий процес взаємодії клієнта з мовною школою починається з первинного звернення. Клієнт може написати через сайт, месенджер, соціальну мережу або зателефонувати. На цьому етапі його зазвичай цікавлять такі питання: які мови доступні для вивчення, скільки коштує навчання, які є формати занять, чи можна навчатися індивідуально, чи є групи для дітей, який графік занять, як визначити рівень знань та як записатися на пробний урок. Значна частина таких запитань повторюється, тому менеджер змушений багаторазово надавати однотипні відповіді.

Після первинної консультації клієнт може перейти до наступного етапу — запису на пробне заняття або уточнення умов навчання. У цей момент важливо не втратити контакт, зафіксувати ім’я та номер телефону клієнта, зберегти зміст його звернення, визначити його потребу та, за необхідності, передати запит

адміністратору або менеджеру. Якщо ці дії виконуються вручну або через різні канали комунікації, виникає ризик втрати інформації, дублювання звернень, несвоєчасної відповіді або некоректного підбору курсу.

Основними проблемами, характерними для невеликих мовних шкіл, є:

— розрізненість каналів комунікації, коли звернення клієнтів надходять із сайту, месенджерів, соціальних мереж та телефонних дзвінків, але не зберігаються в єдиній системі;

— відсутність централізованої історії діалогів, через що менеджер не завжди може швидко переглянути попередні запити клієнта;

— високе навантаження на менеджера через необхідність відповідати на повторювані запитання щодо курсів, цін, графіку та формату навчання;

— залежність якості обслуговування від людського фактора, оскільки відповіді різних менеджерів можуть відрізнятися за повнотою, стилем та точністю;

— складність контролю необроблених звернень, особливо якщо клієнт потребує індивідуальної консультації або бажає записатися на навчання;

— обмежені можливості аналітики, оскільки без єдиного журналу діалогів важко оцінити кількість звернень, частку складних запитів, кількість потенційних записів і якість роботи клієнтського сервісу.

У таких умовах доцільною є розробка спеціалізованої веб-системи підтримки супроводу клієнтів, яка поєднує функції онлайн-чату, бази знань, журналу звернень та AI-асистента. Така система не обов’язково має замінювати повноцінну CRM або навчальну платформу. Її основне завдання — автоматизувати первинну комунікацію з клієнтом, надати швидку відповідь на типові запитання, зафіксувати контактні дані, зберегти історію діалогу та передати складні запити адміністратору.

Використання AI-асистента є доцільним саме на етапі первинної комунікації, оскільки значна частина звернень клієнтів має текстовий характер і може бути оброблена за допомогою технологій обробки природної мови. AI-асистент може формувати відповіді на основі бази знань мовної школи, пояснювати умови навчання, орієнтувати клієнта щодо доступних курсів та визначати ситуації, коли необхідне втручання людини.

Таким чином, предметна галузь характеризується потребою у швидкій, структурованій та контрольованій комунікації з клієнтами. Саме ця потреба визначає подальші вимоги до розроблюваної системи та обґрунтовує вибір архітектурних і технологічних рішень, що детально розглядаються у другому розділі роботи.

1.2 Аналіз підходів і сервісів автоматизації клієнтського супроводу

Перед розробкою власної системи підтримки супроводу клієнтів мовної школи доцільно розглянути як загальні підходи до автоматизації клієнтської комунікації, так і існуючі програмні сервіси, що вже використовуються для схожих задач. Такий аналіз дозволяє визначити переваги й недоліки наявних рішень, оцінити їхню придатність для мовної школи та обґрунтувати доцільність створення власної вебсистеми з AI-асистентом.

У межах цієї роботи аналіз здійснюється у двох напрямах. Перший напрям передбачає розгляд основних підходів до автоматизації клієнтського супроводу: CRM-систем, онлайн-чатів, сценарних чат-ботів та AI-асистентів на основі великих мовних моделей. Другий напрям охоплює огляд конкретних сервісів, функціональність яких є близькою до розроблюваної системи: Tidio Lyro, Intercom Fin та Zendesk AI.

1.2.1 Основні підходи до автоматизації клієнтського супроводу

У сфері клієнтської підтримки використовується кілька підходів до автоматизації комунікації. Кожен із них має власні переваги, обмеження та різний рівень придатності для невеликої мовної школи.

Першим підходом є використання універсальних CRM-систем. CRM-система призначена для зберігання даних про клієнтів, фіксації заявок, контролю етапів взаємодії та аналізу ефективності роботи менеджерів. Для великих організацій такі системи є корисними, оскільки дозволяють централізовано керувати продажами, комунікацією та аналітикою. Проте для невеликої мовної школи повноцінна CRM

може бути надмірно складною, дорогою та вимагати значного часу на налаштування. Крім того, універсальні CRM-системи не завжди мають зручний AI-модуль, здатний відповідати саме на типові запитання щодо типових запитань.

Другим підходом є використання онлайн-чатів із оператором. Такі рішення дозволяють встановити на сайт віджет, через який клієнт може написати менеджеру. Перевагою цього підходу є простота підключення та безпосередній контакт клієнта з представником школи. Однак онлайн-чат не вирішує проблему навантаження на менеджера, оскільки всі відповіді все одно надаються вручну. Якщо кількість звернень зростає, менеджер витрачає багато часу на повторювані консультації, а швидкість реакції на запити може знижуватися.

Третім підходом є використання сценарних, або rule-based, чат-ботів. Такі чат-боти працюють за заздалегідь визначеними правилами: користувач обирає варіанти з меню або вводить запит, який система намагається зіставити з підготовленими шаблонами. Цей підхід є корисним для простих і передбачуваних сценаріїв, наприклад для вибору курсу з меню або отримання контактної інформації. Проте клієнти мовної школи часто формулюють запитання у довільній формі. Наприклад, замість фрази «ціна курсу» користувач може написати: «Підкажіть, скільки коштуватиме англійська для дитини, якщо займатися двічі на тиждень?». Для сценарного чат-бота такі формулювання можуть бути складними, оскільки він не завжди розуміє зміст запиту та контекст.

Четвертим підходом є використання AI-асистентів на основі великих мовних моделей. Такі системи здатні обробляти природну мову, розуміти різні формулювання одного й того самого питання та генерувати відповідь у зручному для користувача стилі. Для мовної школи цей підхід є найбільш перспективним, оскільки дає змогу автоматизувати відповіді на типові запитання клієнтів, зберігати природність комунікації та адаптувати відповіді до конкретної бази знань школи.

Водночас AI-асистент не повинен працювати ізольовано від фактичних даних організації. Якщо модель не має доступу до актуальної інформації про курси, ціни, графік або правила школи, вона може сформувати неточну відповідь. Тому ефективним є підхід, за якого AI-асистент працює разом із базою знань. У такому

випадку модель формує відповіді не довільно, а на основі підготовленого контексту, який адміністратор може оновлювати через панель керування.

Для розроблюваної системи найбільш доцільним є комбінований підхід: вебчат для клієнта, база знань для зберігання актуальної інформації, AI-модуль для формування відповідей та адміністративна панель для контролю діалогів і оновлення даних. Така модель дозволяє автоматизувати типові звернення, але не усуває повністю роль менеджера. Складні або нестандартні запити мають передаватися адміністратору, що особливо важливо у сфері освітніх послуг.

1.2.2 Огляд сучасних сервісів AI-підтримки клієнтів

Для більш обґрунтованого визначення вимог до власної системи розглянемо три реальні сервіси, які використовуються для автоматизації клієнтської підтримки та мають функціональність, схожу до розроблюваної системи: Tidio Lyro, Intercom Fin та Zendesk AI. Ці рішення не є спеціалізованими саме для мовних шкіл, однак вони демонструють сучасні підходи до використання AI у клієнтській комунікації.

Tidio Lyro — це AI-агент для клієнтської підтримки, орієнтований переважно на малий і середній бізнес. Його основне призначення полягає в автоматизації відповідей на типові звернення клієнтів, зменшенні навантаження на службу підтримки та поєднанні AI-відповідей із можливістю живого чату. Сервіс може використовувати підготовлену інформацію компанії для відповідей на запитання, а також допомагає обробляти повторювані запити.

Перевагою Tidio Lyro є швидке впровадження, готовий чат-віджет для сайту, інтеграція з інструментами підтримки та орієнтація на бізнеси, які не мають великої технічної команди. Для мовної школи такий сервіс міг би бути корисним для автоматизації частини комунікації з потенційними учнями. Водночас його недоліком є залежність від зовнішньої платформи, тарифів і готової логіки сервісу. Крім того, можливості глибокої адаптації під навчальний процес, власну структуру бази даних і специфіку конкретної школи можуть бути обмеженими.

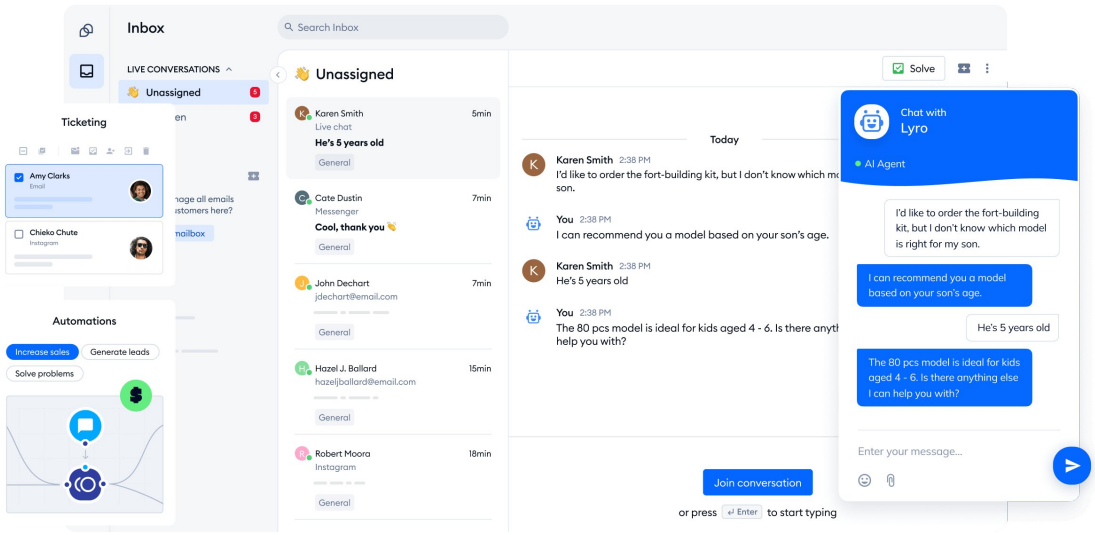

Рисунок 1.1 – Інтерфейс сервісу Tidio Lyro

Intercom Fin — це AI-асистент у складі платформи Intercom, призначений для автоматизованої підтримки клієнтів на основі довідкового центру, бази знань та інших джерел інформації компанії. Його основна ідея полягає в тому, що AI-агент аналізує запит користувача, знаходить релевантну інформацію у підготовлених матеріалах і формує відповідь. Якщо питання не може бути вирішене автоматично, воно може бути передане оператору служби підтримки.

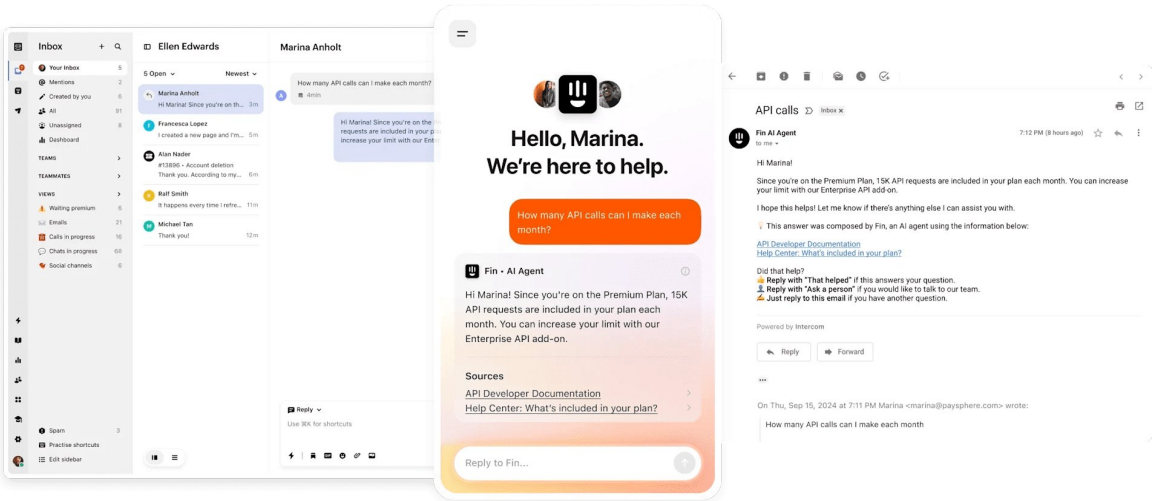

Рисунок 1.2 – Інтерфейс AI-асистента Intercom Fin

Перевагою Intercom Fin є високий рівень інтеграції з повноцінною платформою клієнтської підтримки. Сервіс має інструменти для роботи з

діалогами, аналізу невирішених питань, оцінювання якості відповідей та взаємодії між AI-агентом і живими операторами. Такий підхід є близьким до ідеї розроблюваної системи, оскільки також поєднує AI-відповіді з людським контролем. Недоліком для невеликої мовної школи може бути надмірність функціональності, залежність від платної SaaS-платформи та складність адаптації під простий навчальний проєкт.

Zendesk AI — це набір AI-інструментів у межах платформи Zendesk для автоматизації клієнтського сервісу. Платформа поєднує AI-агентів, систему заявок, базу знань, аналітику, автоматизацію процесів і засоби контролю якості підтримки. Zendesk AI орієнтований на компанії, які мають значний обсяг звернень і потребують повноцінної системи для обробки запитів у різних каналах.

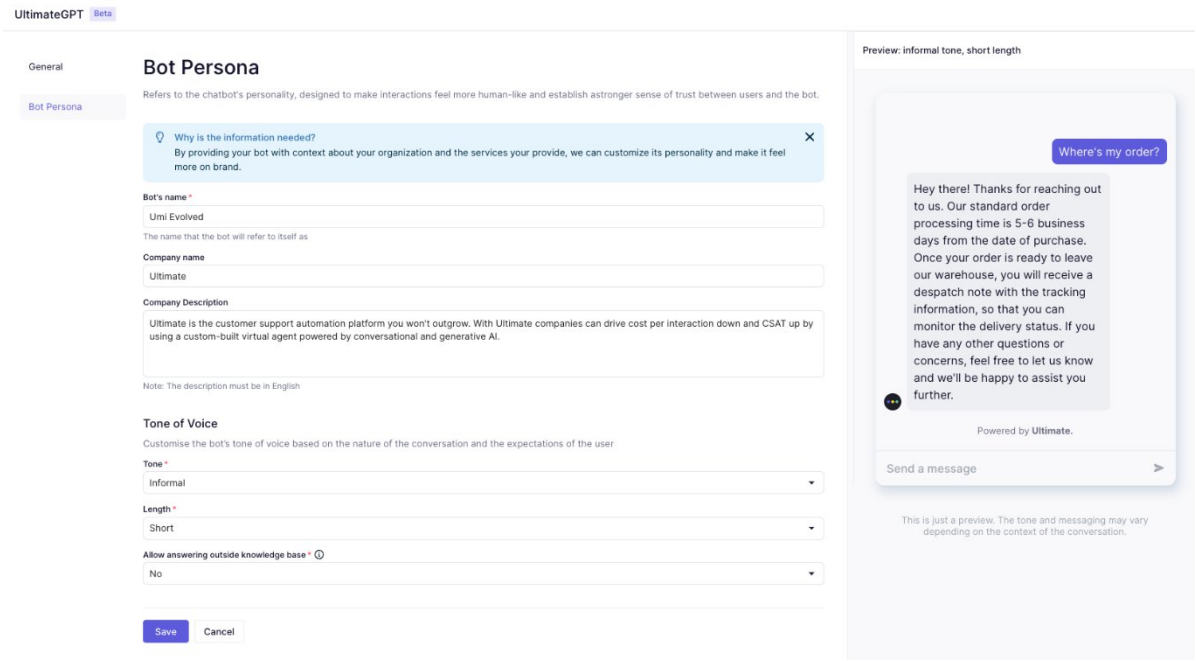

Рисунок 1.3 – Інтерфейс платформи Zendesk AI для підтримки клієнтів

Перевагою Zendesk AI є комплексність: система охоплює не лише чат, а й заявки, довідковий центр, аналітику, контроль якості та роботу операторів. Для великих організацій це є суттєвою перевагою. Проте для невеликої мовної школи така платформа може бути надмірною за масштабом, складністю впровадження та вартістю. Крім того, у межах кваліфікаційної роботи доцільніше реалізувати власну

спрощену систему, яка містить лише необхідні функції: чат, AI-відповіді, базу знань, журнал діалогів та адміністративну панель.

Порівняння розглянутих сервісів із розроблюваною системою наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих AI-сервісів із розроблюваною системою

Критерій	Tidio Lyro	Intercom Fin	Zendesk AI	Розроблювана система
Основне призначення	AI-чат і підтримка клієнтів	AI-агент для автоматизації підтримки	Комплексна AI-платформа	AI-супровід клієнтів онлайн школи
Наявність AI-відповідей	Так	Так	Так	Так
Робота з базою знань	Так	Так	Так	Так
Передача складних запитів людині	Так	Так	Так	Так
Адміністративна панель	Так	Так	Так	Так
Орієнтація саме на мовну школу	Ні	Ні	Ні	Так
Контроль над структурою бази даних	Обмежений	Обмежений	Обмежений	Повний
Складність впровадження	Низька	Середня	Висока	Помірна
Вартість використання	Залежить від тарифу	Залежить від тарифу	Залежить від тарифу	Мінімальна

Аналіз реальних сервісів показує, що сучасний ринок уже пропонує потужні рішення для AI-підтримки клієнтів. Найближчими до розроблюваної системи є сервіси, які поєднують онлайн-чат, AI-агента, базу знань, журнал звернень і можливість передачі складних питань оператору. Саме така логіка використовується у Tidio Lyro, Intercom Fin та Zendesk AI.

Водночас розглянуті сервіси є універсальними та орієнтованими на широкий спектр бізнесів. Вони не враховують специфіку мовної школи як освітньої організації, де важливими є питання віку учня, рівня володіння мовою, формату занять, пробного уроку, запису на курс і комунікації з батьками. Крім того, готові

SaaS-рішення мають обмеження щодо структури даних, логіки роботи, вартості та можливості повної адаптації під конкретний навчальний заклад.

Саме тому в межах кваліфікаційної роботи доцільною є розробка власної системи. Вона не повинна дублювати весь функціонал великих комерційних платформ, а має реалізовувати лише ті можливості, які є необхідними для мовної школи: клієнтський чат, пре-чат форму для збору контактних даних, AI-відповіді на основі бази знань, збереження історії діалогів, розпізнавання намірів користувача, ескалацію складних звернень та адміністративну панель для керування інформацією.

Таким чином, проведений аналіз підтверджує актуальність і практичну доцільність створення власної вебсистеми підтримки супроводу клієнтів мовної школи з використанням AI-асистента. Вона має бути простішою за комерційні платформи, але водночас достатньо функціональною для автоматизації первинної комунікації, підтримки менеджера та підвищення якості обслуговування клієнтів.

1.2.3 Обґрунтування вибору підходу для розроблюваної системи

На основі аналізу підходів і існуючих сервісів для розроблюваної системи обрано підхід власної вебсистеми з інтегрованим AI-модулем та базою знань. Такий вибір обґрунтовується кількома причинами.

По-перше, власна система дозволяє адаптувати логіку роботи саме до мовної школи. На відміну від універсальних сервісів, у ній можна врахувати типові сценарії комунікації: консультація щодо курсів, уточнення віку учня, підбір формату занять, запис на пробний урок, передача складного питання адміністратору.

По-друге, розробка власної системи забезпечує контроль над структурою даних. У базі даних можна окремо зберігати чат-сесії, повідомлення, результати розпізнавання намірів, базу знань, категорії та налаштування. Це спрощує подальший аналіз звернень і розширення функціональності.

По-третє, використання AI API дозволяє отримати якість відповідей, близьку до сучасних комерційних AI-сервісів, без необхідності самостійного навчання мовної моделі. Серверна частина системи формує системний промпт, додає до нього актуальну інформацію з бази знань і передає запит до AI-моделі. Завдяки цьому асистент відповідає не загальними фразами, а з урахуванням даних конкретної мовної школи.

Отже, аналіз існуючих підходів і реальних сервісів показав, що найбільш придатним рішенням для поставленої задачі є створення власної спеціалізованої вебсистеми. Вона поєднує переваги AI-підтримки, бази знань і адміністративного контролю, але при цьому залишається достатньо простою та зрозумілою для реалізації в межах кваліфікаційної роботи бакалавра.

1.3 Концепція системи та визначення вимог

На основі аналізу предметної галузі та існуючих підходів сформовано концепцію системи підтримки супроводу клієнтів мовної школи з використанням AI-асистента. Розроблювана система є вебзастосунком, який забезпечує первинну взаємодію клієнта з мовною школою через чат, автоматичне формування відповідей на основі бази знань, збереження історії діалогів та адміністрування інформаційного наповнення.

Система орієнтована на дві основні групи користувачів. Першою групою є клієнти мовної школи — відвідувачі сайту, які хочуть отримати консультацію, дізнатися інформацію про курси або залишити запит на запис. Другою групою є адміністратори або менеджери школи, які переглядають звернення, аналізують діалоги, керують базою знань та обробляють складні запити.

Функціонально система має забезпечувати такі можливості:

— відображення клієнтського інтерфейсу чату на сайті мовної школи;

— попередній збір контактних даних клієнта перед початком діалогу;

— надсилання повідомлення клієнта до серверної частини;

— формування відповіді AI-асистента на основі бази знань;

— збереження повідомлення клієнта та відповіді AI у базі даних;

— розпізнавання наміру користувача, зокрема бажання записатися на курс або необхідності звернення до живого адміністратора;

— відображення клієнту кнопки або посилання для переходу до адміністратора в разі ескалації;

— авторизацію адміністратора;

— перегляд статистики звернень;

— перегляд журналу діалогів;

— перегляд повної історії конкретної чат-сесії;

— керування базою знань через додавання, редагування та видалення інформаційних записів;

— налаштування службових параметрів системи, зокрема контактного посилання для ескалації.

Варто чітко окреслити межі роботи - розроблювана система не є повноцінною CRM-платформою, системою електронного навчання або фінансовим модулем. Вона не призначена для ведення розкладу занять, обліку оплат, оцінювання учнів чи управління викладачами. Її основне призначення — автоматизація первинної комунікації та підтримка менеджера під час обробки звернень клієнтів.

Такий підхід дозволяє зосередитися на конкретній і практично важливій задачі: створенні AI-асистента, який відповідає на типові запитання, працює з базою знань і передає складні звернення людині. Завдяки цьому система залишається достатньо простою для реалізації в межах бакалаврської роботи, але водночас має практичну цінність для реальної мовної школи.

1.3.1 Функціональні вимоги

Функціональні вимоги визначають перелік дій, які повинна виконувати система. Для розроблюваного вебзастосунку доцільно виділити кілька основних модулів.

Модуль клієнтського чату повинен забезпечувати можливість надсилання повідомлень клієнтом та отримання відповідей від AI-асистента. Перед початком діалогу система має запропонувати клієнту ввести ім’я та номер телефону. Це дозволяє адміністратору надалі ідентифікувати звернення та за потреби зв’язатися з клієнтом.

Модуль AI-асистента повинен обробляти текст повідомлення користувача, формувати запит до AI API, враховувати інформацію з бази знань і повертати відповідь українською мовою. AI-асистент має відповідати лише в межах інформації, що стосується мовної школи, та не повинен надавати неперевірені дані [18]. У разі, якщо питання виходить за межі бази знань або потребує участі людини, система повинна позначити звернення як таке, що потребує ескалації.

Модуль розпізнавання намірів повинен визначати ситуації, коли користувач хоче записатися на курс, отримати індивідуальну консультацію або звернутися до адміністратора. Це дає змогу не лише сформувати відповідь, а й класифікувати звернення для подальшого перегляду в адміністративній панелі.

Модуль збереження діалогів повинен фіксувати кожне повідомлення клієнта та відповідь AI-асистента в базі даних із прив’язкою до конкретної чат-сесії. Це необхідно для подальшого аналізу звернень, контролю якості відповідей і роботи менеджера з історією комунікації.

Модуль адміністративної панелі повинен забезпечувати менеджеру або адміністратору доступ до статистики системи, журналу діалогів і бази знань. Адміністратор повинен мати можливість переглядати звернення, бачити, які з них потребують уваги, відкривати повну історію сесії та оновлювати інформаційні блоки, на основі яких працює AI-асистент.

Модуль авторизації повинен забезпечувати захищений вхід до адміністративної частини системи за допомогою логіна та пароля. Це необхідно для запобігання несанкціонованому доступу до історії діалогів, контактних даних клієнтів і налаштувань системи.

Модуль керування базою знань повинен дозволяти адміністратору створювати, редагувати та видаляти записи, які використовуються AI-асистентом

при формуванні відповідей. Записи можуть бути поділені за категоріями, наприклад: курси, ціни, графік, викладачі, пробне заняття, контакти, правила навчання.

Модуль налаштувань повинен дозволяти зберігати службові параметри, які використовуються в роботі системи. До таких параметрів може належати назва школи, посилання на Telegram адміністратора або інша контактна інформація.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги описують якісні характеристики системи, які впливають на її зручність, надійність і можливість подальшого використання.

Вимоги до зручності використання. Інтерфейс клієнтського чату має бути простим і зрозумілим для користувача без спеціальної підготовки. Клієнт повинен легко почати діалог, ввести контактні дані та отримати відповідь. Адміністративна панель має мати логічну структуру, зрозумілу навігацію та візуальне відображення ключових показників.

Вимоги до продуктивності. Система повинна швидко обробляти стандартні запити, пов’язані із завантаженням сторінок, переглядом журналу діалогів і роботою з базою знань. Час генерації AI-відповіді може залежати від зовнішнього API, тому інтерфейс повинен відображати індикатор очікування, щоб користувач розумів, що запит обробляється.

Вимоги до надійності. Система повинна коректно обробляти помилки, зокрема відсутність повідомлення, проблеми з підключенням до AI API, помилки бази даних або невалідні дані у формах. У разі тимчасової недоступності зовнішнього AI-сервісу система не повинна аварійно завершувати роботу.

Вимоги до безпеки. Доступ до адміністративної панелі має бути обмежений авторизованими користувачами. Паролі адміністраторів повинні зберігатися у хешованому вигляді. Система повинна захищати адміністративні маршрути від несанкціонованого доступу, а також виконувати базову перевірку введених даних.

Вимоги до підтримуваності. Код системи має бути поділений на логічні частини: клієнтський інтерфейс, серверні маршрути, контролери, сервіси та модуль роботи з базою даних. Така структура спрощує подальшу модифікацію системи, додавання нових функцій та виправлення помилок.

Вимоги до портативності. Система має працювати у сучасних веббраузерах і не вимагати складного встановлення на клієнтському боці. Для запуску серверної частини має бути достатньо стандартного середовища Node.js та встановлених залежностей проєкту.

Вимоги до масштабованості. Оскільки система орієнтована на невелику або середню мовну школу, вона не потребує складної мікросервісної архітектури. Водночас обрана структура повинна дозволяти у майбутньому розширити функціональність: додати нові канали комунікації, інтеграцію з CRM, розкладом занять або платіжними сервісами.

1.4 Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку є важливим етапом проєктування програмної системи, оскільки саме від обраних технологій залежить складність реалізації, швидкість роботи, зручність підтримки та можливість подальшого розвитку вебзастосунку. Для розробки системи підтримки супроводу клієнтів мовної школи з використанням AI-асистента було обрано технології, які забезпечують просту реалізацію клієнт-серверної архітектури, збереження даних, інтеграцію з AI API та створення зручного інтерфейсу для клієнта й адміністратора.

Під час вибору технологій враховувалися такі критерії: відповідність функціональним вимогам системи, простота освоєння та використання, наявність документації, підтримка сучасних підходів до веброзробки, можливість швидкого створення прототипу, а також доцільність застосування в межах бакалаврської кваліфікаційної роботи.

Для реалізації системи було обрано такий основний технологічний стек: Node.js і Express.js для серверної частини, HTML, CSS, TailwindCSS і JavaScript для

клієнтської частини, SQLite для збереження даних, а також OpenAI-сумісний API для інтеграції AI-асистента.

1.4.1 Серверна частина (Backend)

Серверна частина є центральним рівнем системи, який відповідає за обробку запитів, взаємодію з базою даних, авторизацію адміністратора, збереження діалогів, формування запитів до AI API та повернення результатів клієнтській частині. Саме backend реалізує основну бізнес-логіку вебсистеми.

Для розробки серверної частини обрано Node.js — середовище виконання JavaScript на стороні сервера. Його використання є доцільним для даного проєкту, оскільки дозволяє застосовувати одну мову програмування як на клієнтському, так і на серверному рівні. Це спрощує розробку, зменшує кількість технологічних переходів і робить структуру проєкту більш зрозумілою.

Node.js має асинхронну неблокуючу модель обробки операцій введення-виведення. Це є важливою перевагою для вебсистеми, яка одночасно виконує запити до бази даних, обробляє HTTP-звернення користувачів і взаємодіє із зовнішнім AI API. Завдяки асинхронності сервер може не блокувати роботу застосунку під час очікування відповіді від зовнішнього сервісу.

Для побудови вебсервера та REST API обрано Express.js. Це мінімалістичний фреймворк для Node.js, який надає зручні засоби для маршрутизації запитів, підключення middleware, обробки JSON-даних та організації API-ендпоінтів. У межах розроблюваної системи Express.js використовується для реалізації маршрутів чату, авторизації адміністратора, роботи з базою знань, перегляду журналу діалогів та налаштувань системи.

Backend системи виконує такі основні функції:

— приймання повідомлень від клієнтського чату;

— створення та оновлення чат-сесій;

— збереження повідомлень користувача та відповідей AI у базі даних;

— отримання інформації з бази знань для формування контексту AI-запиту;

— надсилання запитів до AI API;

— обробка відповіді AI-моделі;

— визначення намірів користувача, зокрема запиту на запис або потреби в ескалації;

— передача клієнту готової JSON-відповіді;

— авторизація адміністратора;

— захист адміністративних маршрутів;

— виконання CRUD-операцій із записами бази знань.

Для захисту адміністративної частини використовується механізм JWT-автентифікації. Після успішного входу адміністратор отримує токен, який використовується для перевірки доступу до захищених маршрутів. Такий підхід дозволяє відокремити публічну частину системи, доступну клієнтам, від адміністративної панелі, яка має бути доступна лише авторизованому користувачу.

Для безпечного зберігання паролів використовується bcrypt. Паролі не зберігаються у відкритому вигляді, а перетворюються на хеш. Це підвищує рівень безпеки системи, оскільки навіть у разі несанкціонованого доступу до бази даних неможливо безпосередньо отримати пароль адміністратора.

Таким чином, використання Node.js та Express.js забезпечує достатню продуктивність, простоту реалізації, зручну організацію REST API та можливість інтеграції з базою даних і AI-сервісом.

1.4.2 Клієнтська частина (Frontend)

Клієнтська частина системи відповідає за взаємодію користувача з вебзастосунком. Вона включає інтерфейс клієнтського AI-чату, форму попередньої реєстрації, вікно діалогу, а також адміністративну панель для перегляду звернень, керування базою знань і налаштування системи.

Для реалізації frontend-частини обрано HTML5, CSS3, TailwindCSS та JavaScript. Такий вибір є доцільним для системи невеликого та середнього

масштабу, оскільки дозволяє створити адаптивний, зрозумілий і візуально привабливий інтерфейс без використання складних frontend-фреймворків.

HTML5 використовується для побудови структури вебсторінок. За його допомогою створюються основні елементи інтерфейсу: форма входу адміністратора, форма пре-чат реєстрації клієнта, область повідомлень, поле введення тексту, кнопки, таблиці, картки статистики та модальні вікна.

CSS3 використовується для візуального оформлення сторінок. За допомогою стилів задаються кольори, відступи, шрифти, розміри елементів, ефекти наведення, анімації та адаптивність. У системі також використовується підхід glassmorphism, який передбачає застосування напівпрозорих блоків, розмиття фону та м’яких візуальних ефектів.

TailwindCSS використовується як утилітарний CSS-фреймворк, що дозволяє швидко створювати сучасний адаптивний інтерфейс за допомогою готових класів. Його перевагою є те, що значна частина оформлення може бути задана без написання великої кількості окремого CSS-коду. Це прискорює розробку та робить верстку більш структурованою.

JavaScript використовується для реалізації інтерактивної логіки клієнтської частини. Він відповідає за обробку подій, відправлення форм, динамічне додавання повідомлень у чат, відображення індикатора очікування відповіді, роботу з localStorage, відкриття модальних вікон і оновлення даних в адміністративній панелі.

Для обміну даними між frontend і backend використовується Fetch API. Завдяки цьому клієнтська частина може надсилати HTTP-запити до серверної частини у форматі JSON та отримувати відповіді без перезавантаження сторінки. Це робить роботу системи більш зручною для користувача та наближає її до поведінки сучасних вебзастосунків.

Клієнтська частина системи виконує такі функції:

— відображення головної сторінки з AI-чатом;

— збір імені та номера телефону клієнта перед початком діалогу;

— надсилання повідомлення користувача на сервер;

— виведення відповіді AI-асистента у вікні чату;

— відображення повідомлення про ескалацію до адміністратора;

— збереження ідентифікатора сесії в localStorage;

— відображення сторінки авторизації адміністратора;

— завантаження статистики, журналу діалогів і бази знань в адмін-панелі;

— створення, редагування та видалення записів бази знань;

— відображення сповіщень про успішні або помилкові дії.

Використання HTML, CSS, TailwindCSS і JavaScript дозволяє реалізувати легкий, адаптивний і достатньо функціональний інтерфейс, який відповідає потребам клієнта мовної школи та адміністратора системи.

1.4.3 База даних

База даних є важливою складовою системи, оскільки забезпечує збереження всієї інформації, необхідної для роботи вебзастосунку. До таких даних належать облікові записи адміністраторів, чат-сесії клієнтів, історія повідомлень, категорії бази знань, інформаційні записи та службові налаштування системи.

Для реалізації бази даних обрано SQLite. Це вбудована реляційна система управління базами даних, яка не потребує окремого серверного процесу та зберігає всю інформацію у файлі. Такий варіант є доцільним для бакалаврської кваліфікаційної роботи та невеликої вебсистеми, оскільки значно спрощує розгортання, тестування й перенесення застосунку між середовищами.

Використання SQLite є оптимальним для даного проєкту, оскільки ця база даних забезпечує простоту, достатню швидкість роботи, підтримку реляційної структури та зручність розгортання. У майбутньому, за потреби масштабування, систему можна адаптувати до використання більш потужної серверної СУБД, наприклад PostgreSQL або Microsoft SQL Server.

1.4.4 Модуль штучного інтелекту

Модуль штучного інтелекту є ключовою особливістю розроблюваної системи, оскільки саме він забезпечує автоматизоване формування відповідей на повідомлення клієнтів мовної школи. На відміну від звичайного чат-бота, який працює за жорсткими сценаріями, AI-асистент може обробляти природну мову та розуміти різні формулювання запитань користувачів.

Для реалізації AI-модуля використовується OpenAI-сумісний API через зовнішнього AI-провайдера. Серверна частина системи формує запит, додає до нього контекст із бази знань і надсилає його до AI API. У відповідь модель повертає згенерований текст, який відображається клієнту у вікні чату.

Основою роботи AI-модуля є системний промпт. Він визначає роль асистента, стиль комунікації, правила формування відповідей і межі використання інформації. У системному промпті AI-асистенту задається інструкція відповідати українською мовою, використовувати лише інформацію з бази знань та не вигадувати дані, яких немає в системі.

Важливою особливістю AI-модуля є використання динамічного контексту. Перед зверненням до AI API серверна частина отримує актуальні записи з бази знань і додає їх до запиту. Завдяки цьому AI-асистент формує відповіді на основі інформації, яку адміністратор може оновлювати через панель керування. Це дозволяє змінювати умови навчання, ціни, опис курсів або контактні дані без зміни програмного коду.

AI-модуль виконує такі функції:

— приймає текст повідомлення користувача від backend-частини;

— отримує контекст із бази знань;

— формує системну інструкцію для AI-моделі;

— надсилає запит до AI API;

— отримує згенеровану відповідь;

— визначає наявність спеціальних маркерів намірів;

— повертає відповідь, очищену від службових маркерів, до серверної частини.

Окремо реалізується логіка розпізнавання намірів користувача. Якщо клієнт хоче записатися на курс, система може позначити таке звернення спеціальним прапорцем booking. Якщо питання є складним або виходить за межі бази знань, система встановлює прапорець escalated і пропонує клієнту звернутися до адміністратора. Такий механізм дозволяє поєднати автоматизацію з людським контролем.

Використання AI API є доцільним для розроблюваної системи, оскільки воно забезпечує гнучку обробку природної мови, високу якість відповідей і можливість швидкої інтеграції з вебзастосунком. При цьому система залишається контрольованою, оскільки AI-асистент працює не самостійно, а на основі підготовленої бази знань і правил, заданих у системному промпті.

Зведений технологічний стек системи наведено в таблиці 1.2.

Таблиця 1.2 – Технологічний стек розроблюваної системи

Рівень системи	Технологія	Призначення
Backend	Node.js	Виконання серверної логіки
Backend	Express.js	Побудова REST API та маршрутизація запитів
Backend	JWT	Авторизація адміністратора та захист маршрутів
Backend	bcrypt	Хешування паролів
Backend	dotenv	Робота зі змінними середовища
Backend	cookie-parser	Обробка cookie у запитах
Frontend	HTML5	Структура вебсторінок
Frontend	CSS3	Візуальне оформлення інтерфейсу
Frontend	TailwindCSS	Адаптивна верстка та швидке створення UI
Frontend	JavaScript	Інтерактивна логіка клієнтської частини
Frontend	Fetch API	Асинхронний обмін даними з сервером
База даних	SQLite	Збереження користувачів, сесій, діалогів, бази знань і налаштувань
AI-модуль	OpenAI-сумісний API	Генерація відповідей AI-асистента
AI-модуль	Системний промпт	Передача правил поведінки та контексту моделі

Отже, обраний технологічний стек повністю відповідає вимогам розроблюваної системи. Backend на основі Node.js та Express.js забезпечує обробку запитів і бізнес-логіку, frontend на HTML, CSS, TailwindCSS і JavaScript створює зручний інтерфейс користувача, SQLite забезпечує просте та надійне збереження даних, а AI API дозволяє реалізувати інтелектуальну обробку звернень клієнтів. Сукупність цих технологій створює основу для практичної реалізації системи, яка детально розглядається у другому розділі роботи.

Висновки до розділу

:

У першому розділі було проаналізовано специфіку клієнтського супроводу в мовній школі та визначено основні проблеми, пов’язані з обробкою звернень, повторюваними консультаціями й відсутністю централізованої історії діалогів. Огляд підходів до автоматизації та сучасних сервісів AI-підтримки клієнтів показав, що готові платформи є функціональними, але не повністю враховують специфіку мовної школи та можуть бути надмірними для невеликого навчального закладу. За результатами аналізу обґрунтовано доцільність створення власної вебсистеми з AI-асистентом, базою знань, клієнтським чатом та адміністративною панеллю, що стало основою для подальшого проєктування у другому розділі.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ З AI-АСИСТЕНТОМ

2.1 Архітектурне проектування та моделювання системи

Архітектурне проектування є одним із ключових етапів розробки вебсистеми, оскільки саме на цьому етапі визначається загальна структура програмного рішення, склад його основних компонентів і принципи їхньої взаємодії. Для системи підтримки супроводу клієнтів мовної школи важливо забезпечити логічний поділ між клієнтським інтерфейсом, серверною бізнес-логікою, базою даних та зовнішнім AI-сервісом. Такий підхід дозволяє зробити систему зрозумілою для реалізації, зручною для подальшої підтримки та придатною для розширення.

У межах цього підрозділу розглядається вибір архітектурного шаблону, моделюється взаємодія користувача, адміністратора та програмних компонентів, а також описується алгоритм обробки повідомлення AI-асистентом. Це дозволяє перейти від загальної концепції системи до конкретної логіки її програмної реалізації.

2.1.1 Вибір архітектурного шаблону та патернів розробки

Проектування веб-системи є фундаментальним етапом розробки, що визначає її масштабованість, підтримуваність та надійність. Для реалізації AI-асистента мовної школи MTSchool було обрано трьохланкову (three-tier) клієнт-серверну архітектуру, яка є одним із найбільш поширених і перевірених підходів у сучасній веб-розробці.

Трьохланкова архітектура передбачає чіткий логічний поділ системи на три рівні: рівень представлення (клієнтська частина, Frontend), рівень бізнес-логіки (серверна частина, Backend) та рівень даних (база даних і зовнішні сервіси). Такий

підхід забезпечує незалежність кожного рівня від інших, що спрощує модифікацію та масштабування окремих компонентів без необхідності перебудови всієї системи.

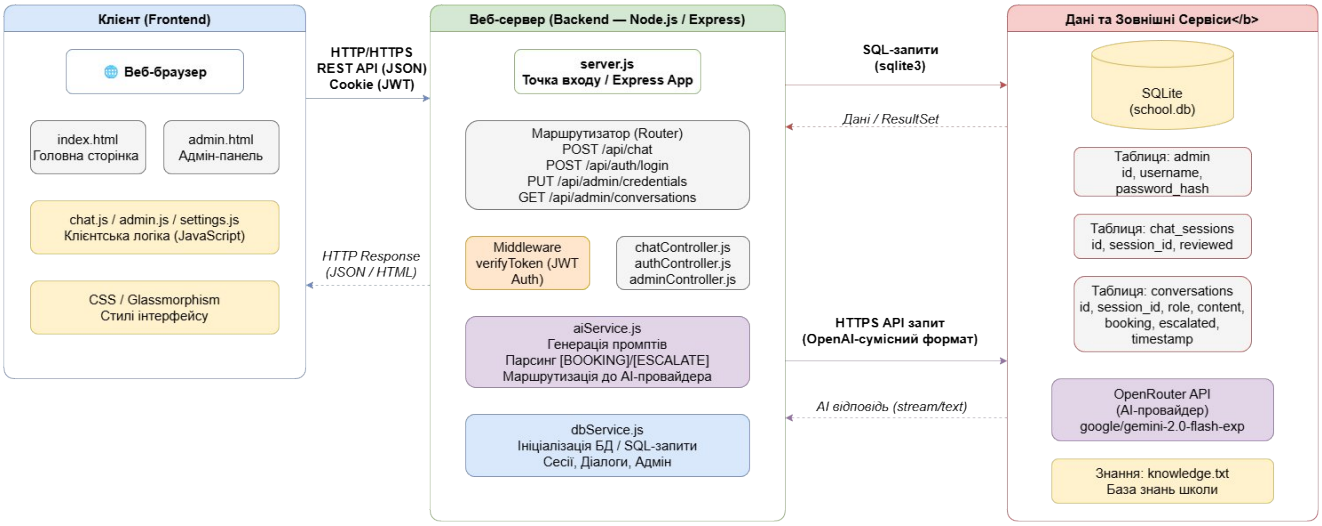

Рисунок 2.1 – Схема трьохланкової клієнт-серверної архітектури системи

Рівень 1 — Клієнтська частина (Frontend) реалізована у вигляді статичних HTML-сторінок, що обслуговуються Express-сервером. До складу клієнтської частини входять: index.html — головна сторінка з інтерфейсом чату та маркетинговим блоком, admin.html — захищена панель адміністратора, login.html — сторінка авторизації. Клієнтська логіка реалізована мовою JavaScript у файлах chat.js, admin.js та settings.js, а стилізація здійснюється за допомогою TailwindCSS v4 у поєднанні з кастомними CSS-стилями у файлі custom.css, що реалізують концепцію glassmorphism.

Рівень 2 — Серверна частина (Backend) побудована на платформі Node.js з використанням фреймворку Express.js. Точкою входу є файл server.js, який ініціалізує Express-додаток, підключає middleware та реєструє всі маршрути API. Бізнес-логіка розподілена між трьома контролерами: chatController.js (обробка чату), authController.js (авторизація) та adminController.js (функції адміністратора). Взаємодія з AI реалізована через сервісний шар aiService.js.

Рівень 3 — Рівень даних складається із вбудованої реляційної СУБД SQLite (файл database.sqlite) та зовнішнього API-сервісу OpenAI, доступного через

провайдер OpenRouter. SQLite не потребує окремого сервера баз даних, що значно спрощує розгортання системи.

Окрім трьохланкової архітектури, в проєкті застосовано патерн MVC (Model-View-Controller), що забезпечує чіткий розподіл обов'язків між компонентами. Роль View виконують HTML-сторінки в директорії public/, роль Controller — файли в src/controllers/, а роль Model — сервіс роботи з базою даних src/config/db.js.

Для захисту маршрутів адміністратора застосовано патерн Middleware Chain — перед виконанням будь-якого захищеного запиту до адмін-API викликається проміжний обробник verifyToken з директорії src/middleware/auth.js, який перевіряє наявність та дійсність JWT-токена у файлах cookie браузера.

Взаємодія між рівнями здійснюється виключно через визначений REST API: клієнт надсилає HTTP-запити до серверу (наприклад, POST /api/chat, GET /api/admin/logs), сервер обробляє їх та повертає JSON-відповіді. Такий підхід повністю відокремлює Frontend від Backend, що дозволяє за необхідності замінити клієнтську частину або реалізувати мобільний додаток без жодних змін на серверній стороні.

2.1.2 Моделювання взаємодії компонентів

Для формального опису вимог та взаємодії акторів системи було розроблено *Use Case Діаграму* (діаграму прецедентів використання). В системі MTSchool AI виокремлено три головні актори: Користувач (клієнт мовної школи), Адміністратор та Система (автоматизована бізнес-логіка серверу).

Актор «Користувач» взаємодіє із системою через веб-браузер і має доступ до таких прецедентів: перегляд головної сторінки, розпочало чат-сесію, надсилання повідомлення до AI-боту, перегляд відповіді та історії чату. Окремо виділено два розширюючих прецеденти: «Залишити запит на запис» та «Ініціювати ескалацію до живого оператора», які активуються системою автоматично, якщо AI виявляє відповідний намір у тексті повідомлення.

Актор «Адміністратор» отримує доступ до захищеної панелі після проходження процесу автентифікації. Його прецеденти охоплюють: перегляд статистики системи (загальна кількість діалогів, кількість ескалацій та запитів на запис), перегляд та фільтрацію журналу всіх діалогів, перегляд конкретної чат-сесії з повним транскриптом, позначення сесій як переглянутих, а також управління обліковими даними (зміна логіна та пароля) на сторінці налаштувань.

Актор «Система» є внутрішнім актором, що реалізує автоматизовані функції без прямої взаємодії з людиною: JWT-автентифікацію, генерацію AI-відповіді через виклик зовнішнього API, збереження діалогів у базу даних та визначення наміру повідомлення.

Рисунок 2.2 – UML Use Case Діаграма системи

Для детального опису динамічної взаємодії компонентів під час обробки одного повідомлення було розроблено *UML Sequence Діаграму* (діаграму послідовностей). Вона відображає точний хронологічний порядок обміну

повідомленнями між шістьма учасниками: Клієнт (Браузер), Express-сервер (Router), Middleware (verifyToken), chatController, aiService та База даних SQLite.

Послідовність починається з надсилання клієнтом POST-запиту на ендпоінт /api/chat з тілом, що містить текст повідомлення та ідентифікатор сесії. Express-сервер приймає запит, після чого для публічного маршруту чату middleware JWT-перевірку пропускає, і запит передається безпосередньо до chatController.

Контролер виконує два запити до бази даних: отримання або створення поточної чат-сесії та завантаження всієї бази знань школи для формування контексту AI-запиту. Після цього контролер передає управління сервісу aiService, який будує системний промпт, формує масив повідомлень та виконує HTTPS-запит до зовнішнього API OpenRouter. Отриману від AI відповідь сервіс парсить на наявність маркерів намірів, після чого повертає структурований об'єкт контролеру. Контролер зберігає повідомлення користувача та відповідь AI до бази даних і формує фінальну JSON-відповідь, яка через Express повертається клієнту.

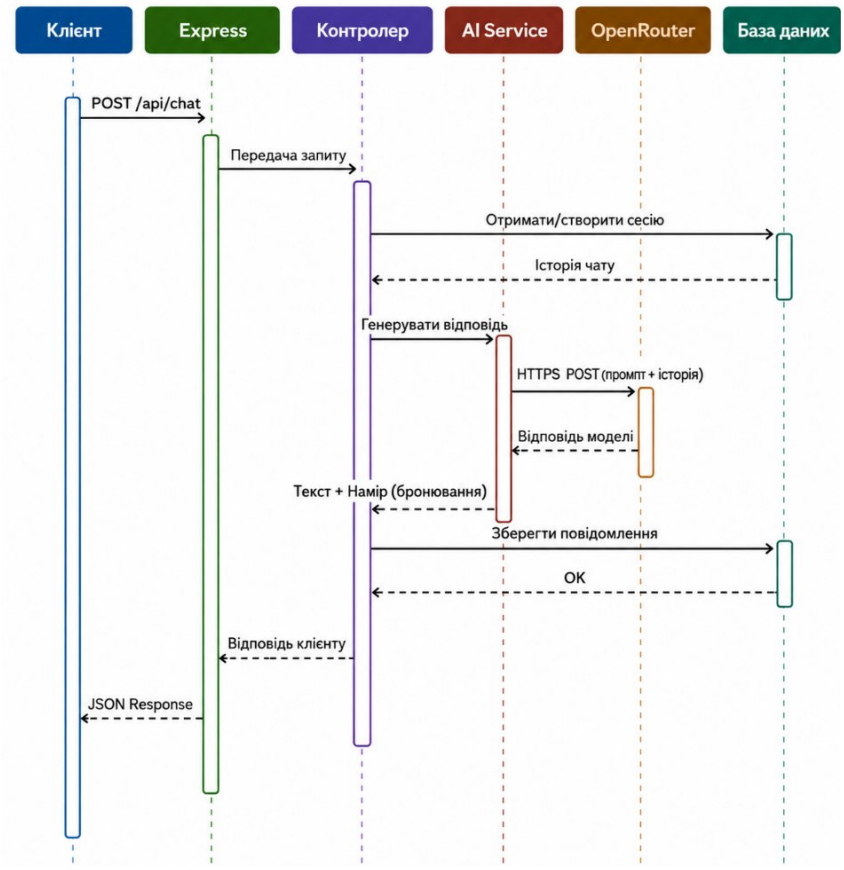

Рисунок 2.3 – Sequence Діаграма: послідовність обробки повідомлення

2.1.3 Алгоритмічне моделювання логіки роботи ШІ

Центральним алгоритмом системи є процес обробки вхідного повідомлення користувача, що включає генерацію AI-відповіді та визначення намірів (intent detection). Для графічного опису цього алгоритму розроблено блок-схему.

Алгоритм починається з отримання серверного POST-запиту, що містить два обов'язкових параметри: message (текст повідомлення користувача) та sessionId (унікальний ідентифікатор сесії, що генерується на стороні клієнта).

На наступному кроці контролер встановлює з'єднання з базою даних SQLite через функцію getDbConnection() і виконує два паралельних запити: отримання всіх записів таблиці knowledge_base для формування контексту відповіді та отримання URL Telegram-адміністратора з таблиці settings. Ці дані передаються до функції generateResponse сервісу aiService.js.

Функція generateResponse спочатку перевіряє режим роботи системи. Якщо змінна середовища OPENAI_API_KEY відсутня або містить значення-заглушку, система переходить у режим Mock (локальний пошук за ключовими словами). В іншому разі — формується системний промпт, що включає всі записи бази знань та набір суворих правил поведінки для AI-моделі: відповідати виключно на основі наданого контексту, використовувати лише українську мову та у разі неможливості відповіді — додавати у текст маркер [ESCALATE].

Сформований запит надсилається до зовнішнього API через клієнт бібліотеки openai, з параметром temperature: 0.2, що обмежує «творчість» моделі та мінімізує генерацію неправдивої інформації (галюцинацій).

Після отримання текстової відповіді від AI-моделі виконується двоетапний аналіз намірів. Перший етап — пошук маркера бронювання. Функція перевіряє наявність у тексті відповіді спеціального маркера [BOOKING], встановленого правилами системного промпту. Якщо маркер знайдено, встановлюється прапорець isBooking = true, а сам маркер видаляється з тексту перед відправкою клієнту. Другий етап — пошук маркера ескалації. Аналогічна перевірка

виконується для маркера [ESCALATE]: при його виявленні встановлюється escalated = true, що в подальшому включає у JSON-відповідь поле telegramUrl для відображення кнопки зв'язку з адміністратором.

Після завершення аналізу контролер виконує INSERT-запит до таблиці conversations, зберігаючи повідомлення користувача, відповідь AI, значення прапорців ескалації та часову мітку. Клієнту повертається JSON-об'єкт такого вигляду: { response, escalated, telegramUrl? }. Клієнтський JavaScript обробляє цю відповідь і, залежно від значень прапорців, відображає або стандартну текстову відповідь, або додатковий банер ескалації з посиланням на Telegram.

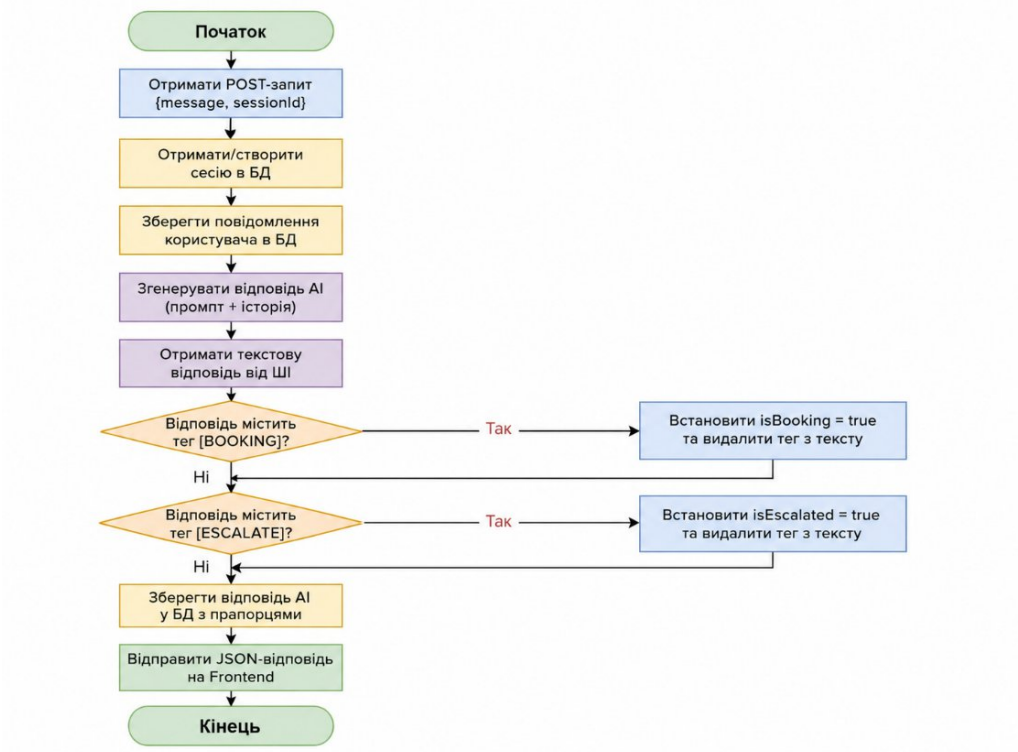

Рисунок 2.4 – Блок-схема алгоритму обробки вхідного повідомлення

2.2 Проектування структури бази даних

Інформаційне забезпечення є невід'ємною складовою будь-якої веб-системи, оскільки визначає спосіб зберігання, структурування та доступу до даних, необхідних для функціонування всіх її підсистем. У проєкті MTSchool AI інформаційне забезпечення реалізоване на основі реляційної системи управління базами даних SQLite, яка зберігає всі дані у єдиному файлі database.sqlite у

кореневій директорії проєкту. Вбудована природа SQLite усуває потребу в окремому сервері баз даних та спрощує розгортання і перенесення системи між середовищами.

Проєктування інформаційного забезпечення охоплює два взаємопов'язаних завдання: визначення логічної та фізичної структури бази даних (склад таблиць, типи даних, ключі та обмеження цілісності) та розробку концептуальної схеми у вигляді ER-моделі, що графічно відображає сутності системи та зв'язки між ними. Результатом цього етапу є повна специфікація схеми бази даних, яка реалізована у файлі src/config/db.js через функцію initDb(), що автоматично створює та наповнює початковими даними всі необхідні таблиці при кожному запуску сервера.

2.2.1 Логічна та фізична структура бази даних

База даних системи MTSchool AI складається з шести таблиць: users, chat_sessions, conversations, categories, knowledge_base та settings. Кожна таблиця реалізує чітко визначену предметну область і разом вони утворюють цілісну схему даних, достатню для підтримки всіх функцій системи: авторизації адміністратора, ведення журналу діалогів, динамічного управління базою знань та конфігурації сервісу.

Таблиця *users* призначена для зберігання облікових записів адміністраторів системи. Паролі у цій таблиці ніколи не зберігаються у відкритому вигляді — лише bcrypt-хеш із фактором складності 10, що унеможливлює відновлення оригінального пароля навіть у разі компрометації файлу бази даних.

Таблиця 2.1 – Специфікація таблиці users

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний числовий ідентифікатор запису
username	TEXT	NOT NULL, UNIQUE	Логін адміністратора для входу в систему
password_hash	TEXT	NOT NULL	Хеш пароля
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час реєстрації

Таблиця *chat_sessions* реєструє унікальні чат-сесії відвідувачів сайту. Первинним ключем слугує текстовий session_id, що генерується на стороні клієнта функцією generateSessionId() і зберігається у localStorage браузера між відвідуваннями. Це дозволяє системі «впізнавати» повторних відвідувачів та відновлювати контекст їхнього діалогу. Прапорець reviewed використовується адміністраторською панеллю для фільтрації вже опрацьованих сесій.

Таблиця 2.2 – Специфікація таблиці chat_sessions

Назва поля	Тип даних	Обмеження	Опис
session_id	TEXT	PRIMARY KEY	Унікальний ідентифікатор сесії
client_name	TEXT	—	Ім'я клієнта
client_phone	TEXT	—	Телефон клієнта
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час початку сесії
reviewed	INTEGER	DEFAULT 0	Прапорець перегляду адміністратором

Таблиця *conversations* є центральною сутністю системи — саме тут зберігаються всі повідомлення кожного діалогу. Поле session_id є зовнішнім ключем, що посилається на chat_sessions.session_id і утворює зв'язок «один-до-багатьох»: одна сесія може містити необмежену кількість записів у цій таблиці. Прапорці escalated та booking фіксують результат аналізу намірів (intent detection), виконаного AI-сервісом.

Таблиця 2.3 – Специфікація таблиці conversations

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор повідомлення
session_id	TEXT	NOT NULL, FOREIGN KEY → chat_sessions	Ідентифікатор сесії
user_message	TEXT	NOT NULL	Текст повідомлення, надісланого користувачем
ai_response	TEXT	NOT NULL	Текстова відповідь, згенерована AI-сервісом
escalated	INTEGER	DEFAULT 0	Прапорець ескалації
booking	INTEGER	DEFAULT 0	Прапорець бронювання
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час збереження повідомлення

Таблиця *categories* забезпечує каталогізацію записів бази знань за тематичними групами. Поле name є унікальним ключем категорії (наприклад, 'courses', 'prices', 'teachers'), а display_name — її відображувана назва українською мовою для адмін-інтерфейсу. Записи цієї таблиці пов'язані з таблицею knowledge_base через зовнішній ключ category.

Таблиця 2.4 – Специфікація таблиці categories

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний числовий ідентифікатор категорії
name	TEXT	NOT NULL, UNIQUE	Системний ключ категорі
display_name	TEXT	NOT NULL	Відображувана назва категорії для UI

Таблиця *knowledge_base* є динамічним сховищем інформаційних блоків школи, якими оперує AI при формуванні відповідей. Кожен запис прив'язаний до категорії через зовнішній ключ category → categories.name. Поля title та content містять відповідно заголовок та детальний текст інформаційного блоку. Адміністратор може повністю керувати вмістом цієї таблиці через CRUD-ендпоінти адмін-панелі, не вносячи жодних змін до програмного коду.

Таблиця 2.5 – Специфікація таблиці knowledge_base

Назва поля	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний числовий ідентифікатор запису
category	TEXT	NOT NULL, FOREIGN KEY → categories.name	Категорія запису
title	TEXT	NOT NULL	Заголовок інформаційного блоку
content	TEXT	NOT NULL	Повний текст інформаційного блоку
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час останнього оновлення запису

Таблиця *settings* реалізована за патерном «ключ-значення» (key-value store). Первинним ключем є текстове поле key, що унеможливлює дублювання налаштувань. Поточна схема містить два обов'язкових записи: telegram_url — посилання на Telegram адміністратора, яке підставляється в банер ескалації на клієнтській стороні, та school_name — повна назва школи. Поле updated_at фіксує час останньої зміни кожного параметру.

Таблиця 2.6 – Специфікація таблиці settings

Назва поля	Тип даних	Обмеження	Опис
key	TEXT	PRIMARY KEY	Унікальний текстовий ключ налаштування
value	TEXT	NOT NULL	Значення налаштування
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час останнього оновлення

2.2.2 Розробка концептуальної схеми та ER-моделі даних

Для графічного відображення структури бази даних та зв'язків між її сутностями розроблено концептуальну схему у вигляді ER-діаграми в нотації Crow's Foot Notation.

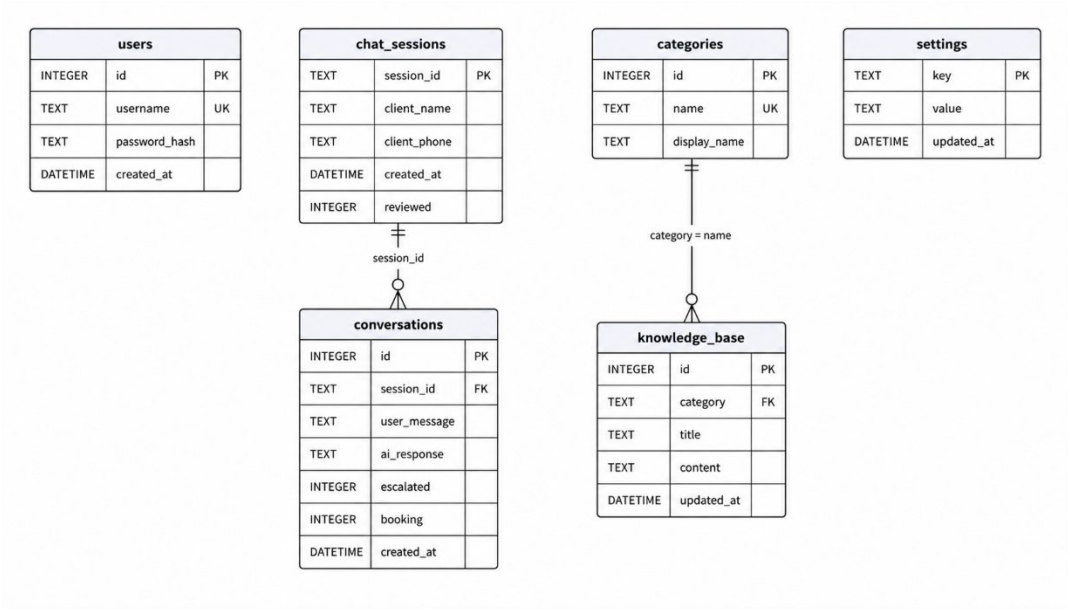

Рисунок 2.5 – ER-діаграма: схема бази даних системи

На діаграмі виділено шість сутностей, що відповідають шести таблицям бази даних. Між ними існує два зв'язки типу «один-до-багатьох» (1:N).

Перший зв'язок — між chat_sessions та conversations: одна чат-сесія (chat_sessions.session_id) може містити довільну кількість записів у таблиці conversations (поле conversations.session_id є зовнішнім ключем). Цей зв'язок є структурним хребтом системи ведення журналу — він дозволяє групувати всі повідомлення одного відвідувача в межах однієї сесії та відображати їх у вигляді єдиного транскрипту в адмін-панелі.

Другий зв'язок — між categories та knowledge_base: одна категорія (categories.name) може об'єднувати довільну кількість інформаційних записів (knowledge_base.category є зовнішнім ключем, що посилається на categories.name). Завдяки цьому зв'язку адміністратор може структурувати базу знань за тематичними розділами (курси, ціни, викладачі, загальна інформація) і при необхідності додавати нові категорії без змін у схемі бази даних.

Таблиці users та settings є незалежними сутностями без зовнішніх ключів до інших таблиць. users ізольована логічно — вона обслуговує виключно механізм автентифікації адміністратора і не перетинається зі структурою даних чату. settings функціонує як глобальне сховище конфігурації, значення з якого зчитуються

сервером при обробці кожного запиту чату (зокрема, поле telegram_url включається у відповідь сервера при виявленні ескалації).

Така схема бази даних забезпечує нормалізацію даних до третьої нормальної форми (3NF): усувається надлишковість, кожен факт зберігається рівно в одному місці, а цілісність даних підтримується через зовнішні ключі та обмеження NOT NULL і UNIQUE.

2.3 Програмна реалізація серверної частини (Backend)

Серверна частина є центральним елементом розроблюваної вебсистеми, оскільки саме вона забезпечує обробку запитів користувачів, взаємодію з базою даних, авторизацію адміністратора, збереження історії діалогів та інтеграцію з AI API. Backend виконує роль проміжного рівня між клієнтським інтерфейсом, сховищем даних і зовнішнім сервісом штучного інтелекту.

У цьому підрозділі розглядається структура серверного середовища, підключення необхідних модулів, організація маршрутів API, реалізація контролерів, робота з базою даних та механізм автентифікації. Окрему увагу приділено інтеграції з AI-провайдером, оскільки саме цей компонент забезпечує формування автоматизованих відповідей на звернення клієнтів мовної школи.

2.3.1 Конфігурація серверного середовища

Серверна частина системи реалізована на платформі Node.js — середовищі виконання JavaScript на стороні сервера, що базується на рушії V8. Node.js забезпечує асинхронну, неблокуючу модель вводу-виводу, що робить його оптимальним вибором для додатків, які часто виконують мережеві запити та операції з базою даних.

Проєкт організований за модульною структурою, де кожна директорія має чітко визначене призначення.

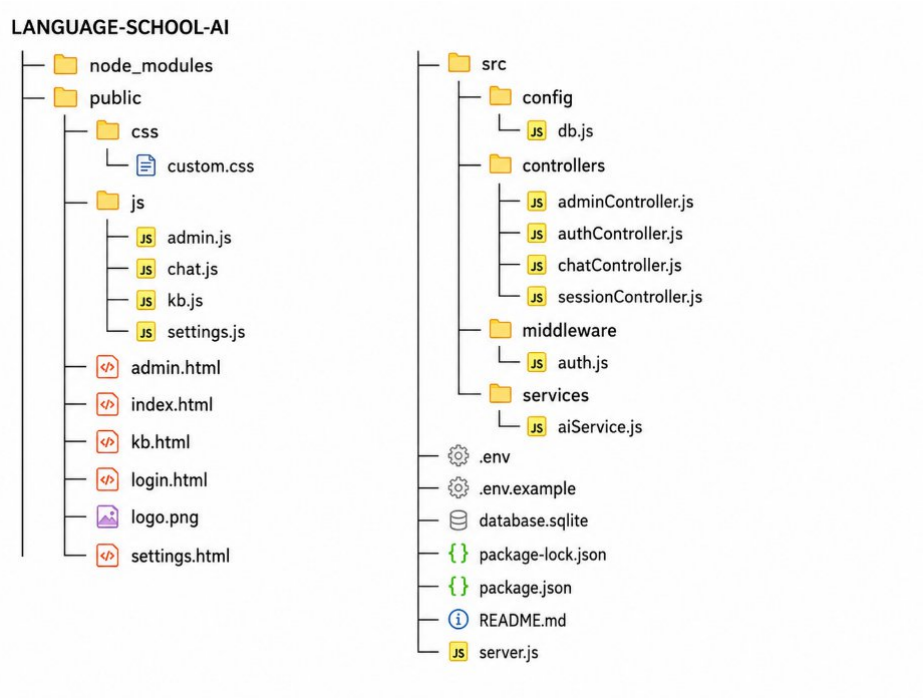

Рисунок 2.6 – Структура каталогів проєкту

Клієнтська частина розміщена у директорії public/, що містить підпапки css/ та js/ для стилів і скриптів інтерфейсу, а також п'ять HTML-сторінок: index.html, admin.html, login.html, kb.html та settings.html. Серверна логіка зосереджена у директорії src/, яка поділена на чотири підмодулі: config/ (ініціалізація бази даних), controllers/ (бізнес-логіка — adminController.js, authController.js, chatController.js, sessionController.js), middleware/ (JWT-автентифікація — auth.js) та services/ (AI-інтеграція — aiService.js). У кореневій директорії знаходяться server.js (точка входу), package.json, database.sqlite, а також конфігураційні файли .env та .env.example.

Залежності проєкту описані у файлі package.json. У розділі dependencies перелічено сім виробничих пакетів: express (HTTP-маршрутизація), sqlite та sqlite3 (Promise-based API для SQLite), bcryptjs (хешування паролів), jsonwebtoken (JWT-автентифікація), cookie-parser (зчитування cookie), dotenv (змінні середовища) та openai (SDK для AI API).

```
{
  "name": "language-school-ai",
  "version": "1.0.0",
  "description": "AI-assisted customer support system for a language school",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "node --watch server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cookie-parser": "^1.4.6",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "openai": "^4.52.0",
    "sqlite": "^5.1.1",
    "sqlite3": "^5.1.7"
  }
}
```

Рисунок 2.7 – Лістинг package.json: конфігурація проєкту

Для запуску в режимі розробки використовується скрипт node --watch server.js, що забезпечує автоматичне перезавантаження сервера при зміні файлів без встановлення сторонніх інструментів. Конфігурація сервера передбачає відокремлення публічних та захищених маршрутів: ендпоінт POST /api/chat є публічним і доступний без автентифікації, тоді як усі адміністративні маршрути (/api/admin/*) захищені middleware verifyToken.

```
const app = express();

// Підключення middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// Публічні маршрути
app.post('/api/chat',       chatController.handleChat);
app.post('/api/auth/login', authController.login);
app.post('/api/auth/logout', authController.logout);

// Захищені маршрути адміністратора (вимагають JWT)
app.get('/api/admin/knowledge',    verifyToken, adminController.getKnowledge);
app.post('/api/admin/knowledge',   verifyToken,
adminController.createKnowledge);
app.put('/api/admin/knowledge/:id',verifyToken,
adminController.updateKnowledge);
app.delete('/api/admin/knowledge/:id', verifyToken,
adminController.deleteKnowledge);
app.get('/api/admin/logs',         verifyToken, adminController.getLogs);
app.get('/api/admin/stats',        verifyToken, adminController.getStats);
app.put('/api/admin/settings',     verifyToken,
adminController.updateSettings);
```

Рисунок 2.8 – Лістинг server.js: підключення middleware та реєстрація маршрутів API

Запуск сервера реалізований через асинхронну функцію startServer(). Вона спочатку викликає initDb() для ініціалізації бази даних і лише після успішного завершення починає прослуховувати порт 3000 або значення змінної PORT. Такий підхід гарантує, що сервер не прийматиме запити до моменту повної готовності схеми БД.

2.3.2 Розробка контролерів взаємодії з компонентами

Уся логіка ініціалізації та взаємодії з базою даних зосереджена у файлі src/config/db.js. Для забезпечення сумісності між різними версіями схеми в системі реалізовано механізм автоматичної міграції: при кожному запуску сервера функція initDb() перевіряє наявність необхідних колонок і, якщо їх немає, додає їх

командою ALTER TABLE. Це дозволяє безпечно оновлювати схему без ризику втрати накопичених даних.

```
async function initDb() {
  const db = await getDbConnection();

  // Створення таблиць (ідемпотентна операція IF NOT EXISTS)
  await db.exec(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      username TEXT UNIQUE NOT NULL,
      password_hash TEXT NOT NULL,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );
    CREATE TABLE IF NOT EXISTS conversations (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      session_id TEXT NOT NULL,
      user_message TEXT NOT NULL,
      ai_response TEXT NOT NULL,
      escalated INTEGER DEFAULT 0,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );
  `);

  // Автоматична міграція: безпечне додавання нових колонок
  const migrations = [
    `ALTER TABLE conversations ADD COLUMN booking INTEGER DEFAULT 0`,
    `ALTER TABLE chat_sessions ADD COLUMN client_name TEXT`,
    `ALTER TABLE chat_sessions ADD COLUMN client_phone TEXT`
  ];
  for (const sql of migrations) {
    try {
      await db.exec(sql);
    } catch (e) {
      // Колонка вже існує — SQLite повертає помилку, ігноруємо
    }
  }

  await db.close();
}
```

Рисунок 2.9 – Лістинг src/config/db.js: ініціалізація таблиць та механізм автоматичної міграції

Конструкція CREATE TABLE IF NOT EXISTS гарантує ідемпотентність — повторний запуск сервера не спричиняє помилок і не перезаписує дані. Масив SQL-команд у циклі for...of із блоком try/catch дозволяє безпечно виконувати міграції: якщо колонка вже існує — SQLite кидає виняток, який ігнорується; якщо відсутня — додається з відповідним DEFAULT.

Автентифікація реалізована у файлі src/controllers/authController.js. При вході система порівнює пароль з хешем у БД через bcrypt.compare(). У разі успіху генерується JWT-токен і встановлюється у HTTPOnly-cookie браузера.

```
async function login(req, res) {
  const { username, password } = req.body;

  const db   = await getDbConnection();
  const user = await db.get(
    'SELECT * FROM users WHERE username = ?', [username]
  );
  await db.close();

  if (!user || !(await bcrypt.compare(password, user.password_hash))) {
    return res.status(401).json({ error: 'Неправильний логін або пароль.' });
  }

  const token = jwt.sign(
    { id: user.id, username: user.username },
    JWT_SECRET,
    { expiresIn: '1d' }
  );

  res.cookie('token', token, {
    httpOnly: true,     // захист від XSS-атак
    sameSite: 'strict', // захист від CSRF-атак
    maxAge: 24 * 60 * 60 * 1000  // 1 доба
  });

  return res.json({ success: true, user: { username: user.username } });
}
```

Рисунок 2.10 – Лістинг src/controllers/authController.js: перевірка пароля та видача JWT

Прапорець httpOnly: true забороняє доступ до cookie через JavaScript — це захищає JWT від XSS-атак. Параметр sameSite: 'strict' обмежує відправку cookie лише для запитів з того самого домену, що запобігає CSRF. Захист маршрутів здійснюється middleware verifyToken (src/middleware/auth.js), яка зчитує токен з req.cookies.token та верифікує його функцією jwt.verify(). При невалідному або відсутньому токені повертається статус 401.

Для підрахунку статистики адмін-панелі контролер adminController.js виконує чотири SQL-запити: загальна кількість діалогів, кількість ескалацій, кількість записів у базі знань та кількість унікальних категорій.

```
async function getStats(req, res) {
  const db = await getDbConnection();

  const logsCount      = await db.get('SELECT COUNT(*) as count FROM conversations');
  const escalatedCount = await db.get(
    'SELECT COUNT(*) as count FROM conversations WHERE escalated = 1'
  );
  const kbCount        = await db.get('SELECT COUNT(*) as count FROM knowledge_base');
  const categoriesCount = await db.get(
    'SELECT COUNT(DISTINCT category) as count FROM knowledge_base'
  );
  await db.close();

  const escalationRate = logsCount.count > 0
    ? Math.round((escalatedCount.count / logsCount.count) * 100)
    : 0;

  return res.json({
    knowledgeItems:  kbCount.count,
    totalChats:      logsCount.count,
    escalatedChats:  escalatedCount.count,
    escalationRate:  escalationRate,
    categoriesCount: categoriesCount.count
  });
}
```

Рисунок 2.11 – Лістинг src/controllers/adminController.js: агрегатна статистика системи

2.3.3 Реалізація інтеграції з API провайдерів ШІ

Взаємодія з AI реалізована у сервісному шарі src/services/aiService.js. Сервіс використовує бібліотеку openai (v4), налаштовану на роботу з провайдером OpenRouter — агрегатором AI-моделей із OpenAI-сумісним API-форматом [13]. Це надає гнучкість: підключення будь-якої підтримуваної моделі (GPT-4o, Gemini Flash, Claude Sonnet тощо) потребує лише зміни змінної середовища OPENAI_MODEL без жодних змін у коді.

Сервіс підтримує два режими роботи. Якщо OPENAI_API_KEY відсутній або містить значення-заглушку, система автоматично переходить у Mock-режим, де відповіді генеруються локально пошуком за ключовими словами. Це дозволяє розгортати та демонструвати систему без активного ключа.

```
const { OpenAI } = require('openai');

const apiKey     = process.env.OPENAI_API_KEY;
const model      = process.env.OPENAI_MODEL || 'gpt-4o-mini';
const isMockMode = !apiKey || apiKey === 'your_openai_api_key_here';

let openaiClient = null;
if (!isMockMode) {
  openaiClient = new OpenAI({ apiKey });
  console.log(`[AI Service] Configured with model "${model}"`);
} else {
  console.warn('[AI Service] Running in Mock Mode (no API key).');
}
```

Рисунок 2.12 – Лістинг src/services/aiService.js: ініціалізація клієнта OpenRouter та перевірка режиму

Центральною функцією сервісу є generateResponse(). Вона будує системний промпт із вбудованою базою знань та правилами поведінки, після чого виконує запит до AI API. Системний промпт є критичним компонентом архітектури — він задає роль бота та механізм виявлення намірів через спеціальні маркери [BOOKING] та [ESCALATE].

```
Ти — ввічливий штучний інтелект-асистент мовної школи MTSchool
(Mother Tongue School). Твоє завдання — відповідати на запитання
клієнтів, використовуючи виключно наданий контекст бази знань.

Важливі правила:
1. Базуй відповіді виключно на наданій базі знань.
   Не вигадуй інформації, якої там немає (ціни, дати, імена).
2. Якщо у базі знань немає відповіді, або користувач просить
   покликати людину / адміністратора, ти повинен:
   - Ввічливо повідомити, що готовий передати діалог людині.
   - Додати в кінці повідомлення маркер [ESCALATE].
3. Якщо користувач хоче записатися на курс або пробне заняття,
   додай в кінці повідомлення маркер [BOOKING].
4. Спілкуйся виключно українською мовою.
   Будь професійним, привітним та лаконічним.

База знань школи:
${contextText}
```

Рисунок 2.13 – Системна інструкція (System Prompt) для нейромережі

Після отримання відповіді від AI-моделі функція виконує парсинг маркерів. Перевіряється наявність [BOOKING], потім [ESCALATE]. При виявленні кожного

з них встановлюється відповідний булевий прапорець, а маркер видаляється з тексту перед поверненням клієнту.

```
const completion = await openaiClient.chat.completions.create({
  model: model,
  messages: [
    { role: 'system', content: systemPrompt },
    { role: 'user',   content: userMessage  }
  ],
  temperature: 0.2  // низька варіативність — менше "галюцинацій"
});

const aiText = completion.choices[0].message.content || '';

// Парсинг маркерів намірів
const isBooking = aiText.includes('[BOOKING]');
const escalated = aiText.includes('[ESCALATE]') ||
                  /зв'язатися з адміністратором/i.test(aiText);

// Очищення тексту від службових маркерів перед відправкою
const cleanedText = aiText
  .replace('[BOOKING]', '')
  .replace('[ESCALATE]', '')
  .trim();

return { text: cleanedText, escalated, isBooking };
```

Рисунок 2.14 – Лістинг src/services/aiService.js: виклик API та парсинг маркерів намірів

Параметр temperature: 0.2 встановлено навмисно низьким для обмеження «творчості» моделі та мінімізації галюцинацій. Значення 0.0 дало б абсолютно детерміновані відповіді, а 1.0 — максимальну варіативність; 0.2 є оптимальним балансом між стабільністю та природністю мови для консультаційного бота.

У разі помилки мережі або перевищення rate limit реалізовано автоматичний graceful fallback: система перехоплює виняток, логує деталі та викликає локальну функцію searchMockKnowledge(), яка виконує пошук за ключовими словами серед записів бази знань. Така поведінка гарантує безперервну роботу чат-сервісу навіть при тимчасовій недоступності зовнішнього AI-провайдера.

2.4 Розробка клієнтського інтерфейсу (Frontend)

Клієнтський інтерфейс забезпечує безпосередню взаємодію користувача із системою, тому його зручність, зрозумілість і швидкість роботи мають важливе значення для практичного використання вебзастосунку. У розроблюваній системі frontend-частина включає два основні напрями: інтерфейс AI-чату для клієнта мовної школи та адміністративну панель для менеджера або адміністратора.

Під час розробки клієнтської частини було враховано необхідність швидкого доступу до основних функцій, адаптивність інтерфейсу для різних пристроїв, динамічне оновлення даних без перезавантаження сторінки та візуальне розділення клієнтської й адміністративної функціональності. У цьому підрозділі описано створення інтерактивного чату, реалізацію адміністративної панелі та організацію асинхронного обміну даними між frontend і backend за допомогою Fetch API.

2.4.1 Створення інтерактивного AI-чату з формою реєстрації

Головна сторінка (public/index.html) є основним інтерфейсом взаємодії відвідувача з системою. Сторінка реалізована на основі компонентної структури з двома головними секціями, розміщеними у дванадцятиколонній сітці CSS Grid фреймворку TailwindCSS. Ліва секція містить маркетингову інформацію: бейдж «Штучний інтелект на зв'язку 24/7» з анімованим індикатором активності, заголовок <h1> з градієнтним текстом та бенто-грид з чотирма інформаційними плитками. Права секція (7 колонок) містить інтерфейс AI-чату у вигляді скляної панелі (glassmorphism).

Перед початком чату система відображає форму пре-чат реєстрації (#prechatModal), яка накладається поверх інтерфейсу з ефектом розмиття фону (backdrop-blur). Форма містить поля client_name та client_phone з валідацією на стороні клієнта: перевіряється непорожність імені та відповідність телефону регулярному виразу. Лише після успішної валідації та відправки даних на сервер модальне вікно закривається і активується поле введення повідомлення.

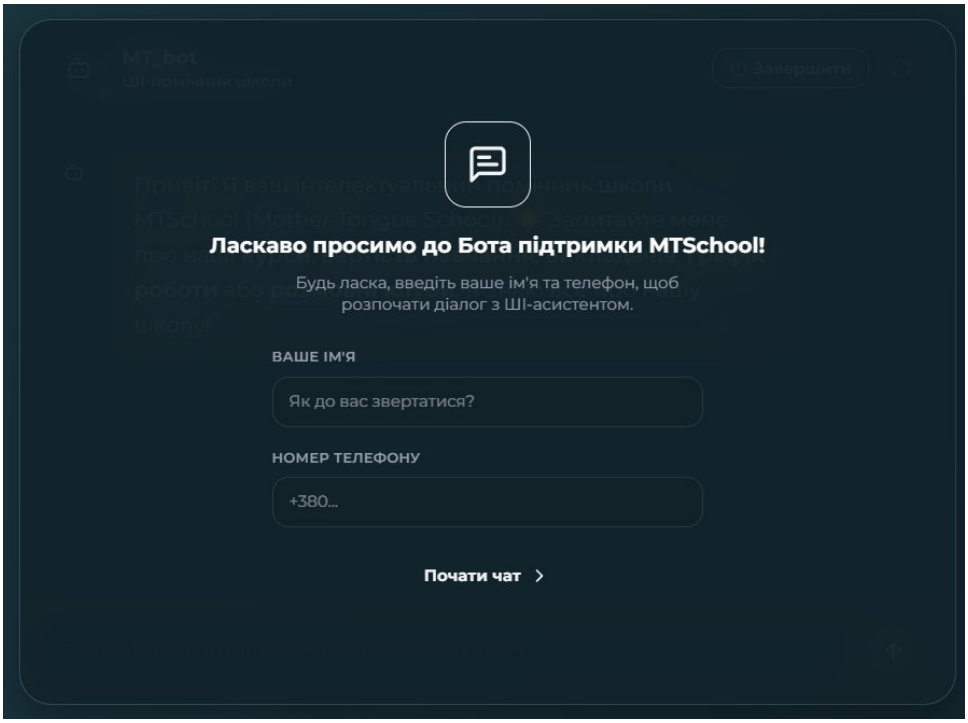

Рисунок 2.15 - Форма пре-чат реєстрації

```
function appendMessage(sender, text, isMarkdown = true) {
  const isBot = sender === 'bot';
  const bubbleWrapper = document.createElement('div');
  bubbleWrapper.className =
    `flex items-start gap-3 max-w-[85%]
     ${isBot ? '' : 'ml-auto flex-row-reverse animate-fade-in-up'}`;

  const iconDiv = document.createElement('div');
  iconDiv.className = `w-8 h-8 rounded-lg flex items-center justify-center
    flex-shrink-0 border border-white/10
    ${isBot ? 'bg-school-teal/50' : 'bg-school-gold/20'}`;
  iconDiv.innerHTML = isBot
    ? '<i data-lucide="bot"  class="w-4 h-4 text-school-gold"></i>'
    : '<i data-lucide="user" class="w-4 h-4 text-school-gold"></i>';

  const contentDiv = document.createElement('div');
  contentDiv.className = isBot
    ? 'bg-white/5 border border-white/5 px-4 py-3 rounded-2xl rounded-tl-none'
    : 'bg-school-teal text-white px-4 py-3 rounded-2xl rounded-tr-none';

  contentDiv.innerHTML = isMarkdown ? formatMessage(text) : text;

  bubbleWrapper.appendChild(iconDiv);
  bubbleWrapper.appendChild(contentDiv);
  chatMessages.appendChild(bubbleWrapper);
  scrollToBottom();
}
```

Рисунок 2.16 – Лістинг public/js/chat.js: функція appendMessage()

Блоки повідомлень генеруються динамічно функцією appendMessage() у файлі public/js/chat.js. Повідомлення бота відображаються ліворуч з іконкою робота (lucide-bot), повідомлення користувача — праворуч з іконкою профілю (lucide-user). Функція formatMessage() виконує санітизацію HTML-спецсимволів та перетворює Markdown-розмітку на теги .

Блоки користувача отримують клас ml-auto flex-row-reverse, що відштовхує їх до правого краю, та анімацію animate-fade-in-up для плавного появлення. Блоки бота мають скляний фон bg-white/5 зі злегка округленим верхнім лівим кутом (rounded-tl-none), що відтворює класичний вигляд чат-месенджера. Під час очікування відповіді від сервера функція showTypingLoader() додає анімований індикатор — три точки з CSS-анімацією bounce, що формує відчуття живої взаємодії.

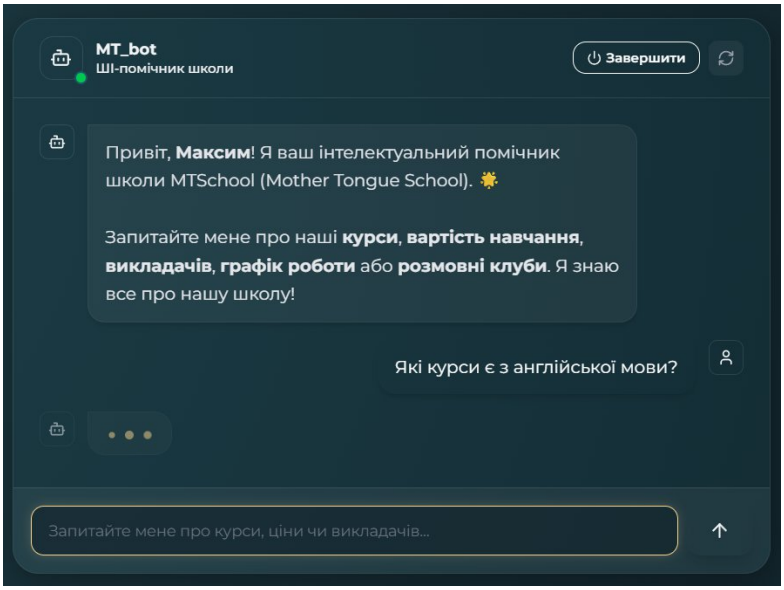

Рисунок 2.17 – Вікно чату з діалогом

Унікальний ідентифікатор сесії генерується один раз при першому відвідуванні та зберігається у localStorage браузера. При наступних відвідуваннях sessionId відновлюється з сховища — це дозволяє серверу прив'язати всі повідомлення одного відвідувача до єдиного запису в таблиці chat_sessions. Якщо сервер повертає escalated: true, клієнтський JavaScript знімає клас hidden з блоку #escalationBanner та підставляє URL Telegram-адміністратора.

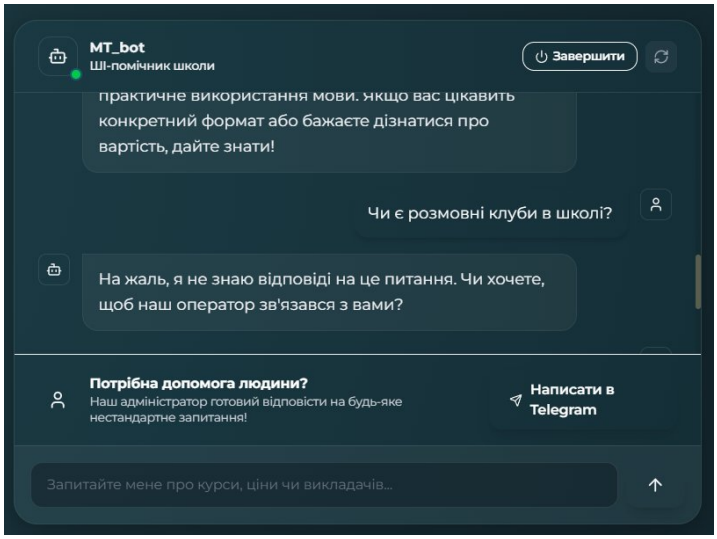

Рисунок 2.18 – Банер з кнопкою «Написати в Telegram»

2.4.2 Реалізація адмін-панелі

Адміністративна панель системи реалізована як єдина сторінка (public/admin.html) з чотирма логічними секціями: дашборд статистики, керування базою знань, журнал діалогів та налаштування. Інтерфейс виконаний у стилі Glassmorphism — усі картки та контейнери мають напівпрозорий фон (bg-white/5), тонку рамку (border-white/10) та ефект розмиття (backdrop-blur).

При завантаженні сторінки скрипт admin.js виконує перевірку автентифікації через GET /api/auth/me. У разі відсутності активної сесії — автоматичне перенаправлення на /login.html.

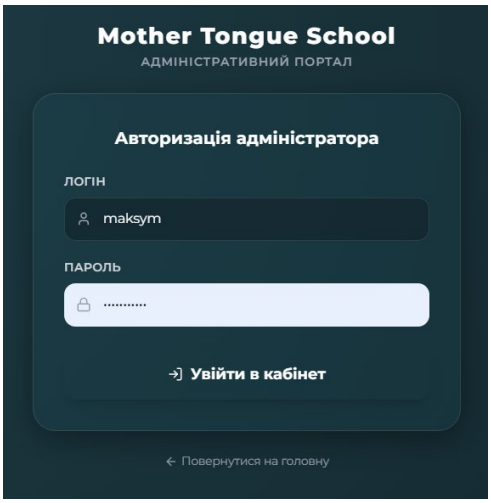

Рисунок 2.19 – Вікно авторизації для адміністратора

Після успішної верифікації паралельно завантажуються всі дані через функцію loadAllData(), яка послідовно викликає loadStats(), loadKnowledge(), loadSettings() та loadLogs().

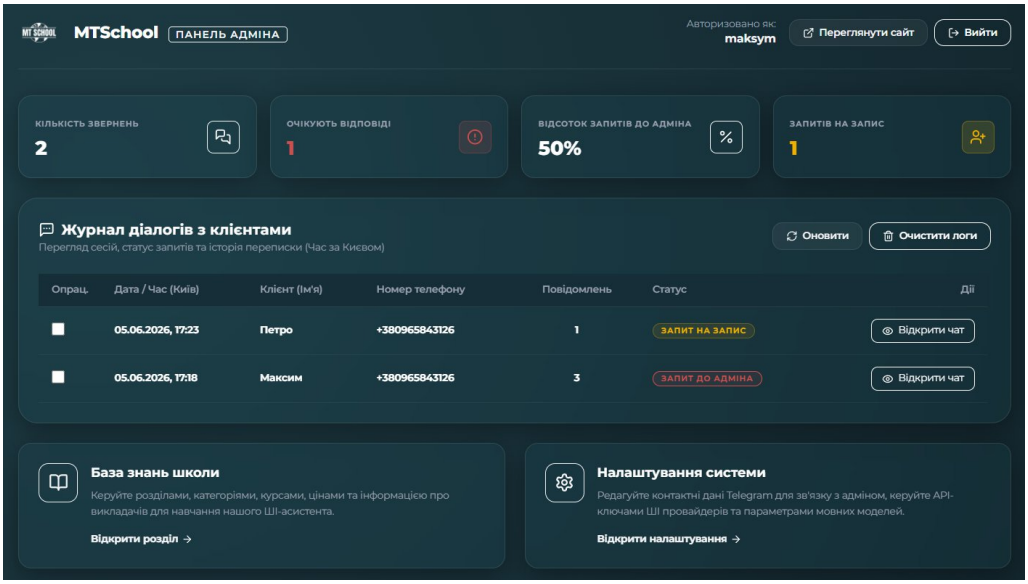

Рисунок 2.20 – Головний дашборд адміністратора

Верхній рядок містить чотири статистичні картки у стилі Bento Grid: «Кількість звернень», «Очікують відповіді», «Відсоток звернень до адміна» та «Кількість запитів на зипис» — остання підсвічена жовтим акцентним кольором. Таблиця журналу відображає рядки із кольоровими бейджами статусів: зелений «Бот» або червоний «Запит до адміна» та жовтий «Запит на запис»

```
async function loadStats() {
  try {
    const res = await fetch('/api/admin/stats');
    if (res.ok) {
      const stats = await res.json();
      statKbCount.textContent        = stats.knowledgeItems;
      statTotalChats.textContent     = stats.totalChats;
      statEscalatedChats.textContent = stats.escalatedChats;
      statEscalationRate.textContent = `${stats.escalationRate}%`;
    }
  } catch (err) {
    console.error('[Stats] Error loading statistics:', err);
  }
}
```

Рисунок 2.21 – Лістинг public/js/admin.js: функція loadStats()

Секція керування базою знань містить таблицю з усіма записами, полем пошуку та фільтром за категорією. Фільтрація виконується локально в браузері без додаткових серверних запитів — функція renderKnowledgeTable() фільтрує масив knowledgeItems за введеним текстом та обраною категорією.

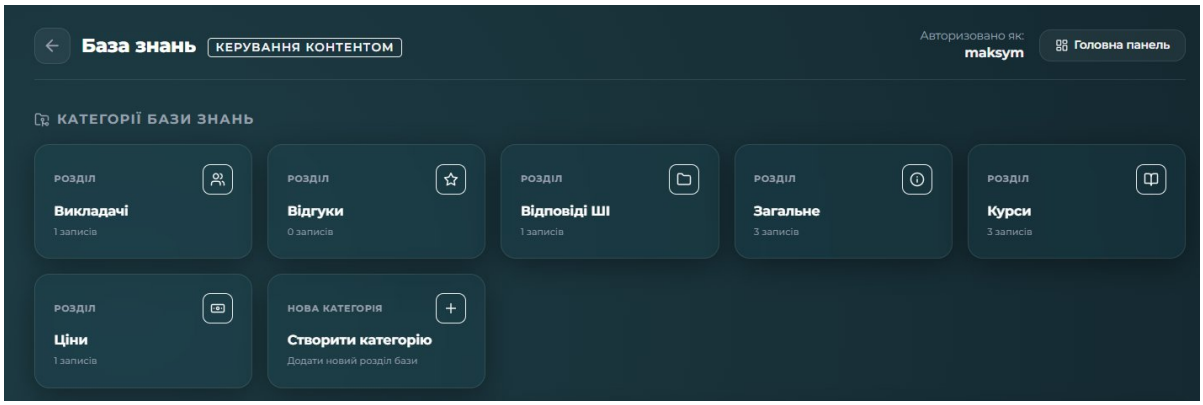

Рисунок 2.22 – Сторінка керування базою знань

```
kbTableBody.innerHTML = filtered.map(item => {
  let badgeClass = 'bg-white/5 border-white/10 text-gray-400';
  if (item.category === 'courses')
    badgeClass = 'bg-school-teal/20 border-school-teal/30 text-school-gold';
  if (item.category === 'prices')
    badgeClass = 'bg-green-500/10 border-green-500/20 text-green-400';
  if (item.category === 'teachers')
    badgeClass = 'bg-purple-500/10 border-purple-500/20 text-purple-400';

  const shortContent = item.content.length > 90
    ? item.content.substring(0, 90) + '...'
    : item.content;

  return `
    <tr class="hover:bg-white/5 border-b border-white/5 transition-all">
      <td class="py-3.5 px-4">
        <span class="px-2.5 py-1 rounded-full border text-[10px]
                     font-bold uppercase ${badgeClass}">
          ${item.category}
        </span>
      </td>
      <td class="py-3.5 px-4 font-semibold text-white">${item.title}</td>
      <td class="py-3.5 px-4 text-gray-400">${shortContent}</td>
      <td class="py-3.5 px-4 text-right">
        <button onclick="editKbItem(${item.id})" title="Редагувати">
          <i data-lucide="edit-2" class="w-3.5 h-3.5 text-school-gold"></i>
        </button>
        <button onclick="deleteKbItem(${item.id})" title="Видалити">
          <i data-lucide="trash-2" class="w-3.5 h-3.5 text-school-coral"></i>
        </button>
      </td>
    </tr>`;
}).join('');
```

Рисунок 2.23 – Лістинг public/js/admin.js

Форма створення/редагування реалізована з динамічним перемиканням режимів. При натисканні «Редагувати» функція editKbItem(id) знаходить запис у масиві knowledgeItems та заповнює поля форми.

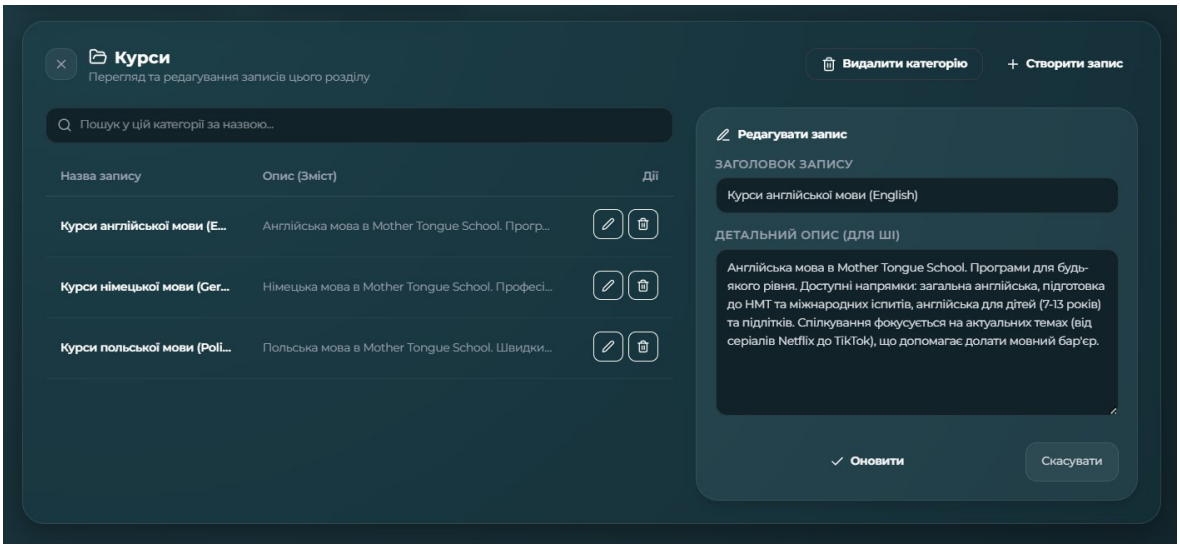

Рисунок 2.24 – Блок редагування бази знань

Журнал діалогів (logsContainer) відображає останні 100 записів із таблиці conversations у двоколонковому макеті пар «запитання / відповідь» з кольоровими бейджами статусів.

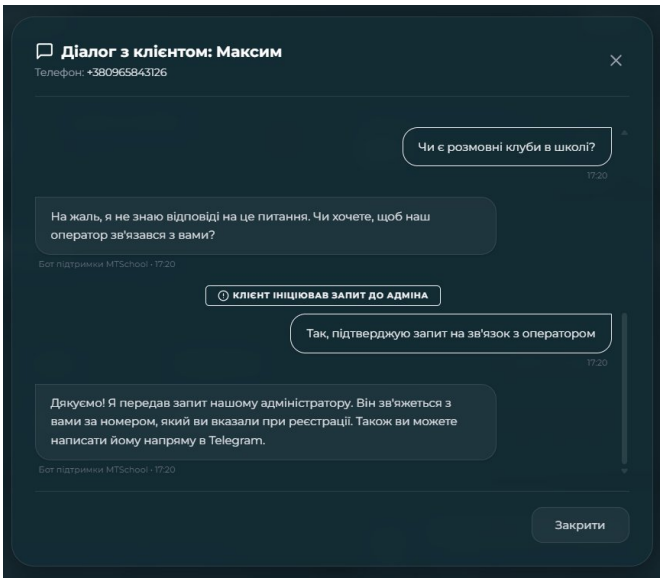

Рисунок 2.25 – Модальне вікно перегляду повної історії чату конкретного клієнта

Усі дії в адмін-панелі супроводжуються Toast-сповіщеннями — плашками, що з'являються знизу екрана з CSS-анімацією translate-y і автоматично зникають через 3 секунди. Toast має три стани: success, error та info.

2.4.3 Реалізація асинхронного обміну даними (Fetch API)

Уся клієнтська взаємодія з сервером реалізована через браузерний Fetch API без перезавантаження сторінки. Fetch API є сучасним стандартом для виконання асинхронних HTTP-запитів у JavaScript, що прийшов на зміну застарілому XMLHttpRequest. Усі запити у системі використовують синтаксис async/await, що забезпечує неблокуючу роботу інтерфейсу під час очікування відповіді сервера та підтримує зручну обробку помилок через try/catch.

```
  try {
    const response = await fetch('/api/chat', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        message:   query,
        sessionId: sessionId
      })
    });

    const data = await response.json();
    hideTypingLoader();

    if (response.ok) {
      appendMessage('bot', data.response);

      // Якщо сервер повернув прапорець ескалації — показуємо банер
      if (data.escalated) {
        telegramEscalateLink.href = data.telegramUrl || '#';
        escalationBanner.classList.remove('hidden');
        escalationBanner.classList.add('flex');
        scrollToBottom();
      }
    } else {
      appendMessage('bot', `Помилка: ${data.error || 'Спробуйте пізніше'}.`);
    }

  } catch (err) {
    hideTypingLoader();
    appendMessage('bot', 'Не вдалося підключитися до сервера.');
  }
});
```

Рисунок 2.26 – Лістинг public/js/chat.js: надсилання повідомлення через Fetch API

Метод e.preventDefault() запобігає стандартній поведінці браузера — перезавантаженню сторінки при відправці форми. Запит надсилається методом POST з тілом у форматі JSON, що включає текст повідомлення та sessionId з localStorage. Відповідь від сервера парсується як JSON функцією response.json().

Блок try/catch забезпечує обробку мережевих помилок — у разі недоступності сервера користувач бачить відповідне повідомлення в інтерфейсі.

Для адміністративних операцій з базою знань реалізований CRUD через Fetch API з підтримкою чотирьох HTTP-методів: GET (завантаження), POST (створення), PUT (оновлення) та DELETE (видалення). Вибір методу та URL залежить від стану форми: якщо formKbId заповнений — виконується оновлення (PUT), якщо порожній — створення (POST).

```
kbForm.addEventListener('submit', async (e) => {
  e.preventDefault();

  const id = formKbId.value;
  const payload = {
    category: formCategory.value,
    title:    formTitleInput.value.trim(),
    content:  formContent.value.trim()
  };

  // Динамічно обираємо URL та метод залежно від режиму форми
  const url    = id ? `/api/admin/knowledge/${id}` : '/api/admin/knowledge';
  const method = id ? 'PUT' : 'POST';

  try {
    const res = await fetch(url, {
      method: method,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    const data = await res.json();

    if (res.ok) {
      showToast(id ? 'Запис оновлено успішно!' : 'Запис створено успішно!');
      clearForm();
      loadKnowledge();  // Оновлення таблиці без перезавантаження
      loadStats();      // Оновлення лічильників статистики
    } else {
      showToast(data.error || 'Помилка виконання операції.', 'error');
    }
  } catch (err) {
    showToast('Не вдалося зберегти запис. Перевірте з\'єднання.', 'error');
  }
});
```

Рисунок 2.27 – Лістинг public/js/admin.js: CRUD-операції бази знань через Fetch API

Після кожної успішної операції автоматично викликаються loadKnowledge() та loadStats() — це оновлює таблицю та лічильники без перезавантаження сторінки. Такий підхід забезпечує миттєвий зворотний зв'язок для адміністратора та знижує

навантаження на сервер, оскільки між клієнтом і сервером передається виключно JSON, а не повний HTML-документ.

2.5 Тестування, верифікація та аналіз ефективності системи

Після завершення програмної реалізації необхідно перевірити коректність роботи основних компонентів системи та відповідність отриманого результату поставленим вимогам. Тестування дозволяє виявити можливі помилки в логіці роботи серверної частини, структурі бази даних, обробці AI-запитів, функціонуванні адміністративної панелі та відображенні клієнтського інтерфейсу.

У межах цього підрозділу виконано перевірку цілісності схеми бази даних і механізму міграцій, тестування точності обробки запитів AI-модулем, а також функціональне тестування інтерфейсу користувача й адаптивності сторінок. Такий підхід дає змогу оцінити не лише працездатність окремих модулів, а й загальну ефективність системи як цілісного програмного рішення.

2.5.1 Тестування цілісності схеми бази даних та міграцій

Для забезпечення надійності зберігання даних було проведено тестування ініціалізації та автоматичної міграції схеми бази даних SQLite. Метою тестування була перевірка коректності створення всіх таблиць (users, chat_sessions, conversations, knowledge_base, settings) при першому запуску сервера, а також безпечного додавання нових стовпців до існуючих таблиць без втрати даних.

Під час тестування сервер було запущено на порожній директорії без файлу database.sqlite. Логи підтвердили успішне виконання команд CREATE TABLE IF NOT EXISTS та початковий seeding таблиці settings базовими конфігураціями. Після цього в код було додано нові міграції для таблиці conversations (поле booking) та chat_sessions (поля client_name, client_phone). Повторний запуск сервера успішно ініціював процес автоматичної міграції через конструкцію ALTER TABLE.

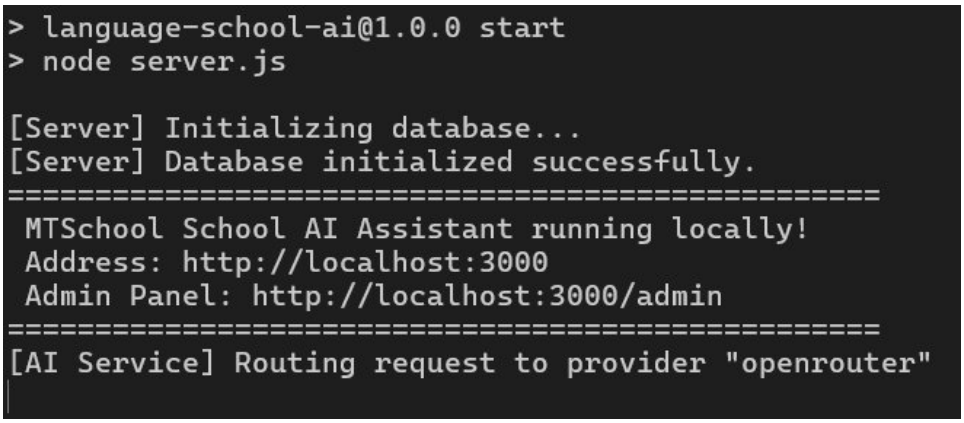

Рисунок 2.28 – Консоль запуску сервера Node.js

Механізм перехоплення помилок try/catch довів свою ефективність: при наступних перезапусках сервера спроби повторного додавання вже існуючих стовпців генерували винятки SQLite, які коректно ігнорувалися системою, не перериваючи роботу сервера. Це гарантує стабільність розгортання оновлень у виробничому середовищі.

2.5.2 Тестування точності обробки ШІ-запитів

Основним завданням інтелектуального модуля системи є не лише надання відповідей на основі бази знань, але й безпомилкове розпізнавання намірів користувача: бажання записатися на курс (встановлення прапорця booking) або потреба зв'язатися з живим адміністратором (встановлення прапорця escalated).

Для автоматизації перевірки бази даних було розроблено скрипт test_flow.js, який імітує відправку повідомлення клієнтом, а потім безпосередньо підключається до файлу database.sqlite та виконує SQL-запити SELECT для перевірки збереження значень booking та escalated у таблиці conversations.

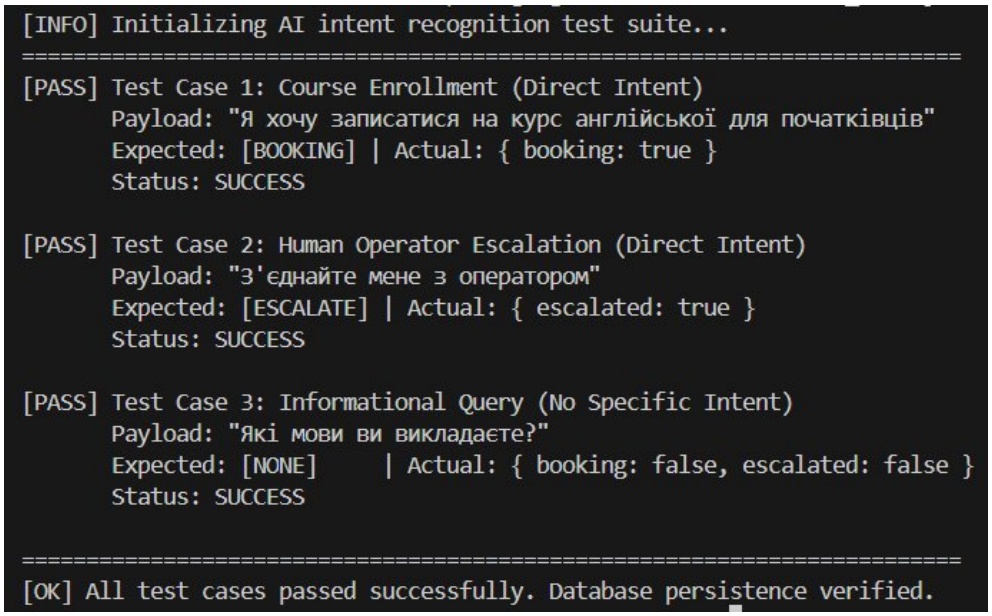

Рисунок 2.29 – Результати тестування розпізнавання намірів користувача

Результати тестування підтвердили, що системний промпт коректно налаштовує AI-модель на використання маркерів, а регулярні вирази на сервері успішно парсять ці маркери та перетворюють їх на логічні (boolean) значення бази даних.

2.5.3 Тестування UI та адаптивності

Клієнтський інтерфейс розроблявся з використанням підходу Mobile-First та утилітарних класів TailwindCSS, що дозволило забезпечити його адаптивність для пристроїв з різною діагоналлю екрана.

Тестування адаптивності проводилося за допомогою інструменту емуляції пристроїв у Google Chrome DevTools. Перевірялися три основні брейкпойнти: мобільні пристрої (ширина менше 768px), планшети (768px – 1024px) та десктопи (понад 1024px). На мобільних пристроях дванадцятиколонна сітка головної сторінки автоматично трансформується у послідовне відображення: маркетинговий блок з інформаційними картками розміщується зверху, а інтерфейс чату — під ним, займаючи 100% ширини екрана. Модальне вікно пре-чат реєстрації

також центрується та адаптується під вузькі екрани, зберігаючи ефект розмиття фону.

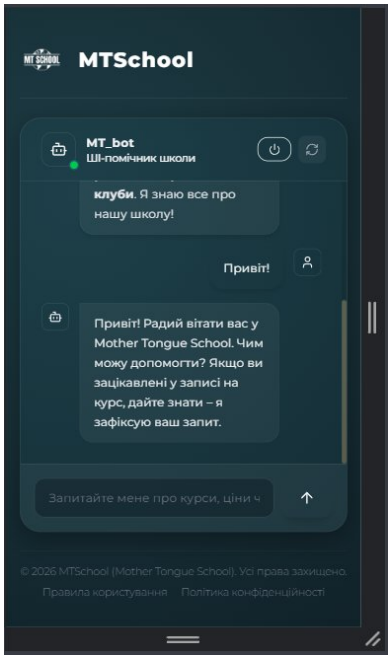

Рисунок 2.30 – Відображення чату на екранах мобільних пристроїв

В адміністративній панелі адаптивність реалізована через приховування менш важливих стовпців у таблиці діалогів на мобільних пристроях (класи hidden md:table-cell).

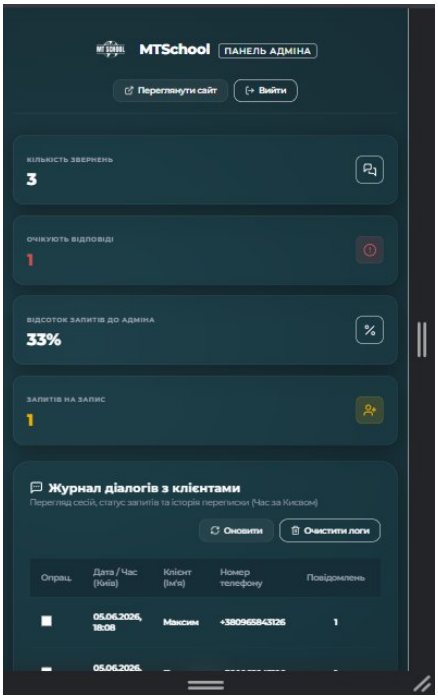

Рисунок 2.31 – Відображення адмін-панелі на екранах мобільних пристроїв

Картки статистики (Bento Grid) перебудовуються з чотирьох колонок в одну. Функціональне тестування в різних браузерах (Google Chrome, Mozilla Firefox, Safari) підтвердило повну кросбраузерну сумісність стилістики Glassmorphism: властивості backdrop-filter та напівпрозорі фони коректно відтворюються сучасними рушіями рендерингу. Усі AJAX-запити через Fetch API успішно оновлюють DOM-дерево без візуальних артефактів чи миготіння екрана.

Висновки до розділу

:

У другому розділі було спроєктовано та реалізовано вебсистему підтримки супроводу клієнтів мовної школи з використанням AI-асистента. Розроблено трирівневу архітектуру системи, структуру бази даних, серверну частину на Node.js та Express.js, клієнтський інтерфейс, адміністративну панель і модуль інтеграції з AI API. Проведене тестування підтвердило коректну роботу основних функцій системи: збереження діалогів, генерації відповідей, розпізнавання намірів користувача, переведення звернень до адміна та адаптивного відображення інтерфейсу.

ВИСНОВКИ

У ході дослідження було розроблено вебсистему підтримки супроводу клієнтів мовної школи з використанням AI-асистента. Результатом роботи стало програмне рішення, призначене для автоматизації первинної комунікації з клієнтами, збереження історії звернень, надання відповідей на типові запитання та передачі складних запитів адміністратору.

Під час виконання роботи було проаналізовано специфіку діяльності оналйн школи та визначено основні проблеми клієнтського супроводу: значну кількість повторюваних звернень, навантаження на менеджерів, ризик втрати повідомлень, відсутність централізованої історії діалогів та потребу в швидкому інформуванні клієнтів щодо курсів, вартості, графіку й умов навчання. Проведений аналіз підтвердив доцільність використання AI-асистента як інструмента підтримки менеджера, а не повної заміни людини.

Було розглянуто основні підходи до автоматизації клієнтської підтримки та сучасні сервіси AI-комунікації. Встановлено, що готові платформи мають широкий функціонал, однак не завжди враховують специфіку мовної школи, можуть бути надмірними за складністю та обмеженими щодо адаптації під конкретний навчальний заклад. Тому було обґрунтовано доцільність створення власної спеціалізованої системи з клієнтським чатом, базою знань, адміністративною панеллю та AI-модулем.

Для реалізації системи було обрано технологічний стек, який відповідає масштабу бакалаврської роботи та поставленим вимогам: Node.js і Express.js для серверної частини, HTML, CSS, TailwindCSS і JavaScript для клієнтського інтерфейсу, SQLite для збереження даних та OpenAI-сумісний API для генерації відповідей AI-асистента. Обрані технології дозволили реалізувати повноцінний вебзастосунок із клієнт-серверною архітектурою, REST API, базою даних, авторизацією адміністратора та інтеграцією із зовнішнім AI-сервісом.

У межах роботи було спроєктовано архітектуру системи, структуру бази даних, логіку взаємодії компонентів та алгоритм обробки повідомлень

користувача. Розроблена база даних забезпечує збереження облікових записів адміністратора, чат-сесій, історії діалогів, категорій, записів бази знань і службових налаштувань. Це дозволяє системі не лише відповідати на звернення клієнтів, а й накопичувати інформацію для подальшого аналізу та контролю якості обслуговування.

Реалізована серверна частина забезпечує приймання повідомлень від клієнта, формування контексту для AI-моделі, збереження результатів діалогу, авторизацію адміністратора та роботу з базою знань. Клієнтська частина містить інтерактивний AI-чат із формою попередньої реєстрації, а також адміністративну панель для перегляду статистики, журналу діалогів, редагування інформаційних записів і налаштувань системи.

Окрему увагу було приділено модулю штучного інтелекту. AI-асистент формує відповіді на основі бази знань мовної школи, що дає змогу уникати довільних або неперевірених відповідей. Реалізовано механізм розпізнавання намірів користувача, зокрема визначення бажання записатися на курс або необхідності передати звернення адміністратору. Такий підхід поєднує переваги автоматизації з можливістю людського контролю.

Проведене тестування підтвердило працездатність основних функцій системи: створення та збереження чат-сесій, запис історії діалогів, генерацію відповідей AI-асистентом, розпізнавання намірів користувача, переадресацію складних звернень, роботу адміністративної панелі та адаптивне відображення інтерфейсу. Отримані результати свідчать про те, що розроблена система відповідає поставленій меті та може бути використана як основа для автоматизації клієнтського супроводу мовної школи.

Практична цінність роботи полягає в тому, що створене рішення дозволяє зменшити навантаження на менеджера, прискорити обробку типових звернень, централізовано зберігати історію комунікації та підвищити якість взаємодії з потенційними клієнтами. Подальший розвиток системи може передбачати інтеграцію з месенджерами, календарем запису на пробні заняття, CRM-системою, платіжними сервісами та розширеними аналітичними звітами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Lyro — AI Customer Service Agent for Better CX : веб-сайт. URL: https://www.tidio.com/ai-agent/ (дата звернення: 01.03.2026).

2. Fin. The #1 AI Agent for customer service : веб-сайт. URL: https://fin.ai/ (дата звернення: 04.03.2026).

3. AI for Customer Service & Support. Zendesk AI Platform : веб-сайт. URL: https://www.zendesk.com/service/ai/ (дата звернення: 08.03.2026).

4. About Node.js. Node.js : веб-сайт. URL: https://nodejs.org/en/about (дата звернення: 12.03.2026).

5. Express.js — Node.js web application framework : веб-сайт. URL: https://expressjs.com/en/ (дата звернення: 16.03.2026).

6. HTML reference. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference (дата звернення: 20.03.2026).

7. CSS: Cascading Style Sheets. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 24.03.2026).

8. JavaScript Guide. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 29.03.2026).

9. Tailwind CSS — Rapidly build modern websites without ever leaving your HTML : веб-сайт. URL: https://tailwindcss.com/ (дата звернення: 03.04.2026).

10. Fetch API. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 08.04.2026).

11. SQLite Home Page : веб-сайт. URL: https://sqlite.org/ (дата звернення: 14.04.2026).

12. API Overview. OpenAI API Reference : веб-сайт. URL: https://developers.openai.com/api/reference/overview/ (дата звернення: 22.04.2026).

13. OpenRouter Quickstart Guide. Developer Documentation : веб-сайт. URL: https://openrouter.ai/docs/quickstart (дата звернення: 01.05.2026).

14. Саченко І. А. Проєктування інформаційних систем : конспект лекцій. Київ : КНУБА, 2024. 96 с.

15. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Книга 1. Організація баз даних та знань : підручник. 3-тє вид., стер. Львів : Магнолія 2006, 2024. 675 с.

16. Кушнерьов О. С., Вадько К. С., Кузченко А. О. Чат-бот на основі нейромережі для надання інформаційної підтримки бізнесу з позиції кібербезпеки. Сучасний захист інформації. 2024. № 4(60). С. 131–140. DOI: 10.31673/2409-7292.2024.040014.

17. Гавадзин Н. О., Клубук А. І., Побігун С. А., Момот В. Л. Зміна моделі роботи чат-ботів у сфері обслуговування клієнтів. Науковий вісник Ужгородського національного університету. Серія : Міжнародні економічні відносини та світове господарство. 2024. Вип. 50. С. 20–24. DOI: 10.32782/2413-9971/2024-50-3.

18. Білан І. А. Глобальні ризики використання чат-ботів, керованих штучним інтелектом. Інформація і право. 2024. № 3(50). С. 147–161.

19. Андрощук А. Г., Малюга О. С. Використання штучного інтелекту у вищій освіті: стан і тенденції. International Science Journal of Education & Linguistics. 2024. Vol. 3, No. 2. P. 27–35. DOI: 10.46299/j.isjel.20240302.04.

20. Головко Д. Ю. Штучний інтелект у діяльності педагога закладу професійної (професійно-технічної) освіти : навчально-методичний посібник. Біла Церква : БІНПО ДЗВО «УМО» НАПН України, 2024. 73 с.

21. Коновальчук Н. Перспективи застосування інструментів на основі штучного інтелекту у викладанні української мови як іноземної. Український педагогічний журнал. 2024.

22. 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.

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

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

25. 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).

ДОДАТОК А

Програмна реалізація обраних архітектур

Лістинг А.1 – Файл конфігурації та ініціалізації сервера

```
1:  const express = require('express');
2:  const cookieParser = require('cookie-parser');
3:  const path = require('path');
4:  require('dotenv').config();
5:  const { initDb } = require('./src/config/db');
6:  const authController = require('./src/controllers/authController');
7:  const adminController = require('./src/controllers/adminController');
8:  const chatController = require('./src/controllers/chatController');
9:  const verifyToken = require('./src/middleware/auth');
10: const app = express();
11: const PORT = process.env.PORT || 3000;
12: app.use(express.json());
13: app.use(express.urlencoded({ extended: true }));
14: app.use(cookieParser());
15: app.use(express.static(path.join(__dirname, 'public')));
16: app.get('/admin', (req, res) => {
17:   res.sendFile(path.join(__dirname, 'public/admin.html'));
18: });
19: app.get('/login', (req, res) => {
20:   res.sendFile(path.join(__dirname, 'public/login.html'));
21: });
22: app.post('/api/chat', chatController.handleChat);
23: app.post('/api/auth/login', authController.login);
24: app.post('/api/auth/logout', authController.logout);
25: app.get('/api/auth/me', authController.me);
26: app.get('/api/admin/knowledge', verifyToken, adminController.getKnowledge);
27: app.post('/api/admin/knowledge', verifyToken, adminController.createKnowledge);
28: app.put('/api/admin/knowledge/:id', verifyToken, adminController.updateKnowledge);
```

```
29:  app.delete('/api/admin/knowledge/:id', verifyToken, adminController.deleteKnowledge);
30:  app.get('/api/admin/settings', verifyToken, adminController.getSettings);
31:  app.put('/api/admin/settings', verifyToken, adminController.updateSettings);
32:  app.get('/api/admin/logs', verifyToken, adminController.getLogs);
33:  app.get('/api/admin/stats', verifyToken, adminController.getStats);
34:  app.use((req, res, next) => {
35:    res.status(404).sendFile(path.join(__dirname, 'public/index.html'));
36:  });
37:  async function startServer() {
38:    try {
39:      await initDb();
40:      app.listen(PORT, () => {
41:      });
42:    } catch (error) {
43:      process.exit(1);
44:    }
45:  }
46:  startServer();
```

Лістинг А.2 – Схема бази даних та міграції SQLite

```
1:  const sqlite3 = require('sqlite3');
2:  const { open } = require('sqlite');
3:  const path = require('path');
4:  const bcrypt = require('bcryptjs');
5:  require('dotenv').config();
6:  const dbPath = path.resolve(__dirname, '../../database.sqlite');
7:  async function getDbConnection() {
8:    return open({
9:      filename: dbPath,
10:     driver: sqlite3.Database
11:   });
```

```
12:     }
13:     async function initDb() {
14:       const db = await getDbConnection();
15:       await db.exec(`
16:         CREATE TABLE IF NOT EXISTS users (
17:           id INTEGER PRIMARY KEY AUTOINCREMENT,
18:           username TEXT UNIQUE NOT NULL,
19:           password_hash TEXT NOT NULL,
20:           created_at DATETIME DEFAULT CURRENT_TIMESTAMP
21:         );
22:       `);
23:       await db.exec(`
24:         CREATE TABLE IF NOT EXISTS knowledge_base (
25:           id INTEGER PRIMARY KEY AUTOINCREMENT,
26:           category TEXT NOT NULL,
27:           title TEXT NOT NULL,
28:           content TEXT NOT NULL,
29:           updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
30:         );
31:       `);
32:       await db.exec(`
33:         CREATE TABLE IF NOT EXISTS conversations (
34:           id INTEGER PRIMARY KEY AUTOINCREMENT,
35:           session_id TEXT NOT NULL,
36:           user_message TEXT NOT NULL,
37:           ai_response TEXT NOT NULL,
38:           escalated INTEGER DEFAULT 0,
39:           created_at DATETIME DEFAULT CURRENT_TIMESTAMP
40:         );
41:       `);
42:       await db.exec(`
43:         CREATE TABLE IF NOT EXISTS settings (
```

```
44:           key TEXT PRIMARY KEY,
45:           value TEXT NOT NULL,
46:           updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
47:         );
48:       `);
49:       const telegramUrl = process.env.TELEGRAM_ADMIN_URL || 'https://t.me/placeholder_school_bot';
50:       await db.run(
51:         `INSERT OR IGNORE INTO settings (key, value) VALUES (?, ?)`,
52:         ['telegram_url', telegramUrl]
53:       );
54:       await db.run(
55:         `INSERT OR IGNORE INTO settings (key, value) VALUES (?, ?)`,
56:         ['school_name', 'MTSchool (Mother Tongue School)']
57:       );
58:       const defaultUser = process.env.DEFAULT_ADMIN_USER || 'admin';
59:       const defaultPass = process.env.DEFAULT_ADMIN_PASSWORD || 'admin_secure_pass_2026';
60:       const adminExists = await db.get('SELECT id FROM users WHERE username = ?', [defaultUser]);
61:       if (!adminExists) {
62:         const salt = await bcrypt.genSalt(10);
63:         const hash = await bcrypt.hash(defaultPass, salt);
64:         await db.run('INSERT INTO users (username, password_hash) VALUES (?, ?)', [defaultUser, hash]);
65:       }
66:       const countKB = await db.get('SELECT COUNT(*) as count FROM knowledge_base');
67:       if (countKB.count === 0) {
68:         const sampleData = [
69:           {
70:             category: 'courses',
71:             title: 'Курси англійської мови (English)',
```

```
72:           content: 'Англійська мова в Mother Tongue School. Програми для будь-якого рівня.'
73:         }
74:       ];
75:       for (const item of sampleData) {
76:         await db.run(
77:           'INSERT INTO knowledge_base (category, title, content) VALUES (?, ?, ?)',
78:           [item.category, item.title, item.content]
79:         );
80:       }
81:     }
82:     await db.close();
83:   }
84:   module.exports = {
85:     getDbConnection,
86:     initDb
87:   };
```

Лістинг А.3 – Маршрутизація та логіка чат-сесій

```
1:   const { getDbConnection } = require('../config/db');
2:   const { generateResponse } = require('../services/aiService');
3:   async function handleChat(req, res) {
4:     try {
5:       const { message, sessionId } = req.body;
6:       if (!message) {
7:         return res.status(400).json({ error: 'Повідомлення не може бути порожнім.' });
8:       }
9:       const sessionKey = sessionId || 'anonymous_session';
10:      const db = await getDbConnection();
11:      const knowledgeBlocks = await db.all('SELECT * FROM knowledge_base');
12:      const telegramSetting = await db.get('SELECT value FROM settings WHERE key = ?', ['telegram_url']);
```

```
13:       const telegramUrl = telegramSetting ? telegramSetting.value : 'https://t.me/placeholder_school_bot';
14:       await db.close();
15:       const aiResult = await generateResponse(message, knowledgeBlocks);
16:       const dbLog = await getDbConnection();
17:       await dbLog.run(
18:         `INSERT INTO conversations (session_id, user_message, ai_response, escalated, created_at)
19:          VALUES (?, ?, ?, ?, CURRENT_TIMESTAMP)`,
20:         [sessionKey, message, aiResult.text, aiResult.escalated ? 1 : 0]
21:       );
22:       await dbLog.close();
23:       const responsePayload = {
24:         response: aiResult.text,
25:         escalated: aiResult.escalated
26:       };
27:       if (aiResult.escalated) {
28:         responsePayload.telegramUrl = telegramUrl;
29:       }
30:       return res.json(responsePayload);
31:     } catch (error) {
32:       return res.status(500).json({ error: 'Виникла внутрішня помилка при обробці запиту.' });
33:     }
34:   }
35:   module.exports = {
36:     handleChat
37:   };
```

Лістинг А.4 – Інтеграція з генеративним ШІ

```
1:   const { OpenAI } = require('openai');
2:   require('dotenv').config();
```

```
3:  const apiKey = process.env.OPENAI_API_KEY;
4:  const model = process.env.OPENAI_MODEL || 'gpt-4o-mini';
5:  let openaiClient = null;
6:  const isMockMode = !apiKey || apiKey === 'your_openai_api_key_here';
7:  if (!isMockMode) {
8:    openaiClient = new OpenAI({
9:      apiKey: apiKey
10:   });
11: }
12: function searchMockKnowledge(query, knowledgeBlocks) {
13:   const cleanQuery = query.toLowerCase();
14:   const escalationPhrases = [
15:     'адмін', 'людин', 'дзвін', 'телефон', 'зв\'яз', 'живий', 'оператор',
16:     'адміністратор', 'telegram', 'телеграм', 'чат', 'менеджер', 'спілкува'
17:   ];
18:   const wantsEscalation = escalationPhrases.some(phrase =>
cleanQuery.includes(phrase));
19:   if (wantsEscalation) {
20:     return {
21:       text: 'Я перенаправлю вас до нашого адміністратора.',
22:       escalated: true
23:     };
24:   }
25:   const matches = knowledgeBlocks.map(block => {
26:     let score = 0;
27:     const queryWords = cleanQuery.split(/[\s,?.!\-()]+/);
28:     queryWords.forEach(word => {
29:       if (word.length < 3) return;
30:       if (block.title.toLowerCase().includes(word)) score += 3;
31:       if (block.content.toLowerCase().includes(word)) score += 1;
32:     });
33:     return { block, score };
```

```
34:       }).filter(m => m.score > 0);
35:       matches.sort((a, b) => b.score - a.score);
36:       if (matches.length > 0) {
37:         const bestBlock = matches[0].block;
38:         return {
39:           text: `Знайдено інформацію у розділі "${bestBlock.title}":\n\n${bestBlock.content}`,
40:           escalated: false
41:         };
42:       }
43:       return {
44:         text: 'Вибачте, але я не знайшов детальної відповіді на це питання у своїй базі знань.',
45:         escalated: true
46:       };
47:     }
48:     async function generateResponse(userMessage, knowledgeBlocks) {
49:       if (isMockMode) {
50:         return searchMockKnowledge(userMessage, knowledgeBlocks);
51:       }
52:       try {
53:         const contextText = knowledgeBlocks.map(block => {
54:           return `[Категорія: ${block.category}] ${block.title}:\n${block.content}`;
55:         }).join('\n\n---\n\n');
56:         const systemPrompt = `Ти — ввічливий штучний інтелект-асистент мовної школи MTSchool. Твоє завдання — відповідати на запитання клієнтів, використовуючи виключно наданий контекст бази знань.
57:     Важливі правила:
58:     1. Твої відповіді мають базуватися виключно на наданій базі знань.
59:     2. Якщо немає відповіді:
60:       - Ввічливо повідомити про це.
61:       - Обов'язково додати маркер "[ESCALATE]".
```

```
3. Спілкуйся українською мовою. Будь привітним.
База знань школи:
${contextText}`;
    const completion = await openaiClient.chat.completions.create({
      model: model,
      messages: [
        { role: 'system', content: systemPrompt },
        { role: 'user', content: userMessage }
      ],
      temperature: 0.2
    });
    const aiText = completion.choices[0].message.content || '';
    const escalated = aiText.includes('[ESCALATE]') ||
                      /зв'язатися з адміністратором/i.test(aiText);
    const cleanedText = aiText.replace('[ESCALATE]', '').trim();
    return {
      text: cleanedText,
      escalated: escalated
    };
  } catch (error) {
    return searchMockKnowledge(userMessage, knowledgeBlocks);
  }
}
module.exports = {
  generateResponse
};
```

Лістинг А.5 – Клієнтська логіка веб-застосунку

```
document.addEventListener('DOMContentLoaded', () => {
  if (window.lucide) {
    window.lucide.createIcons();
  }
```

```
5:        const chatForm = document.getElementById('chatForm');
6:        const chatInput = document.getElementById('chatInput');
7:        const chatMessages = document.getElementById('chatMessages');
8:        const escalationBanner = document.getElementById('escalationBanner');
9:        const telegramEscalateLink = document.getElementById('telegramEscalateLink');
10:       const clearChatBtn = document.getElementById('clearChatBtn');
11:       let sessionId = localStorage.getItem('lingo_chat_session');
12:       if (!sessionId) {
13:         sessionId = 'session_' + Math.random().toString(36).substr(2, 9) + '_' + Date.now();
14:         localStorage.setItem('lingo_chat_session', sessionId);
15:       }
16:       function formatMessage(text) {
17:         let formatted = text
18:           .replace(/&/g, "&amp;")
19:           .replace(/</g, "&lt;")
20:           .replace(/>/g, "&gt;");
21:         formatted = formatted.replace(/\*\*(.*?)\*\*/gs, '<strong>$1</strong>');
22:         formatted = formatted.replace(/\n/g, '<br>');
23:         return formatted;
24:       }
25:       function scrollToBottom() {
26:         chatMessages.scrollTop = chatMessages.scrollHeight;
27:       }
28:       function appendMessage(sender, text, isMarkdown = true) {
29:         const isBot = sender === 'bot';
30:         const bubbleWrapper = document.createElement('div');
31:         bubbleWrapper.className = `flex items-start gap-3 max-w-[85%] ${isBot ? '' : 'ml-
auto flex-row-reverse animate-fade-in-up'}`;
32:         const iconDiv = document.createElement('div');
33:         iconDiv.className = `w-8 h-8 rounded-lg flex items-center justify-center border
border-white/10 ${
34:           isBot ? 'bg-school-teal/50' : 'bg-school-gold/20'
```

```
35:         }`;
36:         iconDiv.innerHTML = isBot
37:           ? '<i data-lucide="bot" class="w-4 h-4 text-school-gold"></i>'
38:           : '<i data-lucide="user" class="w-4 h-4 text-school-gold"></i>';
39:         const contentDiv = document.createElement('div');
40:         contentDiv.className = `${
41:           isBot
42:             ? 'bg-white/5 border border-white/5 px-4 py-3 rounded-2xl rounded-tl-none'
43:             : 'bg-school-teal text-white px-4 py-3 rounded-2xl rounded-tr-none shadow-md'
44:         } leading-relaxed`;
45:         contentDiv.innerHTML = isMarkdown ? formatMessage(text) : text;
46:         bubbleWrapper.appendChild(iconDiv);
47:         bubbleWrapper.appendChild(contentDiv);
48:         chatMessages.appendChild(bubbleWrapper);
49:         scrollToBottom();
50:       }
51:       chatForm.addEventListener('submit', async (e) => {
52:         e.preventDefault();
53:         const query = chatInput.value.trim();
54:         if (!query) return;
55:         appendMessage('user', query);
56:         chatInput.value = '';
57:         try {
58:           const response = await fetch('/api/chat', {
59:             method: 'POST',
60:             headers: {
61:               'Content-Type': 'application/json'
62:             },
63:             body: JSON.stringify({
64:               message: query,
65:               sessionId: sessionId
66:             })
```

```
67:           });
68:           const data = await response.json();
69:           if (response.ok) {
70:             appendMessage('bot', data.response);
71:             if (data.escalated) {
72:               telegramEscalateLink.href = data.telegramUrl || 'https://t.me/bot';
73:               escalationBanner.classList.remove('hidden');
74:               escalationBanner.classList.add('flex');
75:               scrollToBottom();
76:             }
77:           }
78:         } catch (err) {
79:           appendMessage('bot', 'Помилка підключення.');
80:         }
81:       });
82:     });
```