

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ  
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ  
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему: «Розробка SaaS-платформи для управління  
AI-асистентами у B2B моделях»

на здобуття освітнього ступеня бакалавра  
зі спеціальності 121 Інженерія програмного забезпечення  
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання  
ідей, результатів і текстів інших авторів мають посилання  
на відповідне джерело*

\_\_\_\_\_  
(підпис) Сергій МИХАЙЛІВСЬКИЙ

Виконав: здобувач вищої освіти групи ПД-44

\_\_\_\_\_  
Сергій МИХАЙЛІВСЬКИЙ

Керівник: \_\_\_\_\_  
канд. техн. наук Юрій ЗАДОНЦЕВ

Рецензент: \_\_\_\_\_

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**

Завідувач кафедри

Інженерії програмного забезпечення

\_\_\_\_\_ Ірина ЗАМРІЙ

« \_\_\_\_ » \_\_\_\_\_ 2026 р.

**ЗАВДАННЯ  
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Михайлівському Сергію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка SaaS-платформи для управління AI-асистентами у B2B моделях»

керівник кваліфікаційної роботи кандидат технічних наук Юрій ЗАДОНЦЕВ,  
затверджені наказом Державного університету інформаційно-комунікаційних  
технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про використання AI-асистентів для автоматизації бізнес-комунікацій, опис архітектурних підходів до розробки SaaS-платформ, методи інтеграції каналів комунікації з CRM-системами, а також технічна документація з описом засобів розробки серверної та клієнтської частин програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області використання AI-асистентів для автоматизації бізнес-процесів.
2. Проектування SaaS-платформи для управління AI-асистентами у бізнесі.

3. Програмна реалізація та опис функціонування SaaS-платформи для створення, налаштування та використання AI-асистентів.
4. Тестування SaaS-платформи та перевірка основних сценаріїв взаємодії користувачів із системою.
5. Перелік графічного матеріалу: *презентація*
  1. Аналіз аналогів.
  2. Вимоги до SaaS-платформи.
  3. Програмні засоби реалізації.
  4. Діаграма варіантів використання.
  5. Загальна архітектура системи.
  6. ER-діаграма бази даних.
  7. Діаграма послідовності обробки повідомлення клієнта.
  8. Діаграма послідовності створення та виконання AI-асистента
  9. Екранні форми CRM-панелі.
  10. Апробація результатів дослідження.
6. Дата видачі завдання «23» лютого 2026 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи                                                                         | Строк виконання етапів роботи | Примітка |
|-------|-------------------------------------------------------------------------------------------------------------|-------------------------------|----------|
| 1     | Підбір та аналіз науково-технічної літератури                                                               | 23.02-28.02.2026              |          |
| 2     | Аналіз та дослідження існуючих аналогів                                                                     | 01.03-07.03.2026              |          |
| 3     | Аналіз методів автоматизації бізнес-комунікацій за допомогою AI-асистентів                                  | 08.03-17.03.2026              |          |
| 4     | Проектування вебплатформи для створення AI-асистентів, обробки чатів і централізованого керування бізнесами | 18.03-27.03.2026              |          |
| 5     | Програмна реалізація застосунку                                                                             | 28.03-10.04.2026              |          |
| 6     | Тестування застосунку                                                                                       | 11.04-21.04.2026              |          |
| 7     | Оформлення роботи: вступ, висновки, реферат                                                                 | 22.04-29.04.2026              |          |
| 8     | Розробка демонстраційних матеріалів                                                                         | 30.04-10.05.2026              |          |
| 9     | Попередній захист роботи                                                                                    | 11.05-22.05.2026              |          |

Здобувач вищої освіти

\_\_\_\_\_ (підпис)

Сергій МИХАЙЛІВСЬКИЙ

Керівник

кваліфікаційної роботи

\_\_\_\_\_ (підпис)

Юрій ЗАДОНЦЕВ





## РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 82 стор., 4 табл., 15 рис., 26 джерел.

**Мета роботи** – спрощення обслуговування клієнтів бізнесу шляхом автоматизації обробки типових звернень за допомогою AI-асистентів.

**Об'єкт дослідження** – процеси автоматизації взаємодії бізнесу з клієнтами у B2B моделі за допомогою інтелектуальних програмних систем.

**Предмет дослідження** – методи, засоби та архітектурні рішення розробки вебплатформи для створення AI-асистентів, обробки чатів і централізованого керування бізнесами.

**Короткий зміст роботи:** У роботі проаналізовано підходи до використання AI-асистентів для автоматизації бізнес-процесів та клієнтської комунікації. Проаналізовано існуючі програмні рішення для автоматизації підтримки клієнтів і управління діалогами, зокрема Intercom, Zendesk, HubSpot та Freshdesk.

Розроблено архітектуру SaaS-платформи та програмно реалізовано ключові функціональні можливості, зокрема: створення бізнес-профілю, налаштування AI-асистентів, обробку повідомлень клієнтів, інтеграцію з каналами комунікації, збереження історії діалогів, створення лідів та передачу складних звернень оператору підтримки.

Проведено функціональне тестування основних сценаріїв роботи системи. У роботі використано TypeScript, Node.js, React/Next.js, PostgreSQL, Docker та OpenAI Agents SDK для реалізації серверної частини, клієнтської CRM-панелі, бази даних та інтеграції з AI-сервісами.

Сферою використання платформи є автоматизація клієнтської підтримки, первинної консультації користувачів, обробки типових звернень та управління діалогами в діяльності підприємств малого і середнього бізнесу.

**КЛЮЧОВІ СЛОВА:** SAAS-ПЛАТФОРМА, AI-АСИСТЕНТ, CRM-СИСТЕМА, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, КЛІЄНТСЬКА ПІДТРИМКА, ОБРОБКА ПОВІДОМЛЕНЬ, ІСТОРИЯ ДІАЛОГІВ, TYPESCRIPT, NODE.JS, REACT, NEXT.JS, POSTGRESQL, DOCKER, OPENAI AGENTS SDK.

## ЗМІСТ

|                                                                      |    |
|----------------------------------------------------------------------|----|
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....                                       | 9  |
| ВСТУП.....                                                           | 11 |
| 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....                                     | 14 |
| 1.1 Поняття та роль AI-асистентів у бізнес-процесах .....            | 14 |
| 1.2 Проблеми клієнтської комунікації в сучасному бізнесі.....        | 16 |
| 1.3 Аналіз існуючих програмних рішень.....                           | 17 |
| 1.3.1 Аналіз платформи Intercom.....                                 | 18 |
| 1.3.2 Аналіз платформи Zendesk .....                                 | 19 |
| 1.3.3 Аналіз платформи HubSpot.....                                  | 20 |
| 1.3.4 Аналіз платформи Freshdesk.....                                | 20 |
| 1.3.5 Порівняльний аналіз існуючих рішень .....                      | 21 |
| 1.4 Постановка задачі розробки програмного забезпечення .....        | 23 |
| 2 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....                    | 26 |
| 2.1 Обґрунтування вибору архітектурного підходу .....                | 26 |
| 2.2 Засоби реалізації серверної частини .....                        | 28 |
| 2.3 Засоби реалізації клієнтської частини.....                       | 30 |
| 2.4 Система управління базами даних та кешування.....                | 32 |
| 2.5 Засоби інтеграції з AI-сервісами та каналами комунікації .....   | 33 |
| 2.6 Засоби авторизації, оплати, контейнеризації та розгортання ..... | 35 |
| 3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....                         | 38 |
| 3.1 Функціональні та нефункціональні вимоги до системи .....         | 38 |
| 3.2 Загальна архітектура системи.....                                | 42 |

|     |                                                                          |                                     |
|-----|--------------------------------------------------------------------------|-------------------------------------|
| 3.3 | Проектування ролей користувачів та моделі доступу.....                   | 45                                  |
| 3.4 | Проектування механізму обробки повідомлень клієнтів.....                 | 50                                  |
| 3.5 | Проектування бази даних.....                                             | 54                                  |
| 3.6 | Проектування інтеграцій з каналами комунікації .....                     | 59                                  |
| 4   | РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....                  | 63                                  |
| 4.1 | Організація проєкту та структура програмного забезпечення .....          | 63                                  |
| 4.2 | Реалізація API Gateway та внутрішніх сервісів .....                      | 65                                  |
| 4.3 | Реалізація CRM-панелі та клієнтського інтерфейсу.....                    | 69                                  |
| 4.4 | Реалізація AI-асистента, CRM-лідів та передачі діалогу оператору .....   | 76                                  |
| 4.5 | Реалізація інтеграцій з каналами комунікації та платіжною системою ..... | 80                                  |
| 4.6 | Тестування основних сценаріїв роботи системи .....                       | 86                                  |
| 4.7 | Демонстрація результатів роботи програмного забезпечення.....            | 88                                  |
|     | ВИСНОВКИ .....                                                           | 90                                  |
|     | ПЕРЕЛІК ПОСИЛАНЬ .....                                                   | 93                                  |
|     | ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....                                 | 95                                  |
|     | ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ ...                                 | <b>Error! Bookmark not defined.</b> |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

**AI** - Artificial Intelligence, штучний інтелект.

**AI-асистент** - програмний компонент, що використовує засоби штучного інтелекту для обробки повідомлень користувачів, формування відповідей і виконання окремих бізнес-дій.

**API** - Application Programming Interface, програмний інтерфейс для взаємодії між компонентами програмного забезпечення.

**API Gateway** - єдина точка входу до backend-частини системи, яка приймає зовнішні запити та передає їх до відповідних внутрішніх сервісів.

**Backend** - серверна частина програмного забезпечення, яка відповідає за бізнес-логіку, роботу з базою даних, інтеграції та обробку запитів.

**Billing** - модуль системи, що відповідає за тарифні плани, підписки, оплату та перевірку доступу до платних функцій.

**CRM** - Customer Relationship Management, система або модуль для керування взаємодією з клієнтами.

**CRM-лід** - запис про потенційного клієнта, який містить контактні дані, джерело звернення, статус і короткий опис потреби.

**Frontend** - клієнтська частина програмного забезпечення, з якою користувач взаємодіє через браузер.

**Handoff** - передача діалогу від AI-асистента оператору підтримки для ручної обробки звернення.

**HTTP** - HyperText Transfer Protocol, протокол передавання даних у web-середовищі.

**HTTPS** - захищена версія HTTP, що використовує шифрування для передавання даних.

**JSON** - JavaScript Object Notation, текстовий формат обміну структурованими даними.

**JWT** - JSON Web Token, формат токена для передавання інформації про користувача або сесію.

**OIDC** - OpenID Connect, протокол автентифікації користувачів на основі OAuth 2.0.

**REST API** - архітектурний підхід до побудови web API на основі HTTP-запитів.

**SaaS** - Software as a Service, модель надання програмного забезпечення як сервісу через інтернет.

**SDK** - Software Development Kit, набір інструментів і бібліотек для інтеграції з певним сервісом або платформою.

**Thread** - логічний діалог або гілка повідомлень, що об'єднує звернення одного клієнта в межах певного каналу.

**UI** - User Interface, користувацький інтерфейс.

**Webhook** - механізм, за якого зовнішній сервіс автоматично надсилає подію на backend endpoint системи.

**Web-widget** - вбудований чат-модуль, який може бути розміщений на сайті бізнесу для комунікації з клієнтами.

## ВСТУП

*Актуальність:* у сучасних умовах цифровізації бізнесу підприємства все частіше використовують онлайн-канали для взаємодії з клієнтами. Комунікація може відбуватися через веб-сайти, месенджери, електронну пошту та інші цифрові сервіси. Такий підхід розширює можливості бізнесу, однак водночас створює додаткове навантаження на працівників, які повинні швидко відповідати на звернення, обробляти типові запити, передавати складні питання відповідальним спеціалістам і зберігати історію взаємодії з клієнтами.

Особливо актуальною ця проблема є для малого та середнього бізнесу, де значна частина звернень клієнтів повторюється. Користувачі часто запитують про вартість товарів або послуг, умови співпраці, графік роботи, способи оплати, наявність товарів, статус заявки або можливість зв'язку з менеджером. Ручна обробка таких повідомлень потребує часу та може призводити до затримок у відповідях. У результаті підприємство ризикує втратити потенційного клієнта або знизити якість обслуговування.

З розвитком технологій штучного інтелекту виникає можливість створення програмних рішень, що автоматизують первинну комунікацію з клієнтами. Одним із таких рішень є AI-асистент, який здатний аналізувати повідомлення природною мовою, враховувати контекст діалогу, формувати відповіді, створювати ліди та передавати складні звернення оператору підтримки. На відміну від класичних чат-ботів, AI-асистенти можуть більш гнучко реагувати на запити користувачів і працювати з різними сценаріями комунікації.

Веб-платформа для створення та управління AI-асистентами постає як комплексний інструмент, що поєднує функціональність обробки чатів, централізованого керування бізнесами, збереження історії діалогів, роботи з лідами та інтеграції з каналами комунікації. Така система дозволяє бізнесу створювати власних AI-асистентів, налаштовувати їхню поведінку, підключати канали взаємодії з клієнтами та контролювати звернення, які потребують участі оператора.

*Об'єкт дослідження* – процеси автоматизації взаємодії бізнесу з клієнтами у B2B моделі за допомогою інтелектуальних програмних систем.

*Предмет дослідження* – методи, засоби та архітектурні рішення розробки веб-платформи для створення AI-асистентів, обробки чатів і централізованого керування бізнесами.

*Мета роботи* – спрощення обслуговування клієнтів бізнесу шляхом автоматизації обробки типових звернень за допомогою AI-асистентів.

Для досягнення поставленої мети в бакалаврській роботі виконано наступні задачі:

1. Провести аналіз предметної області автоматизації взаємодії бізнесу з клієнтами за допомогою AI-асистентів.
2. Дослідити існуючі програмні рішення для обробки клієнтських звернень, управління чатами та використання AI-асистентів.
3. Визначити функціональні та нефункціональні вимоги до вебплатформи.
4. Спроектувати архітектуру системи, модель доступу користувачів, механізм обробки чатів та структуру бази даних.
5. Реалізувати основні модулі програмного забезпечення: API Gateway, внутрішні сервіси, CRM-панель, модуль AI-асистента, створення лідів та передачу діалогу оператору.
6. Провести тестування основних сценаріїв роботи системи.

У процесі розробки було спроектовано архітектуру вебплатформи, модель доступу користувачів, механізм обробки чатів, структуру бази даних та взаємодію AI-асистента з каналами комунікації. Для реалізації програмного забезпечення використано TypeScript, Node.js, Express, React, Next.js, PostgreSQL, Redis, Docker, Auth0/OIDC, Stripe SDK та OpenAI SDK. Також було реалізовано API Gateway, внутрішні сервіси, CRM-панель, модуль AI-асистента, обробку чатів, створення лідів і передачу складних діалогів оператору підтримки.

*Наукова новизна* роботи полягає в реалізації програмного забезпечення, що поєднує в межах однієї системи створення AI-асистентів, обробку чатів, централізоване керування бізнесами, роботу з CRM-лідами та передачу складних

звернень оператору. Це рішення дозволяє організувати взаємодію клієнтів, AI-асистентів і працівників компанії в єдиному інформаційному середовищі.

*Практична значущість* полягає у створенні інструменту, який може бути використаний підприємствами малого та середнього бізнесу для автоматизації клієнтської підтримки, первинної консультації користувачів, збору лідів, зменшення навантаження на працівників та підвищення швидкості обробки типових звернень. Реалізоване програмне забезпечення може бути адаптоване для інтернет-магазинів, сервісних компаній, освітніх проєктів, консультаційного бізнесу та інших сфер, де важливо швидко й структуровано обробляти повідомлення від клієнтів.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

### 1.1 Поняття та роль AI-асистентів у бізнес-процесах

AI-асистент є програмним компонентом, який використовує засоби штучного інтелекту для взаємодії з користувачем, аналізу повідомлень і формування відповідей [1]. У бізнес-середовищі такий асистент може виконувати роль першої лінії підтримки, помічника з продажів або інструмента для збору первинної інформації про клієнта.

На відміну від звичайного чат-бота, AI-асистент не обмежується лише наперед заданими КНОПКАМИ або жорсткими сценаріями. Класичний чат-бот добре працює тоді, коли користувач обирає один із запропонованих варіантів або вводить очікувану команду. Проте в реальному спілкуванні клієнти часто формулюють питання по-різному, можуть писати неповними реченнями, уточнювати деталі або змінювати тему діалогу. У таких ситуаціях сценарний бот швидко втрачає гнучкість.

AI-асистент може працювати з повідомленнями природною мовою. Він аналізує зміст звернення, враховує попередній контекст діалогу та формує відповідь відповідно до ситуації. Наприклад, якщо клієнт питає про ціну послуги, асистент може надати загальну інформацію або уточнити деталі. Якщо клієнт залишає контактні дані, система може створити лід. Якщо користувач прямо просить зв'язатися з менеджером, діалог може бути переданий оператору підтримки.

У бізнес-процесах AI-асистенти можуть використовуватися в кількох основних напрямках, зокрема для автоматизації підтримки клієнтів, обробки типових звернень і підвищення ефективності роботи працівників [2]. Перший напрям пов'язаний з автоматизацією клієнтської підтримки. У цьому випадку асистент відповідає на типові питання, пояснює умови співпраці, надає інформацію про товари або послуги та допомагає клієнту швидше отримати потрібну відповідь.

Другий напрям стосується первинної обробки заявок. AI-асистент може уточнити ім'я клієнта, контактний номер, електронну пошту, суть звернення або зацікавленість у певній послугі. Після цього отримані дані можуть бути збережені у вигляді CRM-ліда. Це спрощує подальшу роботу менеджера, оскільки він отримує вже структуровану інформацію, а не просто фрагмент переписки.

Третій напрям полягає в допомозі операторам. AI-асистент не обов'язково повинен повністю замінювати людину. У багатьох випадках ефективнішим є змішаний підхід, коли типові питання обробляються автоматично, а складні звернення передаються працівнику компанії. Такий підхід дозволяє зберегти швидкість відповіді та водночас не втратити якість обслуговування у нестандартних ситуаціях.

Четвертий напрям пов'язаний з централізацією роботи з різними каналами комунікації. Бізнес може отримувати звернення з Telegram, web-чату, електронної пошти або інших джерел. Якщо всі повідомлення потрапляють у єдину систему, працівникам легше переглядати історію діалогів, контролювати статуси звернень і працювати з клієнтами без постійного перемикання між різними сервісами.

Для малого та середнього бізнесу AI-асистенти мають практичну цінність, оскільки дозволяють зменшити навантаження на працівників без створення великого відділу підтримки. Водночас для такого бізнесу важливо, щоб система була простою в налаштуванні. Власник або менеджер повинен мати можливість створити асистента, задати йому інструкції, підключити канал комунікації та переглядати результати роботи через зрозумілу CRM-панель.

Таким чином, AI-асистент у бізнес-процесах є не лише інструментом автоматичної відповіді. Його роль ширша: він бере участь у первинній комунікації з клієнтом, допомагає збирати дані, підтримує операторів, працює з історією діалогів і може бути частиною єдиної системи керування бізнес-комунікацією.

## 1.2 Проблеми клієнтської комунікації в сучасному бізнесі

Клієнтська комунікація є однією з ключових частин роботи будь-якого бізнесу. Від того, наскільки швидко й зрозуміло компанія відповідає клієнту, залежить рівень довіри, ймовірність покупки або замовлення послуги, а також загальне враження від взаємодії з брендом. У цифровому середовищі клієнт очікує швидкої реакції, оскільки може одночасно звернутися до кількох компаній і обрати ту, яка відповість першою або найзручніше пояснить умови.

Першою поширеною проблемою є велика кількість типових звернень. У більшості компаній клієнти регулярно питають одне й те саме: вартість товару або послуги, графік роботи, умови доставки, способи оплати, наявність певної позиції, строки виконання замовлення або можливість консультації з менеджером. Такі питання не завжди є складними, але вони повторюються багато разів. У результаті працівники витрачають час на рутину, яку можна частково автоматизувати.

Другою проблемою є затримка відповіді. Якщо клієнт не отримує відповідь одразу, він може втратити інтерес або перейти до конкурента. Особливо це помітно в онлайн-продажах і сервісному бізнесі, де користувач часто приймає рішення швидко. Для невеликої компанії підтримувати постійну швидкість відповіді складно, оскільки працівники можуть бути зайняті іншими задачами або працювати лише в певний час.

Третьою проблемою є розпорошеність каналів комунікації. Бізнес може отримувати повідомлення з різних джерел: сайту, Telegram, email, соціальних мереж або інших сервісів. Якщо ці канали не об'єднані в єдиній системі, працівникам складно контролювати всі звернення. Частина повідомлень може залишитися без відповіді, а історія спілкування з клієнтом може бути неповною.

Четвертою проблемою є відсутність централізованої історії діалогів. Якщо клієнт спочатку звернувся через сайт, а потім написав у месенджер, працівник може не бачити попередній контекст. Через це клієнту доводиться повторно пояснювати ситуацію. Для бізнесу це також незручно, оскільки важлива інформація про потреби користувача може залишатися в окремих чатах або взагалі не зберігатися.

П'ятою проблемою є ручна робота з потенційними клієнтами. Під час діалогу користувач може залишити контактні дані, описати свою потребу або прямо повідомити, що хоче замовити товар чи послугу. Якщо система не створює лід автоматично, менеджер повинен вручну переносити ці дані в CRM або таблицю. Це займає час і створює ризик помилки.

Ще однією проблемою є передача складних звернень людині. Навіть якщо компанія використовує автоматизацію, частина питань все одно потребує участі оператора. Наприклад, клієнт може мати нестандартну ситуацію, складне уточнення або бажання поспілкуватися з менеджером. Якщо система не має механізму передачі діалогу оператору, автоматизація може погіршити досвід користувача замість того, щоб його покращити.

Для малого та середнього бізнесу ці проблеми є особливо важливими, оскільки впровадження AI-рішень дає змогу оптимізувати повторювані операції та підвищити швидкість обробки інформації [8]. Великі компанії можуть використовувати складні CRM-системи, окремі відділи підтримки та індивідуальні інтеграції. Невеликі компанії часто потребують простішого рішення, яке можна швидко впровадити й використовувати без великої технічної команди. Саме тому актуальною є розробка вебплатформи, яка поєднує AI-асистента, обробку чатів, CRM-ліди, передачу діалогу оператору та централізоване керування бізнесами.

Отже, основними проблемами сучасної клієнтської комунікації є повторюваність запитів, затримки відповідей, розпорошеність каналів, відсутність єдиної історії діалогів, ручна робота з лідами та складність передачі звернення оператору. Розробка програмного забезпечення з використанням AI-асистентів дозволяє частково вирішити ці проблеми та зробити обслуговування клієнтів більш організованим.

### **1.3 Аналіз існуючих програмних рішень**

Для визначення вимог до розроблюваного програмного забезпечення було розглянуто існуючі рішення, які використовуються для автоматизації клієнтської

підтримки, обробки звернень, роботи з CRM-даними та організації багатоканальної комунікації. Такий аналіз дозволяє визначити типові функції сучасних платформ, їхні переваги та обмеження.

У межах роботи було обрано чотири платформи для порівняння: Intercom, Zendesk, HubSpot та Freshdesk. Вони пов'язані з клієнтською підтримкою, CRM, автоматизацією комунікації та використанням AI-функцій. Intercom більше орієнтований на чат-комунікацію та AI-підтримку, Zendesk — на тікетну систему, HubSpot — на CRM і роботу з клієнтськими даними, а Freshdesk — на швидке впровадження служби підтримки.

Під час аналізу враховано такі критерії: наявність CRM або зв'язку з CRM-даними, підтримка омніканальної комунікації, можливість самостійного створення асистентів без розробника, орієнтація на B2B SaaS з багатьма бізнесами, простота для малого бізнесу, гнучкість кастомної backend-логіки та можливість адаптації під власну архітектуру.

### 1.3.1 Аналіз платформи Intercom

**Intercom** є платформою для клієнтської підтримки, комунікації з користувачами та автоматизації обробки звернень. Система підтримує live chat, роботу операторів, AI-функції, автоматизацію відповідей і багатоканальну взаємодію. На офіційному сайті Intercom позиціонується як helpdesk-рішення для епохи AI-агентів, а серед можливостей платформи зазначено omnichannel inbox, AI-powered ticketing та допомогу операторам за допомогою Copilot [9].

Перевагою Intercom є зручна централізована робота з повідомленнями клієнтів. Платформа дозволяє об'єднувати звернення з різних каналів в одному інтерфейсі, що спрощує роботу команди підтримки. Також Intercom підтримує омніканальну комунікацію, зокрема email, live chat, phone, SMS і social-канали [10].

Водночас Intercom є готовим комерційним продуктом із власною логікою та архітектурою. Це створює обмеження для випадків, коли потрібно реалізувати власну backend-логіку, специфічну модель даних, кастомний механізм роботи AI-

асистента або повний контроль над інтеграціями. Крім того, для малого бізнесу платформа може бути складною та фінансово не завжди зручною при масштабуванні.

Отже, Intercom є сильним рішенням для клієнтської підтримки з AI-функціями, але не повністю відповідає задачі створення власної вебплатформи з гнучкою backend-логікою, CRM-лідами, обробкою чатів і контролем архітектури.

### 1.3.2 Аналіз платформи Zendesk

**Zendesk** є платформою для організації клієнтського сервісу, роботи з тікетами, обробки звернень і автоматизації підтримки. Zendesk описує себе як AI-first service platform, що дозволяє організаціям обробляти звернення через email, messaging, live chat, phone, social та help center [11].

Сильною стороною Zendesk є тікетна модель. Кожне звернення може бути перетворене на тікет, після чого команда підтримки може відстежувати його статус, призначати відповідального працівника та контролювати процес виконання. Також платформа підтримує AI-функції: self-learning AI agents можуть автоматизувати значну частину взаємодій і покращувати відповіді з кожним вирішенням звернення [11].

Недоліком Zendesk у контексті даної роботи є орієнтація передусім на службу підтримки та тікетну модель. Для розроблюваної системи важливішими є робота з чатами, AI-асистентами, CRM-лідами, передачею діалогу оператору та централізоване керування бізнесами. Крім того, Zendesk є готовою платформою, тому розробник не має повного контролю над внутрішньою логікою обробки повідомлень, структурою даних і способом інтеграції з власними сервісами.

Отже, Zendesk є ефективним інструментом для великих команд підтримки, але для створення власної гнучкої вебплатформи з AI-асистентами та кастомною архітектурою доцільно реалізувати окреме програмне забезпечення.

### 1.3.3 Аналіз платформи HubSpot

**HubSpot** є комплексною CRM-платформою, яка поєднує інструменти для маркетингу, продажів, клієнтського сервісу та автоматизації бізнес-процесів. У межах даної роботи найбільш важливими є можливості HubSpot Service Hub, оскільки вони пов'язані з клієнтськими зверненнями, тікетами, CRM-даними та AI-функціями. HubSpot Service Hub позиціонується як AI-powered, omni-channel рішення, підключене до marketing і sales даних на єдиній customer platform [12].

Головною перевагою HubSpot є сильна CRM-основа. Платформа дозволяє працювати з контактами, компаніями, угодами, зверненнями та історією взаємодії з клієнтом. Це важливо для бізнесу, оскільки звернення клієнтів можуть бути пов'язані з продажами, маркетинговими діями та подальшою роботою менеджерів. HubSpot також підтримує AI-функції в help desk software, зокрема ticket summaries, reply recommendations, sentiment та інші інструменти для підтримки операторів [12].

Основним обмеженням HubSpot є складність екосистеми та залежність від готової логіки платформи. Для малого бізнесу повний набір маркетингових, sales і service-інструментів може бути надлишковим. Крім того, у випадку потреби повного контролю над AI runtime, backend-сервісами, структурою бази даних і механізмом створення лідів готова CRM-платформа не повністю закриває задачу.

Отже, HubSpot є сильним рішенням для CRM і клієнтського сервісу, але для задачі створення власної вебплатформи з AI-асистентами, чатами, CRM-лідами та гнучкою архітектурою необхідне спеціалізоване програмне забезпечення.

### 1.3.4 Аналіз платформи Freshdesk

**Freshdesk** є платформою для клієнтської підтримки, що входить до екосистеми Freshworks. Вона використовується для роботи зі зверненнями клієнтів, тікетами, базою знань, каналами підтримки та автоматизацією процесів

обслуговування. Freshdesk позиціонується як AI-powered customer service solution, що є простою у налаштуванні та використанні [13].

Freshdesk Omni поєднує omnichannel support software, AI-powered chatbots і ticketing [13]. Це дозволяє компаніям обробляти звернення з різних каналів, використовувати чат-ботів і організовувати роботу команди підтримки. Перевагою Freshdesk є відносна простота впровадження, що робить платформу зручною для невеликих команд.

Водночас Freshdesk має обмеження, характерні для готових комерційних платформ. Користувач працює в межах логіки, передбаченої продуктом. Якщо потрібно реалізувати власний спосіб обробки чатів, власну структуру бізнес-просторів, окрему AI runtime логіку, кастомну модель доступу або специфічні інтеграції, можливості платформи можуть бути недостатньо гнучкими.

Отже, Freshdesk є зручним інструментом для організації клієнтської підтримки, але для створення власної системи з AI-асистентами, CRM-лідами, операторською передачею та повним контролем архітектури його можливості не повністю відповідають поставленій задачі.

### **1.3.5 Порівняльний аналіз існуючих рішень**

Після розгляду окремих платформ було виконано їх порівняння за критеріями, важливими для розроблюваного програмного забезпечення. Основну увагу приділено CRM-функціональності, підтримці різних каналів комунікації, можливості створення асистентів, простоті використання для малого бізнесу та гнучкості адаптації під власну архітектуру.

Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1

## Порівняння існуючих програмних рішень

| Показник                                 | Intercom                                              | Zendesk                                                    | HubSpot                                                  | Freshdesk                                              | Розроблена платформа                                       |
|------------------------------------------|-------------------------------------------------------|------------------------------------------------------------|----------------------------------------------------------|--------------------------------------------------------|------------------------------------------------------------|
| <b>Основна задача системи</b>            | Підтримка клієнтів у чатах та inbox                   | Обробка звернень через ticketing/support                   | Управління клієнтами, продажами та сервісом              | Омніканальна підтримка клієнтів                        | Створення та керування AI-асистентами для бізнесу          |
| <b>Що є центром роботи системи</b>       | Клієнтська розмова                                    | Тікет / звернення                                          | CRM-контакт, компанія, угода                             | Тікет / звернення                                      | Бізнес-простір, AI-асистент, чат і лід                     |
| <b>Як використовується AI</b>            | Fin AI Agent автоматизує відповіді у customer service | AI Agents автоматизують запити у support                   | Breeze Agent допомагає у CRM, sales, marketing і service | Freddy Agent автоматизує support-звернення             | AI-асистент створюється як окрема сутність бізнесу         |
| <b>Запуск для нового бізнесу</b>         | Налаштувати workspace, inbox, knowledge base та Fin   | Налаштувати support-акаунт, канали, базу знань і AI Agents | Налаштувати CRM, об'єкти, канали та Customer Agent       | Налаштувати helpdesk, канали, knowledge base та Freddy | Створити бізнес-простір, AI-асистента і підключити канал   |
| <b>Результат після звернення клієнта</b> | Діалог у inbox та історія підтримки                   | Тікет або звернення в support-системі                      | Запис у CRM та історія взаємодії з клієнтом              | Тікет або діалог у helpdesk                            | Чат, повідомлення, лід або handoff у межах бізнес-простору |

Аналіз показує, що всі розглянуті платформи мають розвинені можливості для клієнтської підтримки. Intercom добре підходить для live chat і AI-підтримки, Zendesk — для тікетної системи та служби підтримки, HubSpot — для CRM і роботи з клієнтськими даними, Freshdesk — для швидкого впровадження підтримки в невеликих командах.

Порівняння також показує, що розроблювана платформа відрізняється не лише набором функцій, а й логікою організації роботи. У розглянутих рішеннях основою системи переважно є inbox, тікет, CRM-контакт або helpdesk-звернення. У розроблюваній платформі центральними сутностями є бізнес-простір, AI-асистент,

чат і лід. Це дозволяє розглядати систему не просто як інструмент підтримки, а як окрему B2B SaaS-платформу для керування AI-комунікацією бізнесу з клієнтами.

Окремо слід зазначити, що для розроблюваної системи важливою є не тільки наявність AI-функцій, а й можливість керувати повним циклом взаємодії з клієнтом: від першого повідомлення до створення ліда або передачі діалогу оператору. Саме тому порівняння проводилося не лише за наявністю окремих інструментів, а й за тим, наскільки система дозволяє об'єднати AI-асистента, канали комунікації, CRM-дані та бізнес-логіку в єдиному середовищі.

Водночас жодне з рішень повністю не відповідає задачі розробки власної вебплатформи, орієнтованої на створення AI-асистентів, обробку чатів, CRM-лідів, передачу діалогу оператору та централізоване керування бізнесами. Готові платформи забезпечують швидкий старт, але обмежують гнучкість backend-логіки, структуру даних, механізми інтеграції та контроль над архітектурою.

Таким чином, результати аналізу підтверджують доцільність розробки власного програмного забезпечення, яке поєднує AI-асистентів, обробку чатів, CRM-функціональність, операторську підтримку, інтеграції з каналами комунікації та можливість адаптації під власну архітектуру.

#### **1.4 Постановка задачі розробки програмного забезпечення**

На основі аналізу предметної області та існуючих програмних рішень було визначено необхідність розробки вебплатформи для створення AI-асистентів, обробки чатів і централізованого керування бізнесами. Розглянуті аналоги мають розвинені можливості для клієнтської підтримки, роботи з CRM-даними та багатоканальної комунікації, однак вони не повністю відповідають задачі створення власної системи з гнучкою backend-логікою, контролем архітектури та можливістю адаптації під конкретні бізнес-процеси.

Основна задача розробки полягає у створенні програмного забезпечення, яке дозволяє бізнесу самостійно налаштовувати AI-асистента, підключати канали комунікації, обробляти повідомлення клієнтів, зберігати історію діалогів,

створювати CRM-ліди та передавати складні звернення оператору підтримки. Система має бути орієнтована на малий і середній бізнес, для якого важливими є просте впровадження, зрозумілий інтерфейс і можливість працювати з клієнтами без використання великої кількості окремих сервісів.

Розроблюване програмне забезпечення повинно підтримувати роботу з бізнес-просторами. Кожен бізнес має мати власних користувачів, AI-асистентів, чати, ліди, інтеграції та налаштування. При цьому дані різних бізнесів повинні бути логічно ізольовані, а доступ до них має надаватися відповідно до ролі користувача в конкретному бізнес-просторі.

Важливою частиною системи є механізм обробки чатів. Повідомлення клієнта має надходити з каналу комунікації, перетворюватися у внутрішній формат, прив'язуватися до конкретного бізнесу, клієнта і діалогу, після чого передаватися до модуля AI-асистента. Відповідь асистента повинна зберігатися в історії повідомлень і повертатися клієнту через той самий канал комунікації.

Система також повинна підтримувати створення CRM-лідів і передачу діалогу оператору. Якщо під час спілкування клієнт залишає контактні дані або демонструє зацікавленість у послугі, ці дані мають зберігатися у структурованому вигляді. Якщо звернення є складним або клієнт просить зв'язатися з менеджером, діалог повинен бути переданий оператору підтримки з можливістю перегляду попередньої історії спілкування.

З технічної точки зору система повинна мати модульну архітектуру. Доцільним є використання API Gateway як єдиної точки входу для зовнішніх запитів, окремих внутрішніх сервісів для роботи з бізнесами, асистентами, чатами та інтеграціями, а також реляційної бази даних для збереження основних сутностей системи. Для каналів комунікації доцільно використовувати підхід адаптерів, щоб Telegram, web-widget та email-сценарії передавали повідомлення в єдиний механізм обробки чатів.

Розроблюване програмне забезпечення повинно відповідати таким основним вимогам:

- забезпечувати авторизацію користувачів;

- підтримувати створення і керування бізнес-просторами;
- надавати можливість створення та налаштування AI-асистентів;
- приймати повідомлення з різних каналів комунікації;
- зберігати історію чатів і повідомлень;
- створювати CRM-ліди на основі даних з діалогів;
- підтримувати передачу складних звернень оператору;
- надавати CRM-панель для роботи з чатами, лідами, інтеграціями та налаштуваннями;
- забезпечувати логічну ізоляцію даних різних бізнесів;
- підтримувати подальше розгортання системи у серверному середовищі.

Отже, задача розробки полягає у створенні вебплатформи, яка поєднує AI-асистентів, обробку чатів, CRM-ліди, операторську підтримку, інтеграції з каналами комунікації та централізоване керування бізнесами. Реалізація такого програмного забезпечення дозволить спростити обслуговування клієнтів бізнесу та автоматизувати обробку типових звернень.

## 2 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У другому розділі розглянуто засоби, які були використані для реалізації програмного забезпечення. Оскільки розроблювана система повинна підтримувати створення AI-асистентів, обробку чатів, централізоване керування бізнесами, інтеграцію з каналами комунікації та збереження даних, для її побудови було обрано набір технологій, що дозволяє реалізувати як серверну, так і клієнтську частину.

Під час вибору засобів реалізації враховувалися такі вимоги: можливість побудови вебплатформи, підтримка роботи з API, зручна організація клієнтського інтерфейсу, надійне збереження даних, можливість інтеграції із зовнішніми сервісами, підтримка контейнеризованого розгортання та подальше масштабування системи [3].

Основу програмного забезпечення складають серверна частина, клієнтська CRM-панель, база даних, кеш, інтеграційні сервіси та зовнішні API. Для серверної частини використано Node.js, TypeScript та Express. Для клієнтської частини використано React і Next.js. Для збереження основних даних обрано PostgreSQL, а для кешування, подій і допоміжних механізмів використано Redis. Для контейнеризації застосовано Docker, а для інтеграції з AI-функціональністю використано OpenAI SDK.

### 2.1 Обґрунтування вибору архітектурного підходу

Розроблюване програмне забезпечення має працювати не як окремий чат-бот, а як вебплатформа, що об'єднує кілька функціональних напрямів. Система повинна забезпечувати роботу з бізнесами, користувачами, AI-асистентами, чатами, лідами, каналами комунікації та платіжною моделлю. Через це для реалізації було обрано модульний архітектурний підхід, за якого різні частини системи відповідають за окремі функціональні області [4].

Використання модульної архітектури дозволяє краще розділити відповідальність між компонентами [6]. Наприклад, один сервіс може відповідати за бізнеси та користувачів, інший за AI-асистентів, окремий сервіс за обробку чатів, а інтеграційні сервіси за роботу з Telegram, email або web-widget. Такий підхід спрощує підтримку коду, оскільки логіка кожного модуля не змішується з іншими частинами системи.

Важливим елементом архітектури є API Gateway. Він використовується як єдина точка входу для зовнішніх запитів. Клієнтська частина не повинна напряму звертатися до внутрішніх сервісів, оскільки це ускладнює контроль доступу та підвищує ризик помилок у безпеці. Gateway приймає запити від клієнтського інтерфейсу, перевіряє авторизацію користувача та передає запит до відповідного внутрішнього сервісу.

Такий підхід є зручним для багатокористувацької системи, де різні бізнеси працюють в межах однієї платформи. Для кожного запиту потрібно визначити, хто є користувачем, до якого бізнес-простору він має доступ і які дії йому дозволено виконувати. API Gateway допомагає централізувати частину цієї логіки та не відкривати внутрішні сервіси напряму для зовнішнього середовища.

Ще однією причиною вибору модульної архітектури є необхідність підтримки кількох каналів комунікації. Повідомлення можуть надходити з Telegram, web-widget або email-сценаріїв. При цьому основна логіка обробки повідомлень має залишатися спільною. Для цього канали комунікації доцільно розглядати як адаптери, які приймають повідомлення зі свого середовища, приводять їх до єдиного формату та передають у сервіс обробки чатів.

Сервіс обробки чатів відповідає за роботу з повідомленнями, контекстом діалогу, історією спілкування та взаємодією з AI-асистентом. Він не повинен залежати від того, з якого саме каналу надійшло повідомлення. Завдяки цьому система може розширюватися новими каналами без повної зміни основної логіки роботи AI-асистента.

Окремо важливою є робота з бізнес-просторами. Кожен бізнес у системі має власних користувачів, асистентів, чати, ліди, інтеграції та налаштування. Тому

архітектура повинна підтримувати логічну ізоляцію даних. Це означає, що користувач одного бізнесу не повинен мати доступ до даних іншого бізнесу. Для цього в системі використовується модель членства користувача в бізнес-просторі та перевірка прав доступу під час виконання операцій.

Модульний підхід також спрощує подальший розвиток системи. Наприклад, якщо в майбутньому потрібно додати новий канал комунікації, достатньо створити окремий інтеграційний модуль, який буде передавати повідомлення в загальний механізм обробки чатів. Якщо потрібно розширити платіжну модель, зміни можна вносити в окремий billing-модуль, не переписуючи логіку AI-асистента або CRM-панелі.

Для клієнтської частини було обрано підхід окремого вебзастосунку. CRM-панель повинна надавати користувачу доступ до основних функцій системи: перегляду бізнесів, налаштування AI-асистентів, роботи з чатами, перегляду лідів, керування інтеграціями та підписками. Відокремлення frontend-частини від backend-сервісів дозволяє незалежно розвивати інтерфейс і серверну логіку.

Таким чином, обраний архітектурний підхід дозволяє побудувати систему як набір взаємопов'язаних компонентів, кожен з яких виконує окрему роль [5]. API Gateway відповідає за зовнішню точку входу, внутрішні сервіси реалізують бізнес-логіку, інтеграційні модулі працюють з каналами комунікації, сервіс чатів забезпечує обробку повідомлень, а клієнтська CRM-панель надає користувачу зручний інтерфейс для керування системою.

## **2.2 Засоби реалізації серверної частини**

Серверна частина програмного забезпечення відповідає за основну бізнес-логіку системи. Вона обробляє запити від клієнтської CRM-панелі, перевіряє доступ користувачів, працює з бізнес-просторами, AI-асистентами, чатами, лідами, інтеграціями та платіжною моделлю. Також backend забезпечує взаємодію з базою даних, зовнішніми API та сервісами обробки повідомлень.

Для реалізації серверної частини було обрано TypeScript, Node.js та Express.js. TypeScript використано як основну мову розробки, оскільки він розширює JavaScript статичною типізацією та дозволяє описувати структури запитів, відповідей, моделей даних і внутрішніх DTO [14]. У межах розроблюваної системи це важливо для роботи з користувачами, бізнесами, асистентами, чатами, повідомленнями, лідами, підписками та інтеграціями.

Node.js використано як середовище виконання серверної частини [14]. Його застосування є доцільним для реалізації web API, обробки HTTP-запитів, webhook-подій, інтеграцій із зовнішніми сервісами та взаємодії з AI-модулем. Для розроблюваної системи це дозволяє використовувати єдину JavaScript/TypeScript-екосистему як на backend, так і на frontend-рівні.

Express.js застосовано для побудови HTTP API, маршрутизації запитів, підключення middleware та обробки помилок [15]. У системі Express.js використовується для реалізації API Gateway і внутрішніх сервісів. Через API Gateway проходять запити до модулів бізнесів, асистентів, чатів, інтеграцій, авторизації та оплати. Такий підхід дозволяє централізувати обробку зовнішніх запитів і не відкривати внутрішні сервіси напяму.

Для роботи з базою даних використано Sequelize. Він дозволяє описувати моделі даних, виконувати запити до PostgreSQL і працювати зі зв'язками між сутностями. У межах системи Sequelize використовується для роботи з користувачами, бізнесами, членством у бізнес-просторах, асистентами, повідомленнями, лідами, інтеграціями та підписками.

Для перевірки вхідних даних застосовано Zod. Його використано для опису схем валідації запитів, параметрів, payload-структур та даних, які передаються між сервісами. Це дозволяє зменшити ризик некоректної обробки повідомлень, створення неправильних записів у базі даних або виконання дій із неповними даними.

Для логування серверної частини використано Pino Logger. Логування необхідне для відстеження роботи API Gateway, внутрішніх сервісів, webhook-подій, інтеграцій, помилок обробки повідомлень і результатів виконання AI-

інструментів. У системі це допомагає аналізувати помилки та швидше знаходити проблеми під час розробки й розгортання.

Для тестування серверної частини використано Vitest та Supertest. Vitest застосовується для перевірки окремих функцій і сервісної логіки, а Supertest — для тестування HTTP endpoint-ів. Такий підхід дозволяє перевіряти коректність API-запитів, обробку помилок, авторизацію, створення сутностей і взаємодію між окремими частинами backend.

Отже, обраний набір засобів серверної розробки дозволяє реалізувати модульну backend-частину системи, забезпечити типізацію даних, валідацію запитів, маршрутизацію, роботу з базою даних, логування та тестування основних сценаріїв. Це відповідає вимогам розроблюваної вебплатформи, яка повинна обробляти чати, керувати AI-асистентами, працювати з CRM-лідами, підтримувати інтеграції та забезпечувати централізоване керування бізнесами.

## 2.3 Засоби реалізації клієнтської частини

Клієнтська частина програмного забезпечення відповідає за взаємодію користувача із системою. Через CRM-панель користувач може працювати з бізнес-просторами, створювати та налаштовувати AI-асистентів, переглядати чати, обробляти ліди, керувати інтеграціями, перевіряти підписку та виконувати адміністративні дії. Тому для реалізації frontend-частини було важливо обрати технології, які дозволяють створити зручний, адаптивний і структурований web-інтерфейс.

Основою клієнтської частини є **React** та **Next.js**. React використано для побудови користувацького інтерфейсу на основі компонентного підходу [16]. У межах розроблюваної системи це дозволяє повторно використовувати елементи інтерфейсу, зокрема кнопки, форми, таблиці, картки, модальні вікна та блоки навігації. Завдяки цьому CRM-панель має єдину структуру й однакову логіку взаємодії на різних сторінках.

Next.js використано для організації структури web-застосунку, маршрутизації та побудови сторінок CRM-панелі [17]. У системі окремі сторінки передбачені для роботи з бізнесами, асистентами, чатами, лідами, інтеграціями, billing-сценаріями та налаштуваннями. Такий підхід дозволяє логічно розділити інтерфейс за функціональними зонами та спростити подальше розширення клієнтської частини.

Для роботи із серверними даними використано React Query. Він спрощує отримання, кешування та оновлення даних, які надходять із backend API. У межах CRM-панелі це важливо для списків бізнесів, асистентів, чатів, лідів, інтеграцій і тарифних планів. Завдяки кешуванню користувач отримує швидшу реакцію інтерфейсу, а після виконання дій, наприклад створення асистента або зміни статусу ліда, відповідні дані можуть бути оновлені без повного перезавантаження сторінки.

Для реалізації форм використано React Hook Form. Він застосовується під час створення бізнесу, налаштування AI-асистента, підключення інтеграцій, редагування параметрів і введення даних користувачем. Використання цього інструмента дозволяє зручно керувати станом форми, обробляти помилки валідації та передавати підготовлені дані на backend.

Для стилізації інтерфейсу використано Tailwind CSS. Його застосування дозволяє швидко створювати адаптивні елементи інтерфейсу та підтримувати єдиний візуальний стиль CRM-панелі. У розроблюваній системі Tailwind CSS використовується для оформлення сторінок, таблиць, форм, карток, навігаційних елементів і станів взаємодії користувача з інтерфейсом.

Для відображення іконок використано Lucide React. Іконки застосовуються в навігації, кнопках дій, картках інтеграцій, сторінках налаштувань і елементах CRM-панелі. Це покращує сприйняття інтерфейсу та допомагає користувачу швидше орієнтуватися в основних функціях системи.

Отже, обрані засоби реалізації клієнтської частини дозволяють створити структуровану CRM-панель для керування бізнесами, AI-асистентами, чатами, лідами, інтеграціями та підписками. React і Next.js забезпечують основу web-

застосунку, React Query відповідає за роботу з серверними даними, React Hook Form спрощує реалізацію форм, Tailwind CSS забезпечує стилізацію, а Lucide React використовується для візуального оформлення інтерфейсу.

## 2.4 Система управління базами даних та кешування

Для роботи програмного забезпечення необхідно забезпечити надійне збереження даних, які виникають під час взаємодії користувачів із системою. До таких даних належать облікові записи користувачів, бізнес-простори, членство користувачів у бізнесах, налаштування AI-асистентів, історія чатів, повідомлення, CRM-ліди, інтеграції, підписки та службові події. Оскільки ці сутності мають чіткі зв'язки між собою, для збереження основних даних було обрано PostgreSQL.

**PostgreSQL** є реляційною базою, яка підходить для збереження структурованих даних і підтримує роботу зі зв'язками між таблицями [18]. У розроблюваній системі PostgreSQL використовується як основне сховище даних. У ньому зберігається інформація про користувачів, бізнеси, ролі учасників, AI-асистентів, чати, повідомлення, ліди, інтеграції з каналами комунікації та стан підписок.

Використання реляційної бази даних є доцільним для цієї задачі, оскільки система працює з багатьма пов'язаними сутностями. Наприклад, один користувач може бути учасником кількох бізнесів, один бізнес може мати кількох AI-асистентів, кожен асистент може бути пов'язаний із чатами, а повідомлення повинні зберігатися в межах конкретного діалогу. Така структура потребує контролю зв'язків, цілісності даних і можливості виконання запитів за різними параметрами.

Окрему роль у системі відіграє логічна ізоляція даних різних бізнесів. Кожен бізнес-простір має власні чати, асистентів, ліди, інтеграції та налаштування. Тому під час проєктування бази даних важливо передбачити зв'язок більшості сутностей із конкретним бізнесом. Це дозволяє перевіряти доступ користувача до даних і запобігати ситуаціям, коли інформація одного бізнесу може бути доступна іншому.

Для допоміжних інфраструктурних задач використано Redis. **Redis** є сховищем даних у пам'яті, яке може застосовуватися для кешування, роботи з тимчасовими даними, подіями та механізмами pub/sub [19]. У межах розроблюваної системи Redis використовується як додатковий компонент, який допомагає оптимізувати роботу backend-сервісів і підтримувати сценарії, що не потребують довготривалого збереження в основній базі даних.

Redis може використовуватися для кешування службових даних, тимчасових станів, блокувань і подій, пов'язаних з обробкою повідомлень. Для chat-runtime логіки це особливо важливо, оскільки одночасна обробка кількох повідомлень одного діалогу може призвести до порушення порядку відповідей або повторного виконання окремих дій. Використання допоміжних механізмів блокування дозволяє краще контролювати послідовність обробки звернень.

Таким чином, PostgreSQL і Redis виконують різні, але взаємодоповнювальні функції. PostgreSQL використовується для надійного довготривалого збереження основних даних системи, а Redis — для кешування, тимчасових станів, подій і допоміжних механізмів обробки повідомлень. Такий підхід дозволяє поєднати структуроване збереження бізнес-даних із більш швидкою обробкою окремих runtime-сценаріїв.

## 2.5 Засоби інтеграції з AI-сервісами та каналами комунікації

Однією з ключових особливостей розроблюваного програмного забезпечення є можливість інтеграції з AI-сервісами та різними каналами комунікації. Система повинна не лише зберігати дані про бізнеси, асистентів і чати, а й забезпечувати реальну взаємодію між клієнтом, AI-асистентом та оператором підтримки. Для цього було використано OpenAI SDK, Telegram Bot API, Gmail API як основу для email-сценаріїв, а також підхід web-widget для підключення чату на сайт бізнесу.

**OpenAI SDK** використано для реалізації AI-функціональності системи [20]. За його допомогою AI-асистент може отримувати повідомлення клієнта,

враховувати контекст діалогу, формувати відповідь і за потреби викликати внутрішні інструменти системи. У розроблюваній платформі AI-асистент не розглядається лише як генератор тексту. Він є частиною бізнес-процесу, оскільки може брати участь у створенні CRM-ліда, передавати діалог оператору та працювати з налаштуваннями конкретного бізнесу.

Важливим аспектом AI-інтеграції є робота з контекстом. Для коректної відповіді асистенту необхідно враховувати не лише поточне повідомлення користувача, а й попередню історію діалогу, канал комунікації, бізнес, до якого належить звернення, та налаштування самого асистента. Саме тому повідомлення клієнта передається до AI-модуля разом із системним контекстом, який формується серверною частиною.

Для підключення Telegram використано **Telegram Bot API** [21]. Telegram-інтеграція дозволяє приймати повідомлення клієнтів через webhook, обробляти події та надсилати відповіді від імені бота. У межах системи Telegram виступає одним із зовнішніх каналів комунікації, а отримані з нього повідомлення приводяться до внутрішнього формату та передаються в загальний механізм обробки чатів.

Email-сценарій розглядається як ще один напрям інтеграції з каналами комунікації. Для подальшої реалізації такого каналу може використовуватися Gmail API, який надає можливості для роботи з повідомленнями електронної пошти [22]. У контексті розроблюваної системи email-канал може застосовуватися для обробки звернень, що надходять на поштову скриньку бізнесу, та передачі їх у загальну логіку chat service.

**Web-widget** використовується як канал комунікації, який може бути вбудований на сайт бізнесу. Його задача полягає в тому, щоб надати клієнту можливість написати повідомлення безпосередньо на web-сторінці. Для бізнесу це зручно, оскільки користувач не повинен переходити в інший сервіс або месенджер. Повідомлення з web-widget також передаються в загальний механізм обробки чатів, де вони зберігаються, аналізуються AI-асистентом і можуть бути передані оператору.

Для уніфікації роботи з різними каналами використано підхід platform adapters. Його суть полягає в тому, що кожен зовнішній канал має власний адаптер, який відповідає за приймання подій, перевірку вхідних даних, перетворення повідомлення у внутрішній формат і передачу його до сервісу чатів. Завдяки цьому основна логіка AI-асистента не залежить від конкретного джерела повідомлення.

Такий підхід спрощує подальше розширення системи. Якщо в майбутньому потрібно додати новий канал комунікації, наприклад інший месенджер або форму на сайті, достатньо реалізувати новий адаптер, який буде передавати повідомлення у вже існуючий механізм обробки. При цьому логіка створення лідів, збереження історії діалогів, передача оператору та робота AI-асистента залишаються спільними для всіх каналів.

Отже, обрані засоби інтеграції дозволяють поєднати AI-функціональність із реальними каналами взаємодії з клієнтами. OpenAI SDK забезпечує роботу AI-асистента, Telegram Bot API використовується для підключення Telegram-каналу, Gmail API розглядається як основа для email-сценаріїв, web-widget дозволяє організувати чат на сайті бізнесу, а підхід platform adapters забезпечує єдину логіку обробки повідомлень незалежно від каналу комунікації.

## **2.6 Засоби авторизації, оплати, контейнеризації та розгортання**

Для роботи розроблюваного програмного забезпечення важливо забезпечити не лише реалізацію основної бізнес-логіки, а й підтримку авторизації користувачів, підписної моделі доступу, контейнеризованого запуску та автоматизації процесів розгортання. Оскільки система орієнтована на роботу з бізнес-просторами, чатами, лідами та платіжними функціями, ці засоби є необхідними для безпечної та стабільної експлуатації платформи.

Для автентифікації користувачів використано підхід Auth0 / OIDC. OpenID Connect є протоколом автентифікації, що дозволяє отримувати підтверджену інформацію про користувача на основі стандартизованого механізму [23]. У межах

системи це дає змогу винести частину логіки входу, керування сесіями та перевірки користувача на рівень зовнішнього провайдера автентифікації.

Такий підхід є доцільним для SaaS-платформи, оскільки система повинна працювати з різними користувачами та бізнесами. Після автентифікації користувача backend може визначити його локальний профіль, бізнес-простори, до яких він має доступ, і роль у кожному з них. Це дозволяє розмежувати права власника бізнесу, адміністратора та оператора підтримки.

Для реалізації платіжної моделі використано Stripe Billing. Stripe надає інструменти для роботи з підписками, тарифними планами та оплатою доступу до сервісу [24]. У розроблюваній системі платіжна модель використовується для керування доступом бізнесу до функціональності платформи залежно від активного тарифу або стану підписки.

Окрему роль у платіжній логіці відіграють Stripe webhooks. Вони дозволяють backend-частині отримувати події від платіжної системи, наприклад підтвердження оплати, зміну тарифу або скасування підписки [24]. Це важливо, оскільки стан підписки не повинен оновлюватися лише після повернення користувача на сторінку успішної оплати. Надійнішим є підхід, за якого остаточне оновлення доступу виконується після отримання підтвердженої webhook-події.

Для контейнеризації системи використано Docker та Docker Compose. Docker Compose дозволяє описувати й запускати кілька контейнерів як єдине середовище [25]. У межах розроблюваного програмного забезпечення це зручно для запуску backend-сервісів, бази даних PostgreSQL, Redis та інших інфраструктурних компонентів. Контейнеризація спрощує локальну розробку, тестування та подальше розгортання системи на сервері.

Перевагою Docker Compose є можливість описати залежності між сервісами в одному конфігураційному файлі. Наприклад, backend-сервіси можуть запускатися разом із базою даних і кешем, а розробник отримує повторюване середовище для перевірки роботи системи. Це зменшує кількість помилок, пов'язаних із різницею між локальним середовищем і сервером.

Для автоматизації процесів перевірки та розгортання використано GitHub Actions. Цей інструмент дозволяє створювати CI/CD-сценарії для встановлення залежностей, запуску тестів, збірки застосунку та перевірки конфігурацій [26]. У межах системи GitHub Actions може використовуватися для автоматичної перевірки backend- і frontend-частини після внесення змін у репозиторій.

Автоматизація CI/CD-процесів є важливою для підтримки якості програмного забезпечення. Вона дозволяє швидше виявляти помилки, перевіряти працездатність основних модулів і зменшувати ризик ручних помилок під час підготовки застосунку до розгортання. Для системи, що складається з кількох сервісів, це особливо важливо, оскільки зміни в одному модулі можуть впливати на роботу інших компонентів.

Отже, засоби авторизації, оплати, контейнеризації та розгортання забезпечують інфраструктурну основу програмного забезпечення. Auth0 / OIDC використовується для автентифікації користувачів, Stripe Billing і webhooks — для реалізації підписочної моделі, Docker Compose — для контейнеризованого запуску сервісів, а GitHub Actions — для автоматизації перевірки та підготовки системи до розгортання.

## 3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 3.1 Функціональні та нефункціональні вимоги до системи

Проєктування програмного забезпечення починається з визначення вимог до системи. Вимоги дозволяють описати, які можливості повинна мати вебплатформа, які користувацькі сценарії вона повинна підтримувати та які якісні характеристики необхідно забезпечити під час реалізації. Для розроблюваної системи важливо врахувати як функціональні, так і нефункціональні вимоги, оскільки платформа повинна не лише виконувати основні бізнес-задачі, а й бути зручною, безпечною, стабільною та придатною до подальшого розвитку.

Функціональні вимоги визначають основні дії, які користувач може виконувати в системі. Оскільки вебплатформа призначена для створення AI-асистентів, обробки чатів і централізованого керування бізнесами, вона повинна підтримувати повний цикл роботи з клієнтськими зверненнями: від створення бізнес-профілю до обробки повідомлень клієнтів і передачі складних діалогів оператору.

Однією з базових функціональних вимог є реєстрація та автентифікація користувачів. Користувач повинен мати можливість увійти в систему, після чого йому має бути наданий доступ лише до тих бізнес-профілів і функцій, до яких він має відповідні права. Це необхідно для захисту бізнес-даних і коректного розмежування доступу між різними користувачами.

Наступною функціональною вимогою є створення та керування бізнес-профілями. Бізнес-профіль виступає основною робочою одиницею системи. У межах такого профілю зберігаються AI-асистенти, чати, історія повідомлень, ліди, інтеграції та налаштування. Користувач повинен мати можливість створити бізнес-профіль, переглядати його дані та керувати основними параметрами.

Важливою вимогою є створення, налаштування та редагування AI-асистентів. Користувач із відповідними правами повинен мати можливість

створити асистента, задати його назву, опис, інструкції, модель поведінки та доступні функції. Це дозволяє адаптувати AI-асистента під конкретний бізнес, його стиль комунікації та типові сценарії обробки звернень.

Система також повинна забезпечувати обробку повідомлень клієнтів через чат. Повідомлення може надходити з підключеного каналу комунікації, після чого воно повинно бути передане до backend-частини, пов'язане з відповідним бізнесом, збережене в історії діалогу та оброблене AI-асистентом. Відповідь асистента має бути повернута клієнту через той самий канал.

Окремою функціональною вимогою є збереження історії взаємодій та централізоване керування чатами. Усі повідомлення клієнта, відповіді AI-асистента, повідомлення оператора та службові події повинні зберігатися в системі. Це дозволяє працівникам бізнесу переглядати попередній контекст діалогу, контролювати стан звернення та продовжувати комунікацію без втрати інформації.

Також система повинна підтримувати передачу діалогу оператору за потреби. AI-асистент не повинен повністю замінювати людину в усіх ситуаціях. Якщо клієнт просить зв'язатися з менеджером або звернення потребує ручної обробки, система повинна змінити статус діалогу та зробити його доступним оператору підтримки. Оператор має мати можливість переглянути історію повідомлень і відповісти клієнту через CRM-панель.

Для наочного представлення основних сценаріїв взаємодії користувачів із системою було побудовано діаграму варіантів використання, яку наведено на рисунку 3.1.

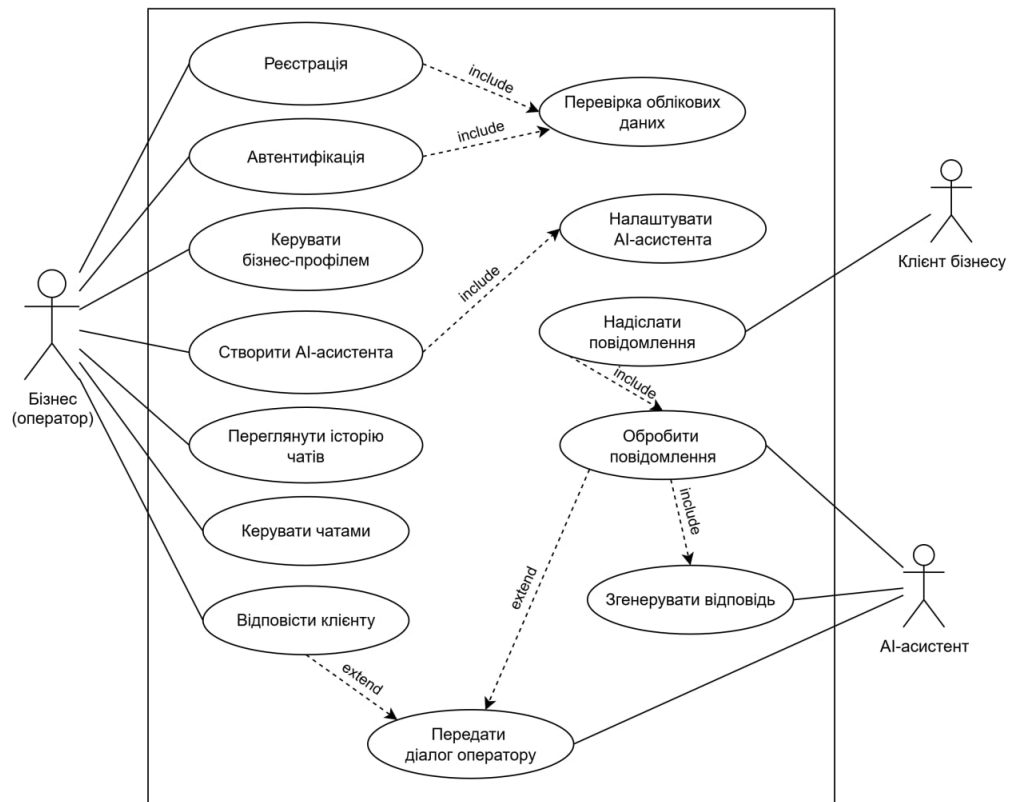


Рис. 3.1. Діаграма варіантів використання SaaS-платформи

До основних функціональних вимог системи належать:

- реєстрація та автентифікація користувачів;
- створення та керування бізнес-профілями;
- створення, налаштування та редагування AI-асистентів;
- обробка повідомлень клієнтів через чат;
- збереження історії взаємодій та централізоване керування чатами;
- передача діалогу оператору за потреби.

Нефункціональні вимоги визначають якісні характеристики системи. Вони описують не конкретні дії користувача, а те, якою повинна бути система під час роботи: швидкою, стабільною, безпечною, зручною та придатною до розширення.

Однією з головних нефункціональних вимог є продуктивність. Система повинна забезпечувати швидку обробку типових запитів. Для звичайних API-операцій час відповіді на типовий запит не повинен перевищувати 2 секунд. Водночас відповіді, які формуються за участю AI-сервісу, можуть потребувати більше часу, тому для AI-відповіді прийнятним є час до 5 секунд. Такий поділ є

доцільним, оскільки затримка AI-відповіді частково залежить від зовнішнього сервісу.

Наступною вимогою є стабільність. Доступність системи має становити не менше 99%. Це важливо, оскільки бізнес може використовувати платформу для обробки клієнтських звернень у реальному часі. Якщо система недоступна, клієнт не може отримати відповідь, а бізнес може втратити потенційне звернення або продаж.

Важливою нефункціональною вимогою є безпека. Система повинна використовувати HTTPS для захищеної передачі даних, автентифікацію користувачів, розмежування доступу та захист конфіденційної інформації через рівні доступу в базі даних. Для автентифікації користувачів використовується Auth0, що дозволяє делегувати процес входу зовнішньому провайдеру і не зберігати паролі користувачів безпосередньо в системі. Після входу користувача система повинна перевіряти його права доступу до конкретного бізнес-профілю та дозволяти виконання лише тих дій, які відповідають його ролі.

Окремо слід врахувати зручність використання. Користувач повинен мати можливість створити бізнес-профіль або AI-асистента за 3-5 кроків. Навігація в CRM-панелі повинна бути зрозумілою, а основні розділи мають бути логічно згруповані. Це важливо, оскільки система орієнтована не лише на технічних спеціалістів, а й на власників бізнесу, менеджерів та операторів підтримки.

Модульність є ще однією важливою вимогою до системи. Програмне забезпечення повинно складатися з окремих модулів, які відповідають за бізнеси, AI-асистентів, чати, інтеграції, авторизацію та інші частини системи. Такий підхід спрощує підтримку коду, дозволяє розширювати систему новими функціями та зменшує ризик впливу змін в одному модулі на інші частини платформи.

Також важливою є надійність інтеграцій. Система взаємодіє із зовнішніми сервісами, зокрема AI API, каналами комунікації, платіжною системою та сервісом авторизації. У разі тимчасової недоступності зовнішнього сервісу система повинна коректно обробляти помилку, не втрачати важливі дані та за можливості повідомляти користувача про проблему.

До основних нефункціональних вимог системи належать:

- продуктивність: час відповіді на типовий запит до 2 секунд, AI-відповідь до 5 секунд;
- стабільність: доступність системи не менше 99%;
- безпека: HTTPS, автентифікація через Auth0, розмежування доступу, захист конфіденційних даних через рівні доступу в базі даних;
- зручність використання: створення бізнес-профілю або AI-асистента за 3-5 кроків, зрозуміла навігація;
- модульність: окремі модулі для бізнесів, асистентів і чатів;
- надійність інтеграцій: коректна обробка помилок у разі недоступності зовнішніх сервісів.

Таким чином, сформовані функціональні та нефункціональні вимоги визначають основу для подальшого проектування програмного забезпечення. Функціональні вимоги описують основні можливості вебплатформи, а нефункціональні визначають якість її роботи, безпеку, стабільність і зручність використання.

### **3.2 Загальна архітектура системи**

Загальна архітектура програмного забезпечення побудована з урахуванням того, що система повинна обробляти запити користувачів CRM-панелі, приймати повідомлення з різних каналів комунікації, запускати AI-асистента, зберігати історію діалогів, працювати з лідами, перевіряти доступ користувачів і підтримувати платіжну модель. Через це система не реалізується як один простий серверний застосунок, а поділяється на клієнтську частину, API Gateway, внутрішні сервіси, базу даних, кеш і зовнішні інтеграції.

Клієнтська частина представлена вебзастосунком, який використовується як CRM-панель для бізнес-користувачів. Через неї користувач може створювати бізнес-профілі, налаштовувати AI-асистентів, переглядати чати, працювати з лідами, підключати інтеграції, переглядати операторські звернення та керувати

підпискою. Клієнтська частина не звертається напряму до внутрішніх backend-сервісів. Усі запити проходять через API Gateway.

API Gateway є єдиною публічною точкою входу до backend-частини. Він приймає HTTP-запити від клієнтської CRM-панелі та зовнішніх каналів, виконує перевірку авторизації, формує довірений контекст користувача та передає запити до відповідних внутрішніх сервісів. Такий підхід дозволяє приховати внутрішню структуру системи від зовнішнього середовища та централізувати частину логіки доступу.

Внутрішні сервіси відповідають за окремі функціональні області системи. Сервіс бізнесів відповідає за бізнес-профілі, учасників бізнесу, ліди, історію чатів, операторські звернення та платіжну модель. Сервіс асистентів відповідає за створення, редагування та збереження налаштувань AI-асистентів. Сервіс чатів відповідає за обробку повідомлень, запуск AI-асистента, збереження історії діалогу, створення лідів через AI tools і передачу звернення оператору.

Окремі інтеграційні сервіси відповідають за роботу з каналами комунікації. Telegram Service приймає webhook-повідомлення від Telegram Bot API, перевіряє їх і передає до сервісу чатів. Gmail Service використовується для email-сценаріїв і може передавати email-like повідомлення в загальний механізм обробки чатів. Widget Service відповідає за публічний web-widget, який може бути вбудований на сайт бізнесу. Усі ці сервіси працюють як адаптери каналів комунікації.

Основні дані системи зберігаються в PostgreSQL. До них належать користувачі, бізнеси, учасники бізнесів, AI-асистенти, повідомлення, чати, ліди, інтеграції, підписки та службові події. PostgreSQL використовується як надійне реляційне сховище, оскільки дані системи мають багато зв'язків і потребують контролю цілісності.

Redis використовується як допоміжне in-memory сховище. Він застосовується для кешування, pub/sub-подій, тимчасових блокувань обробки діалогів і службових механізмів. Наприклад, під час обробки повідомлення клієнта Redis може використовуватися для блокування thread, щоб уникнути одночасної обробки кількох повідомлень одного діалогу.

Зовнішні сервіси виконують окремі спеціалізовані задачі. Auth0 використовується для автентифікації користувачів. OpenAI SDK використовується для роботи AI-асистента та обробки повідомлень природною мовою. Stripe використовується для реалізації підписок, тарифів і платіжних сценаріїв. Telegram Bot API використовується для отримання й надсилання повідомлень через Telegram.

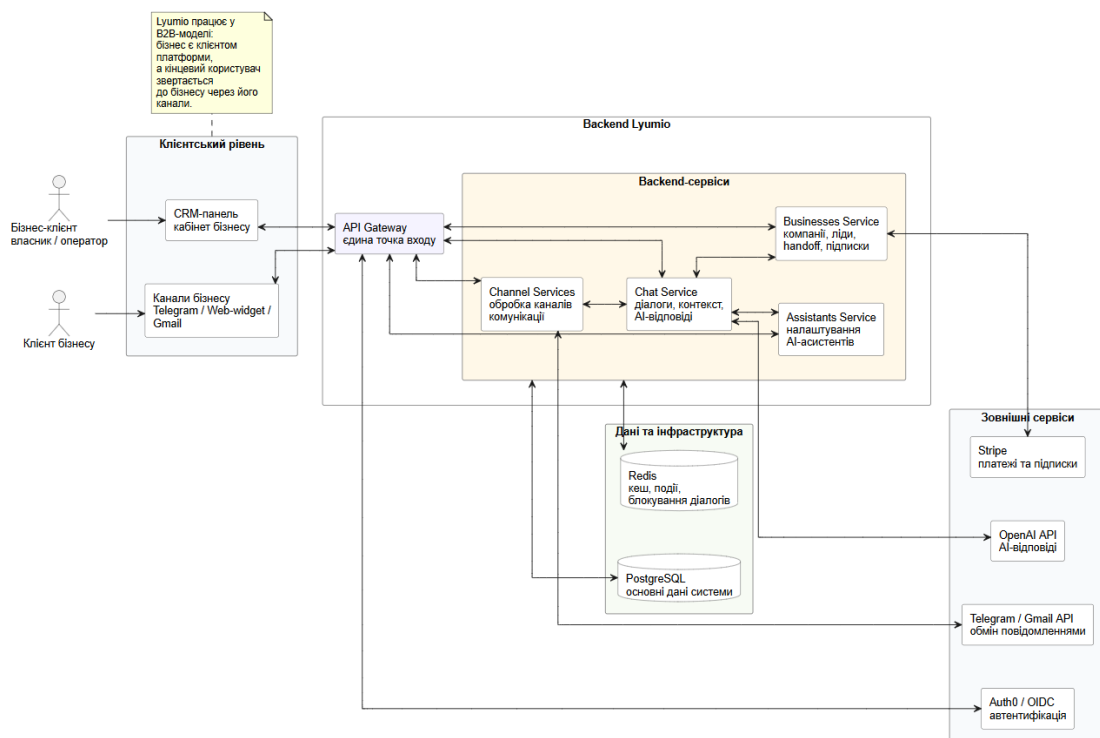


Рис. 3.2. Загальна архітектура системи

На рисунку 3.1 показано загальну архітектуру програмного забезпечення. Користувач працює з CRM-панеллю через браузер. Frontend-застосунок надсилає запити до API Gateway, який передає їх до відповідних backend-сервісів. Внутрішні сервіси взаємодіють з PostgreSQL, Redis та зовнішніми API. Повідомлення клієнтів з Telegram, web-widget або email-сценаріїв проходять через інтеграційні сервіси та потрапляють до Chat Service, де обробляються AI-асистентом.

Особливістю архітектури є розділення системи на доменні частини. Це дозволяє не змішувати логіку бізнесів, асистентів, чатів, інтеграцій і платежів в одному модулі. Наприклад, зміна логіки Telegram-інтеграції не повинна впливати

на billing-модуль, а зміна інтерфейсу CRM-панелі не повинна змінювати AI runtime. Завдяки такому підходу система стає більш зрозумілою для підтримки та подальшого розвитку.

Архітектура також враховує вимоги безпеки. Внутрішні сервіси не повинні бути доступні напряму з браузера. Зовнішній клієнт взаємодіє лише з gateway або публічними endpoint-ами каналів комунікації, які мають власні перевірки. Gateway відповідає за перевірку сесії користувача, а внутрішні сервіси отримують довірений контекст через службові заголовки.

Для багатокористувацької системи важливо, щоб кожна операція виконувалася в межах конкретного бізнес-простору. Тому більшість сутностей системи пов'язана з `business_id`. Це дозволяє обмежувати доступ до даних, відображати користувачу лише його бізнеси та уникати змішування інформації між різними клієнтами платформи.

Обробка повідомлень клієнтів відбувається через єдиний Chat Service. Незалежно від того, чи повідомлення надійшло з Telegram, widget або email-сценарію, воно приводиться до спільного формату. Після цього система визначає бізнес, канал, зовнішнього користувача, thread і передає повідомлення AI-асистенту. Такий підхід дозволяє використовувати одну AI-логіку для різних каналів комунікації.

Загальна архітектура системи дозволяє реалізувати основні функції вебплатформи: централізоване керування бізнесами, створення AI-асистентів, обробку чатів, збереження історії діалогів, створення CRM-лідів, передачу звернень оператору та підтримку платіжної моделі. При цьому система залишається придатною до розширення, оскільки нові канали або функціональні модулі можуть додаватися окремо.

### **3.3 Проєктування ролей користувачів та моделі доступу**

Однією з важливих частин проєктування веб-платформи є визначення ролей користувачів і правил доступу до даних. Оскільки система призначена для роботи

з багатьма бізнесами, необхідно забезпечити таку модель, за якої користувач може працювати лише з тими бізнес-просторами, до яких він має доступ. Це дозволяє уникнути ситуації, коли дані одного бізнесу стають доступними користувачам іншого бізнесу.

У системі використовується модель бізнес-просторів. Бізнес-простір є окремою робочою одиницею, в межах якої зберігаються AI-асистенти, чати, повідомлення, ліди, інтеграції, налаштування та дані підписки. Один користувач може мати доступ до одного або кількох бізнес-просторів. Водночас один бізнес-простір може мати кількох користувачів з різними ролями.

Для зв'язку користувача з бізнесом використовується сутність *membership*. Вона визначає, до якого бізнесу належить користувач, яку роль він має та чи є його участь активною. Завдяки цьому система може перевіряти права доступу не лише за фактом входу користувача, а й за його роллю в конкретному бізнес-просторі.

У межах системи передбачено три основні ролі користувачів: OWNER, ADMIN та OPERATOR. Кожна роль має власний рівень доступу. Такий підхід дозволяє розділити адміністративні, керівні та операторські функції.

Роль OWNER має найвищий рівень доступу. Власник бізнесу може створювати та змінювати бізнес-простір, керувати учасниками, працювати з підпискою, підключати інтеграції, створювати AI-асистентів і переглядати всі дані бізнесу. Також власник повинен мати можливість видалити бізнес або змінити критичні налаштування, які впливають на роботу всієї системи для цього бізнес-простору.

Роль ADMIN призначена для користувачів, які керують операційною частиною бізнесу, але не обов'язково є його власниками. Адміністратор може налаштовувати AI-асистентів, переглядати чати, працювати з лідами, підключати канали комунікації та керувати частиною налаштувань. При цьому доступ до деяких критичних дій, наприклад видалення бізнесу або повного керування власником, може бути обмежений.

Роль OPERATOR призначена для працівників, які безпосередньо працюють із клієнтськими зверненнями. Оператор повинен мати доступ до чатів, історії

повідомлень і звернень, які були передані людині. Його основна задача полягає в тому, щоб прийняти складний діалог, переглянути контекст і відповісти клієнту через CRM-панель. Оператор не повинен мати доступ до платіжної моделі, критичних налаштувань бізнесу або редагування AI-асистентів.

Таблиця 3.1

Ролі користувачів у системі

| Роль     | Опис                          | Основні можливості                                                                                         |
|----------|-------------------------------|------------------------------------------------------------------------------------------------------------|
| OWNER    | Власник бізнес-простору       | Повне керування бізнесом, налаштуваннями, учасниками, підпискою, інтеграціями та AI-асистентами            |
| ADMIN    | Адміністратор бізнес-простору | Керування AI-асистентами, чатами, лідами, інтеграціями та більшістю налаштувань бізнесу                    |
| OPERATOR | Оператор підтримки            | Робота з чатами, перегляд історії діалогів, прийняття звернень, переданих оператору, та відповідь клієнтам |

Модель доступу повинна враховувати не лише роль користувача, а й належність дії до конкретного бізнес-простору. Наприклад, якщо користувач є адміністратором одного бізнесу, це не означає, що він має права адміністратора в іншому бізнесі. Права завжди перевіряються в межах конкретного `business_id`.

Під час виконання запиту система повинна послідовно перевірити кілька умов. Спочатку визначається, чи користувач автентифікований. Далі перевіряється, чи існує бізнес-простір, до якого виконується запит. Після цього система перевіряє, чи має користувач активне membership у цьому бізнесі. Лише після цього перевіряється роль користувача і дозвіл на конкретну дію.

Загальний порядок перевірки доступу можна описати так:

1. Перевірити, чи користувач увійшов у систему.
2. Отримати ідентифікатор бізнес-простору з параметрів запиту або контексту.
3. Перевірити наявність активного membership користувача в цьому бізнес-просторі.
4. Визначити роль користувача.
5. Перевірити, чи дозволена дія для цієї ролі.
6. Виконати дію або повернути помилку доступу.

Такий підхід дозволяє забезпечити контроль доступу на рівні бізнес-просторів. Це особливо важливо для SaaS-системи, де кілька бізнесів використовують одну платформу, але їхні дані повинні залишатися ізольованими.

Окрему увагу потрібно приділити діям, які виконуються не з CRM-панелі, а через зовнішні канали комунікації. Наприклад, повідомлення клієнта може надійти з Telegram або web-widget. У такому випадку система не отримує звичайну користувацьку сесію з CRM-панелі. Замість цього повідомлення повинно бути пов'язане з конкретним бізнесом через інтеграцію, канал або захищений контекст. Це дозволяє правильно визначити, до якого бізнесу належить повідомлення, і не дозволяє клієнту самотійно підставити довільний `business_id`.

Для внутрішніх сервісів важливо отримувати вже перевірений контекст користувача або бізнесу. API Gateway відповідає за перевірку сесії користувача та передачу довірених службових даних у внутрішні сервіси. При цьому зовнішній клієнт не повинен мати можливість самотійно передати службові заголовки та видати себе за іншого користувача. Тому gateway повинен очищати підозрілі заголовки, які приходять ззовні, і формувати trusted headers самотійно.

У системі також необхідно враховувати доступ до платіжної моделі. Керування підпискою повинно бути доступне лише власнику або адміністратору бізнесу. Оператор не повинен мати можливість змінювати тариф, відкривати Stripe Checkout або скасовувати заплановані зміни підписки. Це обмеження потрібне для захисту фінансових дій бізнесу.

Таблиця 3.2

## Розмежування доступу до основних функцій

| Функція                                           | OWNER | ADMIN | OPERATOR |
|---------------------------------------------------|-------|-------|----------|
| Перегляд бізнес-простору                          | +     | +     | +        |
| Зміна налаштувань бізнесу                         | +     | +     | -        |
| Керування учасниками                              | +     | +     | -        |
| Створення та редагування AI-асистента             | +     | +     | -        |
| Перегляд чатів                                    | +     | +     | +        |
| Відповідь клієнту в діалозі, переданому оператору | +     | +     | +        |
| Керування лідами                                  | +     | +     | +        |
| Підключення інтеграцій                            | +     | +     | -        |
| Керування підпискою                               | +     | +     | -        |
| Видалення бізнес-простору                         | +     | -     | -        |

Умовні позначення в таблиці: «+» означає, що дія дозволена; «-» означає, що дія заборонена;

Запропонована модель доступу дозволяє розділити відповідальність між користувачами. Власник має повний контроль над бізнесом, адміністратор керує операційними процесами, а оператор працює з клієнтськими діалогами. Така структура є зрозумілою для бізнесу і водночас достатньо гнучкою для подальшого розширення.

Таким чином, модель ролей і доступу є важливою частиною проєктування системи. Вона забезпечує логічну ізоляцію бізнес-даних, захищає критичні дії, обмежує доступ до фінансових операцій і дозволяє організувати роботу кількох користувачів у межах одного бізнес-простору.

### **3.4 Проєктування механізму обробки повідомлень клієнтів**

Одним із ключових процесів у системі є обробка повідомлень клієнтів. Саме цей механізм забезпечує основну цінність вебплатформи, оскільки дозволяє приймати звернення з різних каналів комунікації, передавати їх AI-асистенту, зберігати історію діалогу, створювати ліди та за потреби передавати розмову оператору підтримки.

Повідомлення клієнта може надійти з різних джерел. У межах системи передбачено обробку звернень із Telegram, web-widget та email-сценаріїв. Кожен із цих каналів має власні технічні особливості. Telegram передає повідомлення через webhook, web-widget надсилає повідомлення через public API, а email-сценарій може використовувати окремий адаптер для передачі листів у загальний механізм чатів. Незважаючи на різницю між каналами, після первинної обробки всі повідомлення повинні надходити до єдиного сервісу обробки чатів.

Для цього використовується підхід, за якого зовнішні канали працюють як адаптери. Їхнє завдання полягає не в тому, щоб самостійно реалізовувати AI-логіку, а в тому, щоб прийняти повідомлення, перевірити його, визначити бізнес і канал, сформувати внутрішній контекст та передати повідомлення до Chat Service.

Завдяки цьому основна логіка обробки повідомлень залишається спільною для всіх каналів.

Загальний процес обробки повідомлення клієнта складається з кількох етапів. Спочатку повідомлення надходить у відповідний інтеграційний сервіс. Наприклад, якщо клієнт пише в Telegram, повідомлення приймає Telegram Service. Якщо клієнт пише через чат на сайті, повідомлення приймає Widget Service. Після цього інтеграційний сервіс визначає, до якого бізнесу та каналу належить звернення.

Наступним етапом є формування контексту діалогу. Контекст містить інформацію про платформу, бізнес, канал, зовнішнього користувача та конкретний thread. Такий підхід дозволяє системі пов'язувати нові повідомлення з уже існуючою історією спілкування. Це важливо, оскільки AI-асистент повинен відповідати не ізольовано на одне повідомлення, а з урахуванням попереднього діалогу.

У системі не передбачається, що клієнт або браузер напряму передає довільний `business_id`. Це важливо для безпеки, оскільки користувач не повинен мати можливість підставити ідентифікатор чужого бізнесу. Для зовнішніх каналів бізнес визначається через `integration`, `widget key`, `channel id` або захищений контекст, який формується backend-частиною.

Після формування контексту повідомлення передається до Chat Service. Цей сервіс відповідає за основну обробку діалогу. Він перевіряє валідність запиту, визначає thread, завантажує конфігурацію AI-асистента, отримує попередню історію повідомлень і готує дані для звернення до AI-сервісу.

Перед запуском AI-асистента система повинна перевірити, чи не перебуває діалог у режимі передачі оператору. Якщо для thread активний handoff, автоматична відповідь AI-асистента не повинна генеруватися. У такому випадку повідомлення зберігається в історії, а оператор може переглянути його в CRM-панелі та відповісти клієнту вручну. Це потрібно для того, щоб AI-асистент не заважав людині під час активної ручної обробки звернення.

Якщо handoff не активний, Chat Service формує запит до AI-асистента. До цього запиту входить повідомлення клієнта, інструкції асистента, історія діалогу,

дані про бізнес і додаткові правила поведінки. AI-асистент аналізує звернення та формує відповідь. У деяких випадках він може не лише повернути текст, а й ініціювати виконання внутрішнього інструменту.

До таких інструментів належить створення ліда. Якщо клієнт залишив контактні дані або проявив інтерес до товару чи послуги, AI-асистент може передати ці дані у відповідний backend-інструмент. Система перевіряє отримані дані, прив'язує їх до конкретного бізнесу, каналу та thread, після чого створює або оновлює CRM-лід.

Іншим важливим інструментом є передача діалогу оператору. Якщо клієнт просить людину, менеджера або оператора, AI-асистент може викликати відповідний інструмент. У результаті для thread створюється handoff-запит, статус діалогу змінюється, а оператор отримує можливість прийняти звернення в CRM-панелі.

Після формування відповіді AI-асистента результат зберігається в базі даних. У системі зберігаються повідомлення клієнта, відповідь асистента, службові події та за потреби повідомлення оператора. Збереження історії є обов'язковим, оскільки вона використовується для відображення діалогів у CRM-панелі, відновлення контексту та подальшого аналізу звернень.

Після збереження відповіді Chat Service повертає результат інтеграційному сервісу. Далі інтеграційний сервіс надсилає відповідь клієнту через той самий канал, з якого надійшло повідомлення. Наприклад, Telegram Service надсилає відповідь через Telegram Bot API, а Widget Service повертає повідомлення у web-widget.

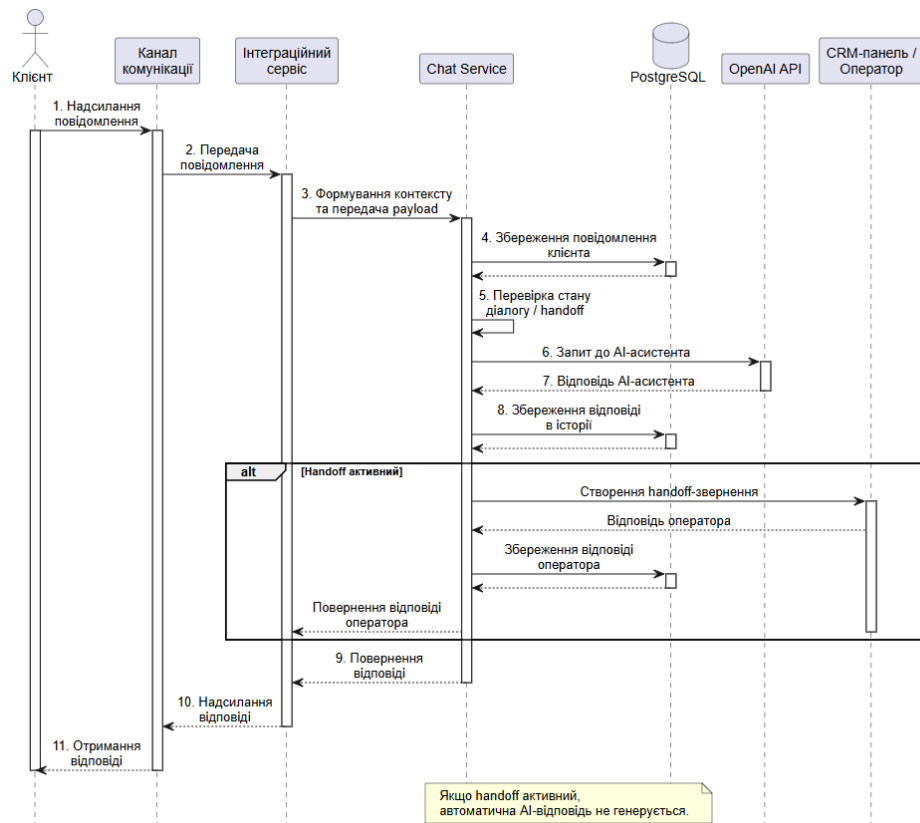


Рис. 3.3. Діаграма послідовності обробки повідомлення клієнта

Процес обробки повідомлення можна описати такою послідовністю:

1. Клієнт надсилає повідомлення через підключений канал комунікації.
2. Інтеграційний сервіс приймає повідомлення та перевіряє його коректність.
3. Система визначає бізнес, канал, зовнішнього користувача та thread.
4. Повідомлення передається до Chat Service.
5. Chat Service зберігає повідомлення клієнта в базі даних.
6. Система перевіряє, чи активна передача діалогу оператору.
7. Якщо handoff активний, автоматична відповідь AI-асистента не генерується.
8. Якщо handoff не активний, система завантажує налаштування AI-асистента.
9. AI-асистент формує відповідь або викликає внутрішній інструмент.
10. Результат обробки зберігається в історії діалогу.
11. Відповідь повертається клієнту через той самий канал комунікації.

12. За потреби CRM-панель оновлює дані про чат, лід або handoff.

Окрему увагу в механізмі обробки повідомлень потрібно приділити порядку повідомлень. Якщо клієнт швидко надсилає кілька повідомлень поспіль, система не повинна обробляти їх хаотично. Для цього може використовуватися thread lock, який тимчасово блокує паралельну обробку одного діалогу. Такий механізм дозволяє уникнути дублювання відповідей, повторного створення лідів або некоректного оновлення стану handoff.

Також важливим є збереження службових даних про повідомлення. Окрім самого тексту, система повинна зберігати роль автора повідомлення, платформу, channel id, external user id, external thread id, thread key та додаткові metadata. Це дозволяє правильно відображати повідомлення в CRM-панелі та відновлювати контекст під час наступних звернень клієнта.

У межах проєктування також передбачається, що AI-асистент не повинен самостійно визначати критичні дані системи, такі як business\_id або права доступу. Усі такі дані мають надходити з backend-контексту. Це дозволяє уникнути ситуації, коли некоректне або згенероване AI значення впливає на доступ до даних іншого бізнесу.

Механізм обробки повідомлень є центральною частиною системи, оскільки саме через нього поєднуються канали комунікації, AI-асистент, CRM-лідів, історія діалогів і операторська підтримка. Завдяки використанню спільного Chat Service система може однаково обробляти звернення з різних каналів і при цьому залишатися придатною до подальшого розширення.

### **3.5 Проєктування бази даних**

База даних є однією з ключових частин розроблюваного програмного забезпечення, оскільки саме в ній зберігаються основні дані системи: користувачі, бізнес-простори, AI-асистенти, чати, повідомлення, лідів, інтеграції, підписки та службові записи. Під час проєктування бази даних необхідно було врахувати, що

система працює з багатьма бізнесами, тому дані кожного бізнес-простору повинні бути логічно ізольовані [7].

Для реалізації бази даних використовується PostgreSQL. Такий вибір є доцільним, оскільки дані системи мають чітку структуру та багато зв'язків між сутностями. Наприклад, користувач може бути учасником кількох бізнесів, бізнес може мати AI-асистента, інтеграції, чати, ліди та підписку, а кожен чат складається з окремих повідомлень. Реляційна модель добре підходить для опису таких зв'язків.

Основною сутністю системи є бізнес-простір. Він представляє окремий бізнес, який використовує платформу для автоматизації комунікації з клієнтами. У межах бізнес-простору зберігаються налаштування, асистенти, канали комунікації, чати, ліди та інша інформація. Такий підхід дозволяє централізовано керувати всіма процесами конкретного бізнесу.

Сутність користувача зберігає локальний профіль особи, яка працює з CRM-панеллю. Оскільки автентифікація виконується через зовнішній сервіс Auth0/OIDC, у базі даних також передбачається зв'язок між зовнішнім ідентифікатором користувача та локальним записом. Це дозволяє відокремити процес входу від внутрішньої бізнес-логіки платформи.

Для зв'язку користувачів із бізнес-просторами використовується таблиця членства. Вона визначає, до якого бізнесу належить користувач, яку роль він має та чи є його доступ активним. Завдяки цьому один користувач може працювати з кількома бізнесами, а один бізнес може мати кількох користувачів з різними ролями.

AI-асистент є окремою сутністю, пов'язаною з бізнес-простором. У ній зберігаються назва, опис, інструкції, модель, налаштування поведінки та доступні інструменти. Саме ці дані використовуються під час обробки повідомлень клієнтів. Для кожного бізнесу може бути налаштований власний асистент, який відповідає специфіці діяльності компанії.

Для збереження історії комунікації використовуються сутності чатів і повідомлень. Чат представляє окремий діалог або thread, пов'язаний з конкретним бізнесом, каналом комунікації та зовнішнім користувачем. Повідомлення

зберігають текст, роль автора, платформу, ідентифікатори каналу та додаткові службові дані. Завдяки цьому система може відобразити повну історію діалогу в CRM-панелі та використовувати її як контекст для AI-асистента.

Окремою частиною бази даних є CRM-ліди. Лід містить інформацію про потенційного клієнта: ім'я, телефон, електронну пошту, короткий опис потреби, статус, джерело звернення та зв'язок із конкретним бізнесом. Лід може бути створений вручну користувачем або автоматично за допомогою AI-інструмента під час діалогу з клієнтом.

Також у базі даних зберігається інформація про передачу діалогу оператору. Для цього використовується окрема сутність handoff, яка містить статус звернення, причину передачі, коротке резюме, рівень терміновості, відповідального оператора та час прийняття або завершення звернення. Це дозволяє контролювати, які діалоги очікують відповіді людини, які вже прийняті оператором, а які були завершені.

Інтеграції з каналами комунікації також потребують збереження в базі даних. Наприклад, для Telegram-інтеграції необхідно зберігати ідентифікатор бізнесу, ім'я бота, ідентифікатор бота, зашифрований токен, статус інтеграції та службову інформацію про webhook. Для web-widget зберігається ключ віджета, статус, дозволені домени, привітальне повідомлення та налаштування зовнішнього вигляду.

Платіжна модель також відображається в структурі бази даних. Для цього передбачено таблиці тарифних планів, підписок бізнесів, Stripe-подій та billing audit log. Тарифні плани містять код, назву, опис, вартість, ліміти та доступні функції. Підписка бізнесу містить інформацію про активний план, статус, період дії, Stripe customer id, Stripe subscription id та можливі заплановані зміни тарифу.

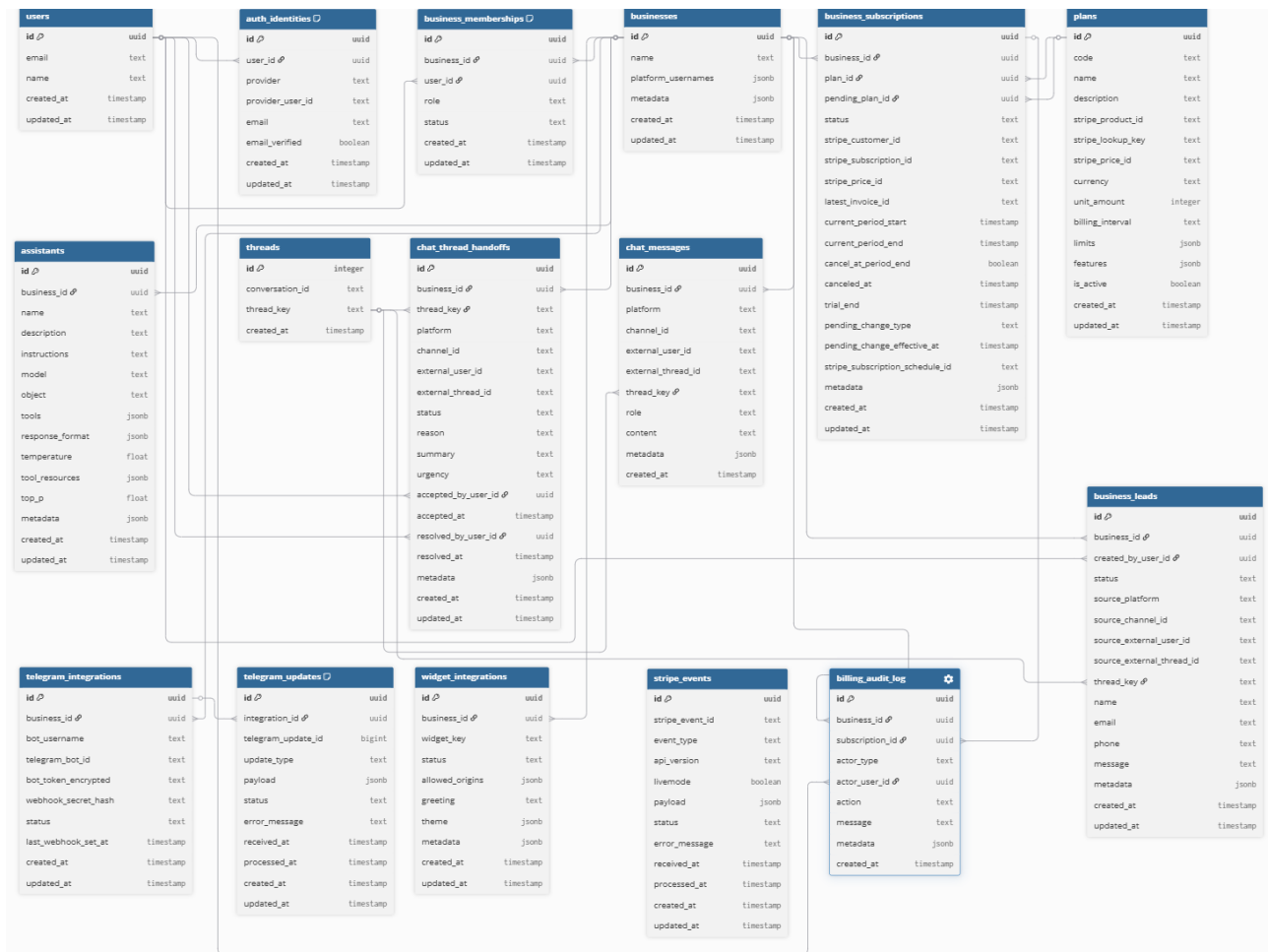


Рис. 3.4. Схема бази даних

Основні таблиці бази даних:

1. users - зберігає локальні профілі користувачів системи.
2. auth\_identities - зберігає зв'язок між зовнішнім провайдером авторизації та локальним користувачем.
3. businesses - зберігає бізнес-простори.
4. business\_memberships - зберігає ролі користувачів у бізнес-просторах.
5. assistants - зберігає налаштування AI-асистентів.
6. chat\_messages - зберігає історію повідомлень у діалогах.
7. business\_leads - зберігає CRM-ліди.
8. chat\_thread\_handoffs - зберігає стан передачі діалогу оператору.
9. telegram\_integrations - зберігає дані Telegram-інтеграцій.
10. widget\_integrations - зберігає дані web-widget інтеграцій.

11. plans - зберігає тарифні плани.
12. business\_subscriptions - зберігає підписки бізнесів.
13. stripe\_events - зберігає оброблені Stripe webhook-події.
14. billing\_audit\_log - зберігає журнал платіжних дій.

Важливою вимогою до бази даних є підтримка логічної ізоляції даних. Для цього більшість таблиць, які містять бізнес-дані, мають поле `business_id`. Це дозволяє виконувати запити лише в межах конкретного бізнес-простору та перевіряти доступ користувача через `membership`. Такий підхід є основою безпеки для багатокористувацької SaaS-системи.

Для таблиці повідомлень важливо зберігати не лише текст, а й службові поля. До таких полів належать `platform`, `channel_id`, `external_user_id`, `external_thread_id`, `thread_key`, `role`, `content`, `metadata` та `created_at`. Завдяки цьому система може правильно групувати повідомлення в діалоги, розділяти канали комунікації та відображати історію в CRM-панелі.

Для таблиці лідів важливими є поля, які дозволяють визначити джерело звернення. Наприклад, `source_platform`, `source_channel_id`, `source_external_user_id`, `source_external_thread_id` і `thread_key`. Це дозволяє зв'язати лід із конкретним діалогом і зрозуміти, з якого каналу він був отриманий.

Для handoff-сценаріїв важливо зберігати статус звернення. Статус може показувати, що діалог очікує оператора, вже прийнятий людиною або завершений. Завдяки цьому CRM-панель може відображати актуальний стан звернення, а Chat Service може визначити, чи потрібно генерувати AI-відповідь, чи діалог уже обробляється оператором.

Під час проєктування бази даних також було враховано необхідність збереження службових подій платіжної системи. Stripe може повторно надсилати webhook-події, тому в системі потрібно зберігати ідентифікатори оброблених подій. Це дозволяє уникнути дублювання обробки та некоректного оновлення підписки.

Таким чином, спроектована структура бази даних забезпечує збереження основних сутностей вебплатформи, підтримує зв'язки між користувачами,

бізнесами, асистентами, чатами, лідами, інтеграціями та підписками. Використання `business_id` і `membership` дозволяє реалізувати логічну ізоляцію даних, а збереження службових полів забезпечує коректну роботу багатоканальної комунікації та AI-обробки повідомлень.

### 3.6 Проєктування інтеграцій з каналами комунікації

Однією з важливих вимог до розроблюваного програмного забезпечення є підтримка різних каналів комунікації. Бізнес може отримувати звернення не лише через один інтерфейс, а з кількох джерел: месенджерів, електронної пошти або чату, вбудованого на сайт. Тому під час проєктування системи було передбачено інтеграційний підхід, за якого кожен канал комунікації має окремий адаптер, але всі повідомлення після первинної обробки передаються в єдиний механізм роботи з чатами.

Основна ідея такого підходу полягає в тому, що Telegram, web-widget та email-сценарії не повинні мати власної окремої логіки AI-обробки. Кожен канал відповідає лише за свою платформну частину: отримання повідомлення, перевірку запиту, визначення бізнесу, формування внутрішнього контексту та передачу повідомлення до Chat Service. Основна логіка обробки діалогу, запуск AI-асистента, створення лідів, збереження історії та передача оператору виконуються централізовано.

Такий підхід дозволяє уникнути дублювання коду. Якщо AI-логіка була б реалізована окремо для Telegram, окремо для widget і окремо для email, система стала б складнішою для підтримки. Будь-яка зміна в роботі AI-асистента потребувала б оновлення кількох різних модулів. Натомість у спроектованій архітектурі всі канали передають повідомлення до спільного сервісу чатів, тому основна логіка залишається єдиною.

Telegram-інтеграція призначена для приймання повідомлень від користувачів через Telegram-бота. Для кожного бізнесу може бути підключений власний бот або інтеграція, яка пов'язується з конкретним бізнес-простором. Після підключення

Telegram-бота система реєструє webhook, через який Telegram надсилає повідомлення на backend. Отримане повідомлення перевіряється, після чого система визначає відповідний бізнес, канал та зовнішнього користувача.

Під час проєктування Telegram-інтеграції важливо врахувати безпеку webhook-запитів. Система повинна перевіряти секретний токен, щоб переконатися, що запит надійшов від коректного джерела. Також необхідно зберігати інформацію про оброблені Telegram updates, щоб уникнути повторної обробки одного й того самого повідомлення. Це важливо, оскільки зовнішні сервіси можуть повторно надсилати події у разі помилки або затримки.

Web-widget є каналом комунікації, який дозволяє бізнесу вбудувати чат на власний сайт. Для клієнта це зручно, оскільки він може поставити питання без переходу в месенджер або електронну пошту. Для бізнесу widget є способом швидко підключити AI-асистента до сайту та отримувати звернення від відвідувачів.

Під час проєктування widget-інтеграції важливо врахувати, що widget key є публічним, оскільки він використовується у коді сайту. Тому безпека не повинна будуватися лише на секретності цього ключа. Backend повинен самостійно визначати бізнес за widget key, перевіряти дозволені домени та формувати захищений контекст сесії. Клієнтський браузер не повинен мати можливість самостійно передати довільний business\_id.

Для widget також важливо зберігати сесію відвідувача. Якщо користувач повертається до чату або надсилає кілька повідомлень у межах одного звернення, система повинна пов'язати ці повідомлення з одним thread. Це дозволяє відображати повну історію діалогу в CRM-панелі та надавати AI-асистенту контекст попередніх повідомлень.

Email-сценарій у системі розглядається як ще один канал комунікації. Його основна задача полягає в тому, щоб привести email-повідомлення до спільного формату чату. У такому випадку адреса поштової скриньки або Gmail thread id можуть використовуватися для визначення каналу та конкретного діалогу. Це

дозволяє обробляти email-like повідомлення через той самий Chat Service, що й Telegram або web-widget.

У межах подальшого розвитку email-інтеграція може бути розширена до production-рівня. Для цього потрібно реалізувати OAuth-підключення поштової скриньки, збереження refresh token у захищеному вигляді, отримання подій через Gmail API або Google Pub/Sub, дедуплікацію листів та відправлення відповідей через поштовий API. Проте навіть на етапі проєктування важливо, щоб email-канал залишався адаптером, а не окремою системою AI-обробки.

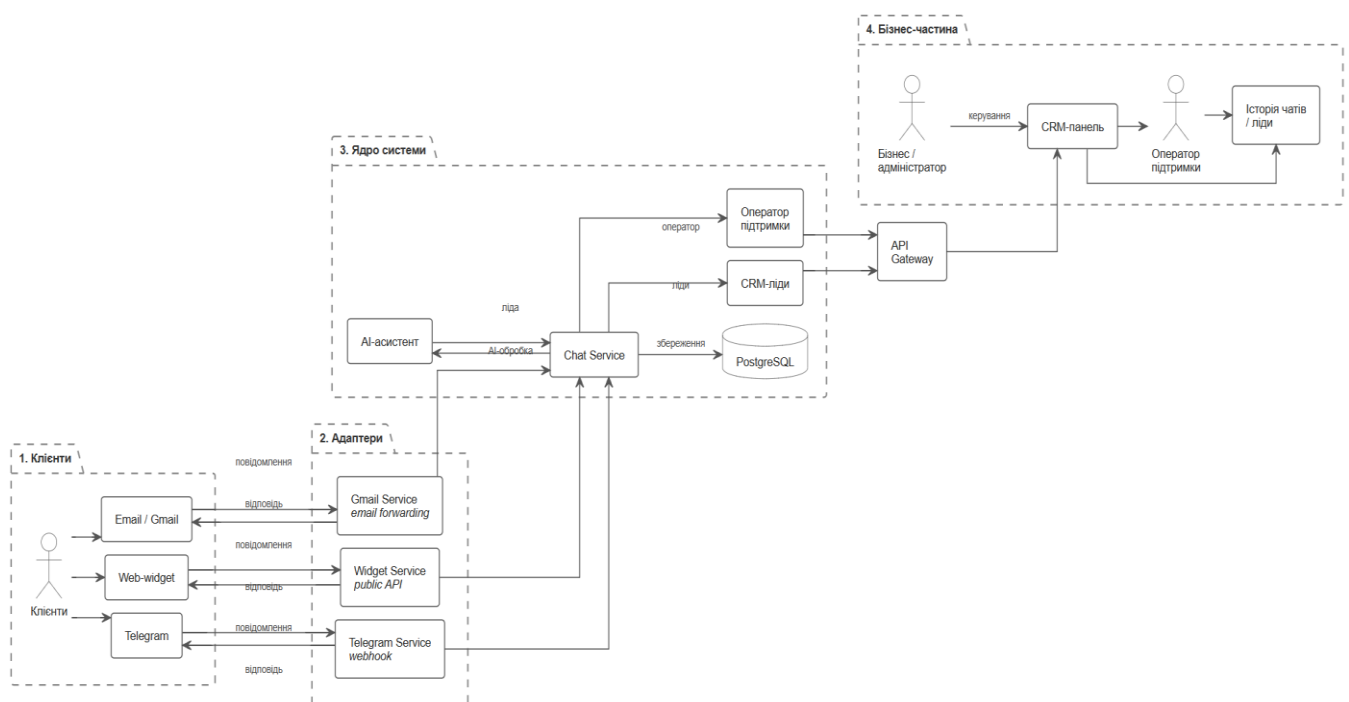


Рис. 3.5. Схема інтеграції каналів комунікації

Загальний механізм інтеграції каналів можна описати так:

1. Клієнт надсилає повідомлення через Telegram, web-widget або email-канал.
2. Відповідний інтеграційний сервіс приймає повідомлення.
3. Сервіс перевіряє коректність запиту та визначає бізнес-простір.
4. Система формує внутрішній контекст повідомлення.
5. Повідомлення передається до Chat Service.
6. Chat Service виконує основну обробку діалогу.
7. Відповідь повертається в інтеграційний сервіс.

8. Інтеграційний сервіс надсилає відповідь клієнту через початковий канал.

Перевагою такого підходу є те, що система залишається розширюваною. Якщо в майбутньому потрібно буде додати новий канал, наприклад WhatsApp, Instagram або інший месенджер, достатньо створити новий адаптер. Він повинен буде приймати повідомлення зі свого середовища, формувати стандартний контекст і передавати дані до Chat Service. Основна логіка AI-асистента, лідів, історії чатів і операторської передачі при цьому не змінюється.

Окремо слід зазначити, що всі канали повинні працювати з єдиною моделлю thread. Thread дозволяє групувати повідомлення одного клієнта в межах конкретного бізнесу, каналу та зовнішнього користувача. Завдяки цьому система може зберігати історію звернень, правильно відображати чати в CRM-панелі та не змішувати діалоги між різними клієнтами або бізнесами.

Також важливо, щоб інтеграційні сервіси не приймали критичні дані напрямку від клієнта без перевірки. Наприклад, browser у web-widget не повинен самостійно вказувати `business_id`, а Telegram-користувач не повинен мати можливість впливати на те, до якого бізнесу буде прив'язане повідомлення. Визначення бізнесу повинно виконуватися backend-частиною на основі integration, widget key, webhook route або іншого захищеного механізму.

Проектування інтеграцій з каналами комунікації дозволяє зробити систему зручною для бізнесу та клієнтів. Бізнес може підключити кілька каналів, а клієнт може звертатися через той сервіс, який йому зручніший. Водночас для операторів і адміністраторів усі звернення відображаються в єдиній CRM-панелі, що спрощує контроль і подальшу роботу з клієнтами.

Таким чином, інтеграції з каналами комунікації спроектовані за адаптерним підходом. Telegram, web-widget та email-сценарії відповідають за приймання й доставку повідомлень, а основна бізнес-логіка обробки чатів залишається в Chat Service. Це забезпечує єдиний механізм роботи з діалогами, спрощує підтримку системи та створює основу для додавання нових каналів у майбутньому.

## 4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Організація проєкту та структура програмного забезпечення

Реалізація програмного забезпечення виконувалася з урахуванням вимог і архітектурних рішень, визначених у попередньому розділі. Основним завданням на етапі реалізації було створення вебплатформи, яка дозволяє керувати бізнесами, налаштовувати AI-асистентів, обробляти чати, зберігати історію повідомлень, працювати з CRM-лідами, передавати складні діалоги оператору та підтримувати інтеграції з каналами комунікації.

Проєкт було організовано як систему з окремою backend-частиною та frontend-застосунком. Такий підхід дозволяє незалежно розвивати серверну логіку та клієнтський інтерфейс. Backend відповідає за обробку запитів, бізнес-логіку, роботу з базою даних, AI-інтеграцію, авторизацію, інтеграції з каналами комунікації та платіжну модель. Frontend відповідає за CRM-панель, через яку користувач працює з бізнесами, асистентами, чатами, лідами, інтеграціями та підпискою.

Структура проєкту поділена на два основні репозиторії: backend і frontend. Backend-частина містить API Gateway, внутрішні сервіси, shared library, міграції бази даних, Docker-конфігурації, тести та службові файли. Frontend-частина містить Next.js-застосунок, компоненти інтерфейсу, сторінки CRM-панелі, API layer, React Query hooks, форми та тести клієнтської частини.

Backend-частина реалізована за модульним підходом. У ній передбачено кілька сервісів, кожен з яких відповідає за окрему функціональну область. Така структура дозволяє не змішувати логіку бізнесів, AI-асистентів, чатів, інтеграцій і платіжної моделі в одному великому серверному застосунку.

Основні частини backend-репозиторію:

1. gateway - єдина публічна точка входу до backend-частини.
2. services/assistants - сервіс для керування AI-асистентами.

3. services/businesses - сервіс для роботи з бізнесами, лідами, handoff-зверненнями та billing-модулем.

4. services/chat - сервіс обробки повідомлень і запуску AI-асистента.

5. services/telegram - сервіс інтеграції з Telegram.

6. services/gmail - сервіс для email-сценаріїв.

7. services/widget - сервіс для web-widget каналу.

8. shared-lib - спільна бібліотека для backend-сервісів.

9. db/migrations - міграції бази даних.

10. docker-compose файли - конфігурації для запуску системи в контейнерах.

Shared library використовується для винесення спільної логіки, яка потрібна різним backend-сервісам. До неї можуть входити модулі для роботи з базою даних, логуванням, змінними середовища, авторизацією, валідацією, readiness endpoints, OpenAI client, кешем, billing-перевірками та службовими типами. Завдяки цьому однакова логіка не дублюється в кожному сервісі.

Frontend-частина реалізована як Next.js-застосунок. Вона містить сторінки та функціональні розділи CRM-панелі. Основними розділами інтерфейсу є Dashboard, Chats, Assistants, Leads, Handoffs, Integrations, Businesses, Billing та Settings. Така структура відповідає основним сценаріям роботи користувача в системі.

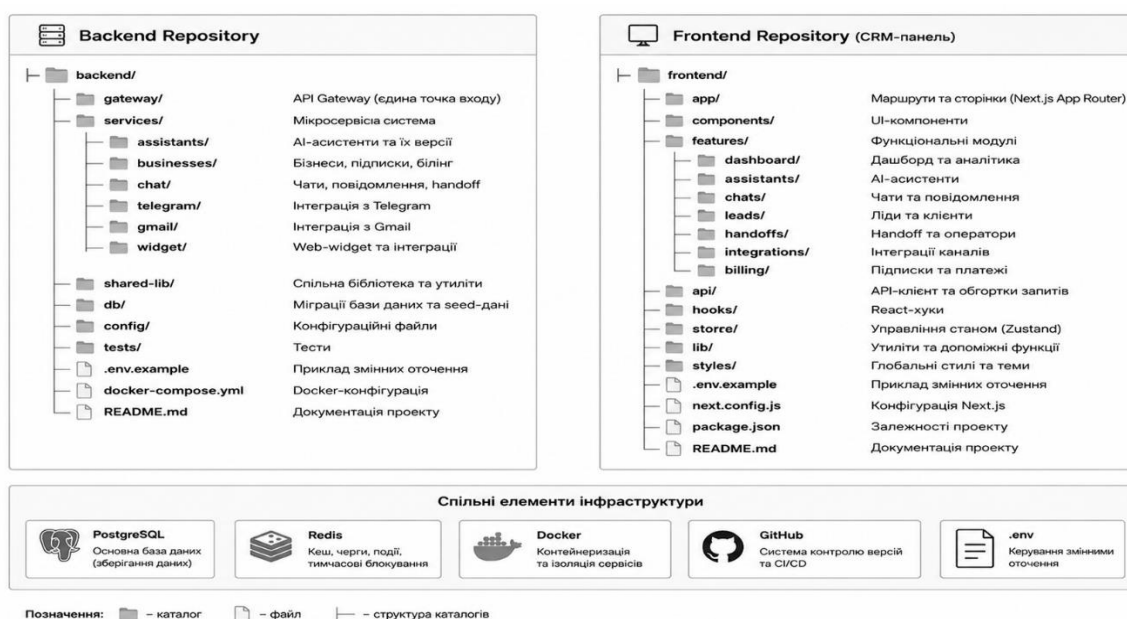


Рис. 4.1. Структура програмного забезпечення

Клієнтська частина взаємодіє з backend через API Gateway. Для запитів використовується єдиний API layer, який відповідає за відправлення запитів, обробку помилок, передачу cookies та типізацію відповідей. Для роботи із серверними даними використовується React Query, що дозволяє кешувати відповіді, оновлювати дані після змін і керувати станами завантаження.

Організація проєкту у вигляді окремих сервісів і клієнтського застосунку спрощує подальшу підтримку системи. Якщо необхідно змінити логіку Telegram-інтеграції, це можна зробити в окремому сервісі, не змінюючи CRM-панель або billing-модуль. Якщо потрібно оновити інтерфейс сторінки лідів, зміни виконуються у frontend-частині без прямого втручання в Chat Service.

Також така структура зручна для розгортання. Кожен сервіс може запускатися в окремому Docker-контейнері. Це дозволяє ізолювати середовище виконання, налаштувати порти, змінні середовища, healthchecks і залежності між компонентами. PostgreSQL і Redis також запускаються як окремі сервіси інфраструктури.

Отже, організація проєкту побудована так, щоб розділити відповідальність між основними частинами системи. Backend реалізує бізнес-логіку та інтеграції, frontend забезпечує користувацький інтерфейс, shared library містить спільні модулі, а Docker-конфігурація дозволяє запускати систему в ізольованому середовищі.

## **4.2 Реалізація API Gateway та внутрішніх сервісів**

API Gateway є центральною точкою входу до backend-частини системи. Він приймає запити від frontend-застосунку, обробляє auth routes, перевіряє сесію користувача та проксирує запити до відповідних внутрішніх сервісів. Такий підхід дозволяє не відкривати всі backend-сервіси напряму для зовнішнього доступу.

Gateway виконує кілька важливих функцій. По-перше, він обслуговує маршрути авторизації. Через нього користувач може перейти до входу, зареєструватися, отримати інформацію про поточну сесію або вийти з системи. По-

друге, gateway виступає проксі для внутрішніх сервісів. Наприклад, запити до асистентів передаються в assistants service, запити до бізнесів і billing-модуля передаються в businesses service, а запити до Telegram, Gmail або Widget передаються у відповідні інтеграційні сервіси.

Основні маршрути API Gateway:

1. GET /health - перевірка доступності gateway.
2. GET /ready - перевірка готовності gateway та підключених сервісів.
3. GET /auth/login - запуск процесу входу користувача.
4. GET /auth/signup - запуск процесу реєстрації.
5. GET /auth/me - отримання інформації про поточного користувача.
6. POST /auth/logout - вихід користувача із системи.

Крім auth routes, gateway проксирує запити до внутрішніх сервісів. Наприклад, маршрут '/assistants' передається до сервісу асистентів, '/businesses' до сервісу бізнесів, '/billing' до billing-модуля, '/telegram' до Telegram Service, '/chat' до Chat Service, '/gmail' до Gmail Service, а '/widget' до Widget Service. Наявність gateway дозволяє frontend-застосунку працювати з одним backend endpoint, не знаючи внутрішньої структури сервісів.

Важливою частиною реалізації gateway є робота з авторизацією. Після входу користувача через Auth0 система створює локальну сесію, яка зберігається в httpOnly cookie. Frontend не зберігає access token у localStorage, що зменшує ризики безпеки. Для перевірки стану сесії frontend звертається до '/auth/me'.

Після перевірки користувача gateway може передати у внутрішні сервіси довірений контекст. Такий контекст містить інформацію про користувача, наприклад його ідентифікатор, email або ім'я. При цьому gateway повинен очищати службові заголовки, які могли прийти від зовнішнього клієнта, і самостійно формувати trusted headers. Це потрібно, щоб користувач не міг підробити внутрішній auth context.

Assistants Service відповідає за роботу з AI-асистентами. У ньому реалізовано маршрути для отримання списку асистентів, створення асистента, перегляду конкретного асистента, оновлення налаштувань і видалення. Асистент зберігає

назву, опис, інструкції, модель, параметри поведінки, доступні інструменти та зв'язок із бізнесом.

Під час створення або редагування AI-асистента система перевіряє права користувача. Оператор не повинен мати можливість змінювати налаштування асистента. Такі дії дозволені власнику або адміністратору бізнесу. Це відповідає моделі доступу, описаній у попередньому розділі.

Businesses Service відповідає за основний SaaS-домен системи. У ньому реалізовано роботу з бізнес-просторами, учасниками, лідами, історією чатів, handoff-зверненнями, інтеграціями та billing-модулем. Саме цей сервіс містить значну частину бізнес-логіки, пов'язаної з керуванням компанією в межах платформи.

Для бізнес-просторів реалізовано базові CRUD-операції: створення, перегляд, оновлення та видалення. Після створення бізнесу користувач отримує роль власника. Надалі він може працювати з цим бізнесом, переглядати пов'язані дані та керувати налаштуваннями відповідно до своєї ролі.

У Businesses Service також реалізовано роботу з CRM-лідами. Ліди можуть створюватися вручну через CRM-панель або автоматично через AI tool під час діалогу з клієнтом. Для ліда зберігаються контактні дані, статус, джерело звернення, thread key та інша службова інформація. Це дозволяє менеджеру продовжити роботу з потенційним клієнтом після первинної взаємодії.

Окремою частиною Businesses Service є handoff-механізм. Якщо AI-асистент або клієнт ініціює передачу діалогу оператору, у системі створюється handoff-запит. Оператор може переглянути такий запит у CRM-панелі, прийняти його, відповісти клієнту та завершити звернення. Поки handoff активний, автоматичні відповіді AI-асистента повинні бути призупинені.

Chat Service відповідає за обробку повідомлень клієнтів. Він приймає повідомлення з інтеграційних сервісів, визначає контекст, завантажує налаштування AI-асистента, перевіряє handoff-стан, звертається до AI-сервісу, зберігає повідомлення та повертає відповідь у канал комунікації. Це один із

ключових сервісів системи, оскільки саме він об'єднує AI runtime, історію діалогів, CRM-ліди та передачу оператору.

Telegram Service реалізує інтеграцію з Telegram Bot API. Він приймає webhook-повідомлення, перевіряє secret token, виконує дедуплікацію updates і передає текстові повідомлення до Chat Service. Після отримання відповіді від Chat Service сервіс надсилає її користувачу через Telegram.

Gmail Service використовується для email-сценаріїв. У поточній реалізації він може працювати через mock-forwarding endpoint, який дозволяє передавати email-like повідомлення до Chat Service. Це потрібно для перевірки загального AI workflow для поштових звернень і підготовки до подальшого production-підключення Gmail.

Widget Service відповідає за web-widget канал. Він надає public endpoints для embed script, конфігурації, створення сесії, надсилання повідомлення та отримання історії. Widget дозволяє бізнесу вбудувати чат на власний сайт, а повідомлення від відвідувачів передавати в загальний механізм обробки чатів.

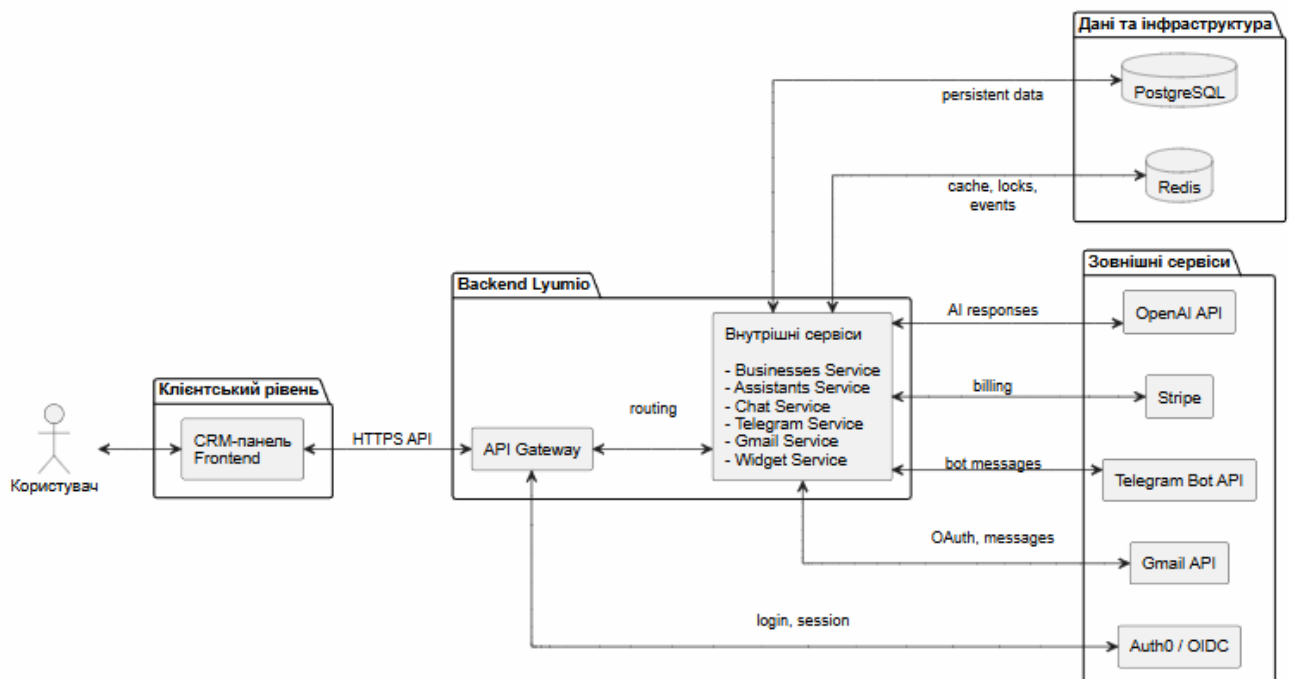


Рис. 4.2. Реалізація API Gateway та внутрішніх сервісів

Усі внутрішні сервіси мають власну відповідальність, але працюють як частини однієї системи. Gateway приймає зовнішні запити, внутрішні сервіси виконують доменну логіку, PostgreSQL зберігає основні дані, Redis використовується для кешування та службових механізмів, а зовнішні API забезпечують авторизацію, AI-функціональність, оплату та комунікацію з платформами.

Таким чином, реалізація API Gateway та внутрішніх сервісів дозволила розділити систему на окремі функціональні частини. Це спрощує підтримку коду, підвищує безпеку, дозволяє приховати внутрішні сервіси від зовнішнього доступу та створює основу для подальшого розширення платформи.

### **4.3 Реалізація CRM-панелі та клієнтського інтерфейсу**

Клієнтська частина програмного забезпечення реалізована у вигляді CRM-панелі, через яку користувач взаємодіє з основними можливостями системи. Саме через цей інтерфейс користувач може керувати бізнес-просторами, створювати та налаштовувати AI-асистентів, переглядати чати, працювати з лідами, обробляти звернення, передані оператору, підключати інтеграції та керувати підпискою.

Основною вимогою до CRM-панелі була зручність використання. Система орієнтована на бізнес-користувачів, тому інтерфейс повинен бути зрозумілим не лише для розробників або технічних спеціалістів, а й для власників бізнесу, менеджерів та операторів підтримки. Через це основні функції були розподілені за окремими розділами, щоб користувач міг швидко знайти потрібну дію.

Клієнтська частина побудована на основі Next.js та React. Next.js використовується для організації сторінок, маршрутизації та структури застосунку. React використовується для створення компонентів інтерфейсу. Такий підхід дозволяє розділити інтерфейс на незалежні частини, які можна повторно використовувати на різних сторінках CRM-панелі.

Основними розділами CRM-панелі є:

1. Dashboard.

2. Businesses.
3. Assistants.
4. Chats.
5. Leads.
6. Handoffs.
7. Integrations.
8. Billing.
9. Settings.

Розділ Dashboard призначений для загального огляду стану бізнесу. На цій сторінці користувач може бачити ключову інформацію про активний бізнес-простір, кількість чатів, лідів, активних звернень або підключених інтеграцій. Такий розділ є зручним стартовим екраном після входу в систему, оскільки дозволяє швидко оцінити поточний стан роботи бізнесу.

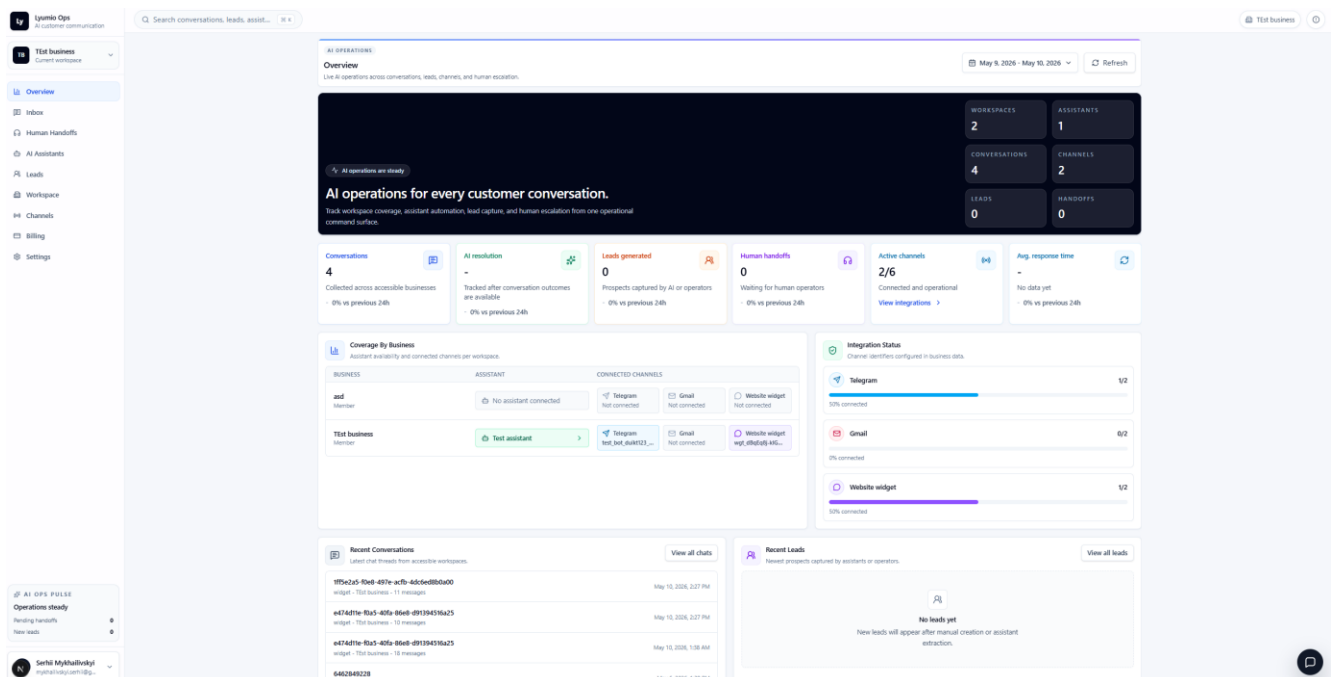


Рис. 4.3. Головна сторінка CRM-панелі

Розділ Businesses використовується для роботи з бізнес-просторами. Користувач може переглядати доступні йому бізнеси, створювати новий бізнес-простір або перемикатися між ними. Такий підхід є важливим для

багатокористувацької системи, оскільки один користувач може мати доступ до кількох бізнесів. При цьому всі запити до backend виконуються в контексті конкретного business\_id.

Розділ Assistants призначений для створення та налаштування AI-асистентів. У ньому користувач може створити нового асистента, вказати його назву, опис, інструкції, модель поведінки та інші параметри. Ці дані використовуються backend-частиною під час обробки повідомлень клієнтів. Через цю сторінку бізнес може адаптувати AI-асистента під власну сферу діяльності, стиль комунікації та типові сценарії звернень.

Під час реалізації сторінки асистентів важливо було зробити форму налаштування зрозумілою. Користувач не повинен працювати напряму з технічними параметрами AI runtime. Він має бачити прості поля: назву, опис, інструкції та додаткові налаштування. Технічна частина обробляється backend-сервісами, а CRM-панель надає зручний інтерфейс для введення цих даних.

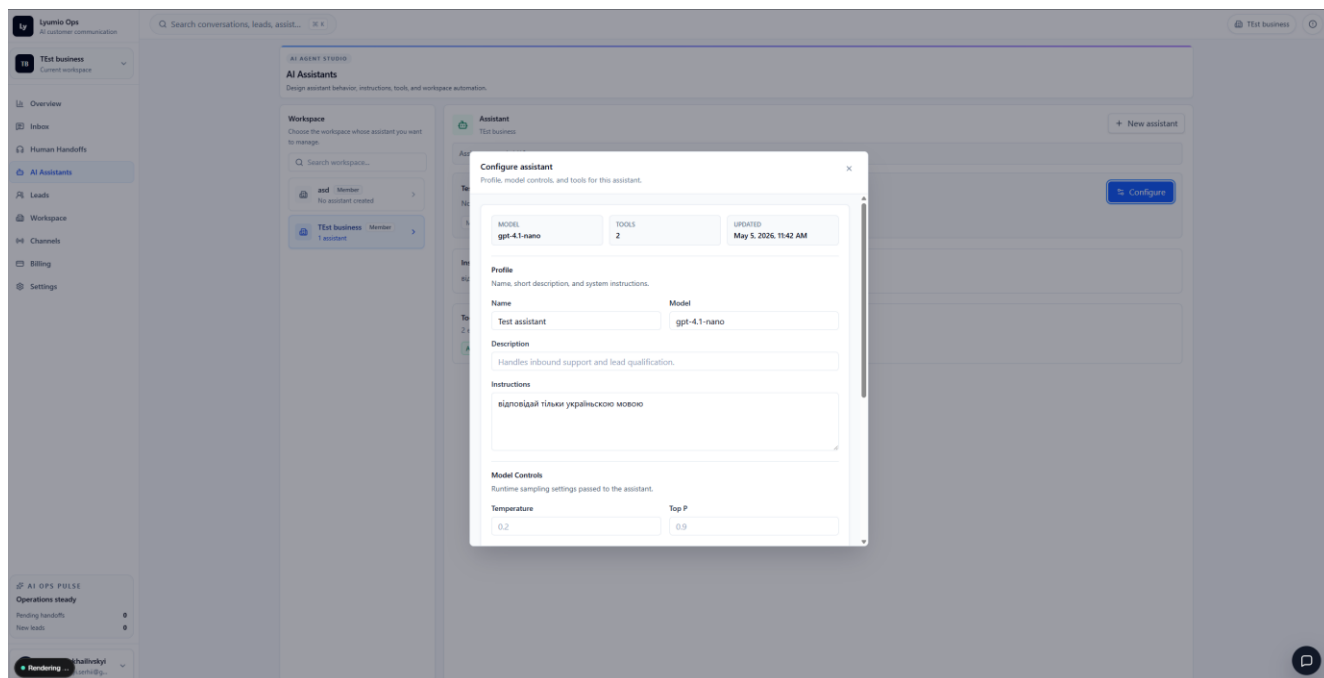


Рис. 4.4. Сторінка створення та налаштування AI-асистента

Розділ Chats є одним із ключових у CRM-панелі. Він дозволяє переглядати клієнтські діалоги, історію повідомлень, канал комунікації, статус звернення та

останню активність. У цьому розділі користувач може бачити, з якого каналу надійшло повідомлення, хто є клієнтом, які повідомлення вже були надіслані та чи потребує діалог участі оператора.

Для сторінки чатів важливою є зручна структура інтерфейсу. Зазвичай вона складається зі списку діалогів та області перегляду конкретного чату. У списку діалогів користувач бачить коротку інформацію про кожен чат, а при виборі конкретного діалогу відкривається історія повідомлень. Такий підхід схожий на звичні інтерфейси месенджерів, тому користувачу легше орієнтуватися в системі.

Повідомлення в чаті можуть мати різні ролі. Це може бути повідомлення клієнта, відповідь AI-асистента, відповідь оператора або службова подія. В інтерфейсі ці повідомлення повинні відображатися зрозуміло, щоб оператор міг швидко визначити, хто саме писав і на якому етапі знаходиться діалог.

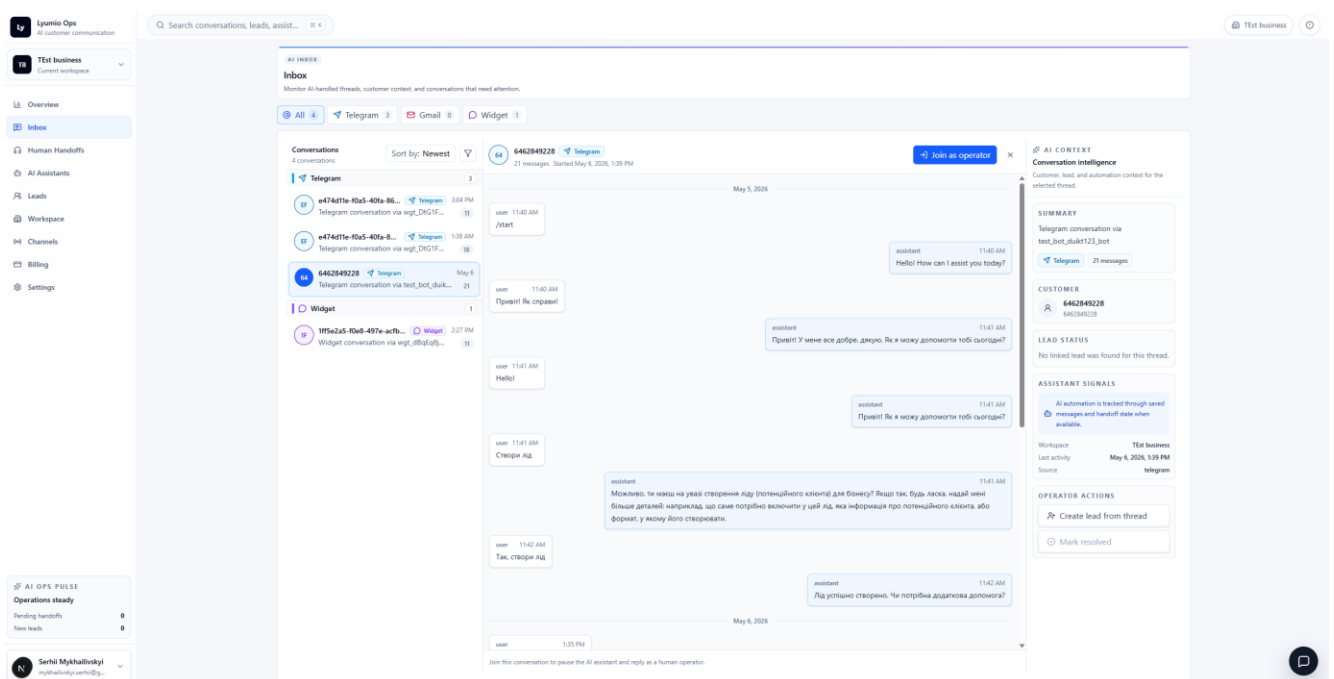


Рис. 4.5. Сторінка перегляду чатів

Розділ Leads призначений для роботи з потенційними клієнтами. Лід може бути створений автоматично під час діалогу з AI-асистентом або вручну через CRM-панель. У списку лідів відображаються контактні дані, статус, джерело

звернення, короткий опис потреби та дата створення. Це дозволяє менеджеру швидко перейти від первинного звернення до подальшої роботи з клієнтом.

Для лідів важливою є можливість зміни статусу. Наприклад, лід може бути новим, у роботі, завершеним або відхиленим. Зміна статусу допомагає відстежувати процес обробки звернення та не втрачати потенційних клієнтів. Якщо лід був створений на основі чату, у CRM-панелі доцільно зберігати зв'язок між лідом і відповідним діалогом.

Розділ Handoffs використовується для роботи зі зверненнями, які були передані оператору. Якщо AI-асистент визначив, що потрібна участь людини, або клієнт прямо попросив менеджера, система створює handoff-звернення. У CRM-панелі оператор може переглянути такі звернення, прийняти діалог, ознайомитися з історією повідомлень і відповісти клієнту.

Цей розділ є важливим, оскільки автоматизація не повинна повністю замінювати людину. У складних або нестандартних ситуаціях клієнт повинен мати можливість отримати відповідь від оператора. Тому CRM-панель повинна показувати оператору всі необхідні дані: історію діалогу, причину передачі, статус звернення та можливість відповіді.

Розділ Integrations призначений для підключення каналів комунікації. Через нього користувач може налаштовувати Telegram-інтеграцію, web-widget або інші канали. Для кожної інтеграції відображається її статус, основні параметри та доступні дії. Наприклад, для Telegram може бути показано стан підключення бота, а для widget - ключ віджета та дозволені домени.

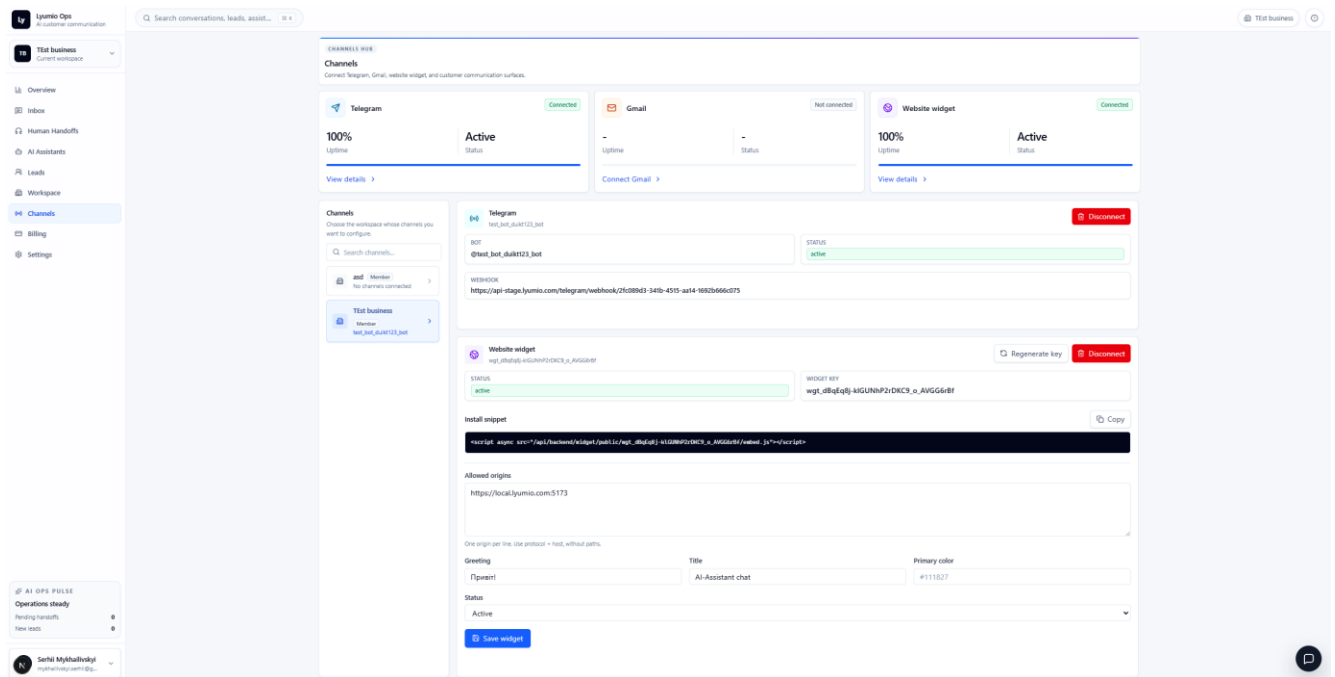


Рис. 4.6. Сторінка інтеграцій

Розділ Billing використовується для роботи з підпискою. Користувач з відповідними правами може переглядати активний тариф, доступні плани, статус підписки та виконувати дії, пов'язані з оплатою. Наприклад, система може дозволяти перейти до Stripe Checkout, відкрити Customer Portal або змінити тарифний план.

Для billing-розділу важливо, щоб користувач чітко бачив поточний стан підписки. Якщо план активний, система повинна показувати його назву, період дії та основні доступні можливості. Якщо є запланована зміна тарифу, вона також повинна бути відображена. Це зменшує кількість непорозумінь і робить платіжну модель прозорішою для бізнесу.

Розділ Settings призначений для загальних налаштувань бізнес-простору. У ньому можуть розміщуватися параметри профілю бізнесу, налаштування доступу, службова інформація та інші конфігураційні можливості. Доступ до цього розділу повинен залежати від ролі користувача, оскільки оператору не потрібно змінювати критичні параметри бізнесу.

Для роботи з даними CRM-панель використовує API Gateway. Усі запити від frontend надсилаються до gateway, а він передає їх до відповідних внутрішніх

сервісів. Це дозволяє клієнтській частині не знати про внутрішню структуру backend і працювати з єдиною точкою входу.

Для завантаження та оновлення серверних даних використовується React Query. Наприклад, список чатів, список лідів, дані асистентів, інформація про підписку та інтеграції можуть кешуватися на клієнті. Після створення або оновлення даних відповідні запити інвалідовуються, і CRM-панель отримує актуальну інформацію з backend.

Форми в CRM-панелі реалізовані з використанням React Hook Form. Це стосується форм створення бізнесу, налаштування AI-асистента, підключення інтеграції, створення ліда або зміни окремих параметрів. Форми повинні перевіряти коректність введених даних і показувати користувачу зрозумілі повідомлення про помилки.

Для стилізації використовується Tailwind CSS. Він дозволяє швидко створювати адаптивний інтерфейс, підтримувати єдиний стиль компонентів і зручно працювати з відступами, кольорами, сітками та станами елементів. У CRM-панелі це важливо, оскільки інтерфейс містить багато таблиць, карток, форм, статусів і списків.

Особливу увагу під час реалізації клієнтської частини було приділено розділенню доступу за ролями. Наприклад, оператор може бачити чати та handoff-звернення, але не повинен мати доступу до керування підпискою або критичних налаштувань бізнесу. Власник і адміністратор мають ширші права. Частина таких обмежень реалізується на backend, але frontend також повинен приховувати недоступні дії, щоб інтерфейс не вводив користувача в оману.

Таким чином, CRM-панель реалізована як основний інтерфейс взаємодії користувача із системою. Вона об'єднує керування бізнесами, AI-асистентами, чатами, лідами, інтеграціями, операторськими зверненнями та підпискою. Завдяки використанню Next.js, React, React Query, React Hook Form і Tailwind CSS клієнтська частина є структурованою, адаптивною та придатною для подальшого розвитку.

#### 4.4 Реалізація AI-асистента, CRM-лідів та передачі діалогу оператору

Однією з основних частин розроблюваного програмного забезпечення є модуль AI-асистента. Його задача полягає в обробці повідомлень клієнтів, формуванні відповідей, використанні контексту діалогу, створенні CRM-лідів та передачі складних звернень оператору підтримки. На відміну від звичайного чат-бота, AI-асистент не працює лише за наперед визначеними сценаріями, а може аналізувати повідомлення користувача природною мовою.

У системі AI-асистент пов'язаний з конкретним бізнес-простором. Це означає, що кожен бізнес може мати власного асистента з окремими налаштуваннями, інструкціями, описом діяльності та доступними інструментами. Такий підхід дозволяє адаптувати поведінку асистента під конкретну компанію, її сферу роботи та стиль спілкування з клієнтами.

Під час створення AI-асистента користувач вказує основні параметри: назву, опис, інструкції, модель поведінки та додаткові налаштування. Ці дані зберігаються в базі даних і використовуються під час обробки кожного нового повідомлення. Коли клієнт надсилає повідомлення через Telegram, web-widget або інший канал, система визначає бізнес, знаходить відповідного асистента та формує контекст для AI-обробки.

Обробка повідомлення починається з того, що Chat Service отримує внутрішній payload від інтеграційного сервісу. У цьому payload міститься інформація про платформу, канал, зовнішнього користувача, thread, бізнес і саме повідомлення. Далі система зберігає повідомлення клієнта в базі даних, перевіряє стан діалогу та визначає, чи можна запускати AI-асистента.

Якщо діалог уже переданий оператору, AI-асистент не повинен автоматично відповідати клієнту. Це важливо, щоб не створювати ситуацію, коли клієнту одночасно відповідає і людина, і автоматизована система. У такому випадку нове повідомлення клієнта зберігається в історії, а оператор бачить його в CRM-панелі та може відповісти вручну.

Якщо передача оператору не активна, Chat Service завантажує налаштування AI-асистента, отримує історію діалогу та формує запит до AI-сервісу. До запиту додаються інструкції асистента, попередні повідомлення, контекст бізнесу та доступні інструменти. Після цього AI-асистент формує відповідь або викликає один із внутрішніх інструментів системи.

У межах реалізації передбачено, що AI-асистент може виконувати не лише текстову відповідь. Він також може ініціювати створення ліда або передачу діалогу оператору. Для цього використовуються внутрішні інструменти, які викликаються через backend-логіку. Такий підхід дозволяє поєднати можливості AI з конкретними бізнес-діями в системі.

CRM-лід є сутністю, яка зберігає інформацію про потенційного клієнта. Лід може містити ім'я, телефон, електронну пошту, короткий опис потреби, статус, джерело звернення, платформу, канал та thread. Якщо клієнт у процесі спілкування залишає контактні дані або повідомляє про зацікавленість у товарі чи послугі, AI-асистент може передати ці дані до інструмента створення ліда.

Під час створення ліда система не повинна довіряти даним без перевірки. Навіть якщо дані були отримані через AI-асистента, backend виконує валідацію полів, перевіряє бізнес-контекст і прив'язує лід до правильного `business_id`. Це потрібно для того, щоб уникнути помилок у структурі даних і не допустити створення ліда в чужому бізнес-просторі.

Створення ліда дозволяє бізнесу не втрачати потенційних клієнтів. Якщо під час діалогу користувач залишив номер телефону або описав свою потребу, ця інформація зберігається у CRM-панелі. Менеджер може відкрити розділ Leads, переглянути нові звернення, змінити статус ліда та продовжити роботу з клієнтом.

Окрім створення лідів, важливою функцією AI-асистента є передача діалогу оператору. Така передача може відбутися в кількох випадках. Наприклад, клієнт прямо просить зв'язатися з менеджером, питання є нестандартним, AI-асистент не може надати достатньо точну відповідь або звернення потребує участі людини. У такому випадку AI-асистент викликає інструмент `handoff`.

Після виклику handoff-інструмента система створює запис про передачу діалогу оператору. У цьому записі зберігається бізнес, thread, канал, причина передачі, коротке резюме діалогу, статус і час створення. Діалог переходить у стан, за якого автоматичні відповіді AI-асистента призупиняються, а оператор може прийняти звернення в CRM-панелі.

Оператор у CRM-панелі бачить список звернень, які очікують обробки. Він може відкрити діалог, переглянути історію повідомлень, побачити причину передачі та відповісти клієнту. Після завершення обробки оператор може закрити звернення. Такий механізм дозволяє поєднати автоматизацію з ручною підтримкою і не залишати клієнта без відповіді в складних ситуаціях.

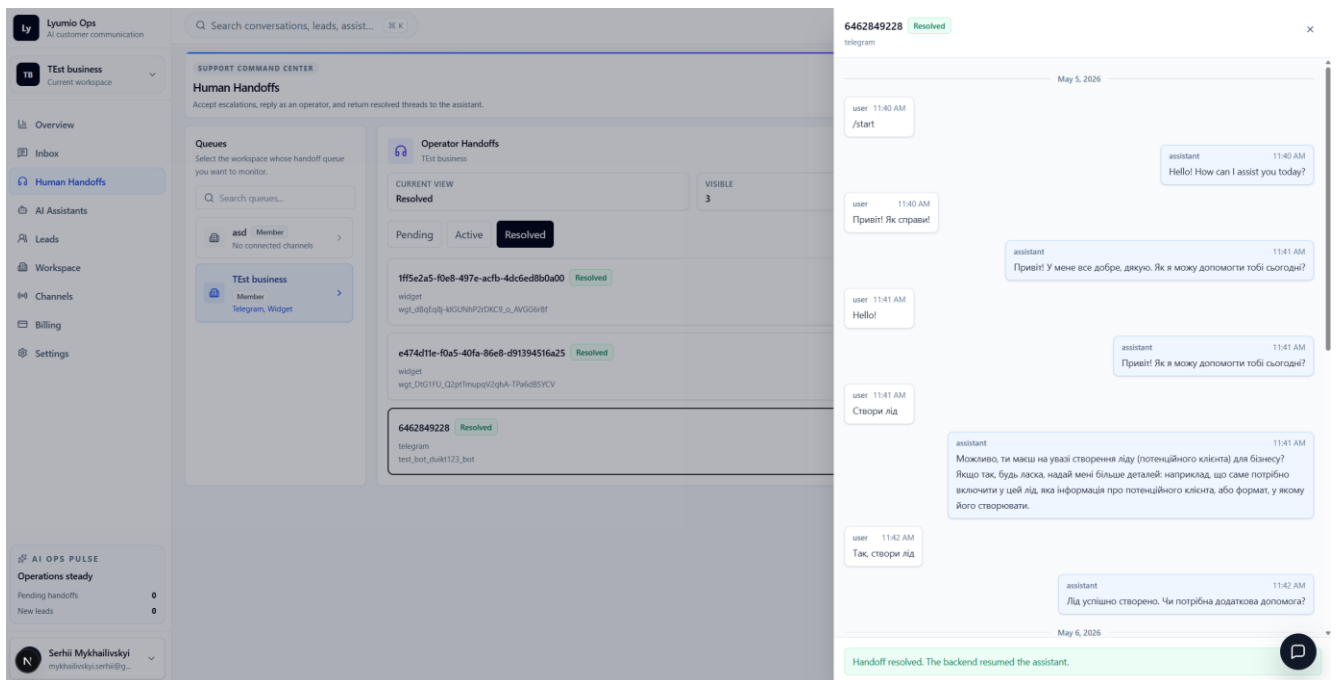


Рис. 4.7. Сторінка звернень, переданих оператору

У реалізації AI-асистента також важливо врахувати збереження історії діалогу. Кожне повідомлення клієнта, відповідь AI-асистента, повідомлення оператора та службова подія зберігаються в базі даних. Це дозволяє відновлювати контекст під час наступних звернень і показувати повну історію спілкування в CRM-панелі.

Для уникнення дублювання дій у системі передбачено контроль обробки діалогу. Якщо клієнт надсилає кілька повідомлень поспіль, система повинна уникати хаотичної паралельної обробки одного thread. Для цього може використовуватися тимчасове блокування обробки діалогу. Такий підхід дозволяє зменшити ризик повторного створення ліда, дублювання відповіді або некоректної зміни handoff-статусу.

AI-асистент працює в межах backend-контексту. Це означає, що критичні дані, такі як `business_id`, права доступу, тарифні обмеження або `thread identity`, не повинні визначатися самим AI. Вони формуються серверною частиною на основі інтеграції, бізнес-простору, сесії користувача або службового контексту. Це підвищує безпеку та передбачуваність роботи системи.

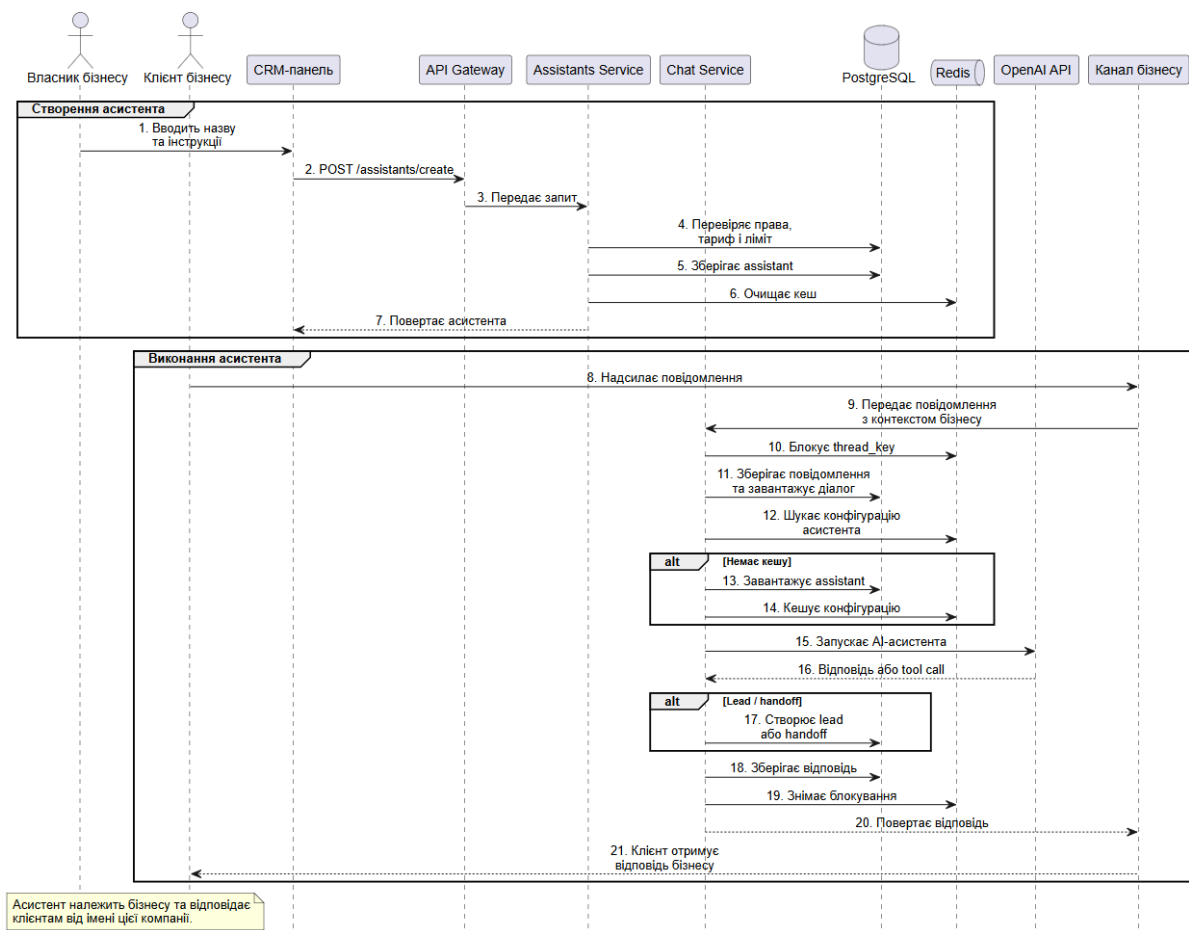


Рис. 4.8. Діаграма послідовності створення та виконання AI-асистента

Загальний процес роботи AI-асистента можна описати так:

1. Клієнт надсилає повідомлення через підключений канал.
2. Інтеграційний сервіс передає повідомлення до Chat Service.
3. Chat Service зберігає повідомлення клієнта.
4. Система перевіряє, чи активна передача оператору.
5. Якщо handoff не активний, завантажуються налаштування AI-асистента.
6. AI-асистент формує відповідь або викликає внутрішній інструмент.
7. Якщо потрібно, створюється CRM-лід.
8. Якщо потрібно, діалог передається оператору.
9. Результат зберігається в базі даних.
10. Відповідь повертається клієнту через відповідний канал.

Таким чином, реалізація AI-асистента в системі охоплює не лише генерацію текстових відповідей, а й виконання прикладних бізнес-дій. AI-асистент може працювати з контекстом діалогу, створювати ліди, передавати звернення оператору та зберігати історію взаємодії. Це дозволяє використовувати його як частину повноцінної CRM-системи, а не лише як окремий чат-бот.

#### **4.5 Реалізація інтеграцій з каналами комунікації та платіжною системою**

Для роботи вебплатформи важливо було реалізувати не лише CRM-панель та AI-асистента, а й інтеграції із зовнішніми сервісами. Саме через інтеграції система отримує повідомлення клієнтів, передає їх у загальний механізм обробки чатів, повертає відповіді користувачам і підтримує підписочну модель доступу для бізнесів.

У межах реалізації було передбачено кілька основних напрямів інтеграцій: Telegram, web-widget, Gmail-сценарій та Stripe Billing. Telegram і web-widget використовуються як канали комунікації з клієнтами. Gmail-сценарій використовується для перевірки email-like обробки повідомлень. Stripe Billing відповідає за платіжну модель, тарифні плани, підписки, зміну плану та синхронізацію платіжних подій.

Основним принципом реалізації інтеграцій є адаптерний підхід. Кожен канал комунікації має власний сервіс, який відповідає за особливості конкретної платформи. При цьому основна логіка обробки повідомлень не дублюється в кожному каналі. Після первинної обробки повідомлення передається до Chat Service, де вже виконується збереження історії, запуск AI-асистента, створення лідів або передача діалогу оператору.

Telegram Service реалізує інтеграцію з Telegram Bot API. Для підключення Telegram-бота користувач у CRM-панелі вводить bot token. Після цього backend перевіряє токен через Telegram API, отримує інформацію про бота, зберігає інтеграцію в базі даних і налаштовує webhook. Завдяки цьому Telegram починає надсилати повідомлення користувачів на backend endpoint системи.

Після отримання webhook-події Telegram Service перевіряє її коректність. Для цього використовується секретний токен webhook, який дозволяє переконатися, що запит надійшов від очікуваного джерела. Також у сервісі передбачена перевірка повторної обробки подій, оскільки Telegram може повторно надіслати update у разі помилки або затримки відповіді від backend.

Отримане повідомлення Telegram Service приводить до внутрішнього формату системи. У цьому форматі міститься інформація про платформу, канал, зовнішнього користувача, thread, бізнес і текст повідомлення. Після цього повідомлення передається до Chat Service. Такий підхід дозволяє Chat Service однаково обробляти повідомлення незалежно від того, з якого каналу воно надійшло.

Якщо AI-асистент сформував відповідь, Telegram Service надсилає її користувачу через Telegram Bot API. Якщо діалог був переданий оператору, автоматична відповідь AI-асистента не генерується, а оператор може відповісти клієнту через CRM-панель. У такому випадку відповідь оператора також повертається клієнту через Telegram.

Web-widget реалізовано як канал комунікації, який може бути вбудований на сайт бізнесу. Його задача полягає в тому, щоб відвідувач сайту міг написати повідомлення в чат, а система передала це повідомлення AI-асистенту або

оператору. Для бізнесу такий канал є зручним, оскільки не потребує переходу клієнта в окремий месенджер.

Для підключення widget бізнес отримує спеціальний widget key. Цей ключ використовується у фрагменті коду, який вставляється на сайт. Після завантаження сторінки браузер звертається до backend, отримує конфігурацію widget і створює сесію відвідувача. При цьому backend самостійно визначає бізнес за widget key, а не довіряє business\_id, переданому з браузера.

Під час реалізації widget-каналу важливо було врахувати безпеку. Оскільки widget key є публічним, його не можна вважати повністю секретним. Тому система додатково перевіряє allowed origins. Це означає, що widget може працювати лише на доменах, які дозволені в налаштуваннях інтеграції. Якщо запит надходить з іншого origin, backend повинен його відхилити.

Widget Service реалізує endpoints для отримання конфігурації, створення сесії, надсилання повідомлення та отримання історії. Коли користувач надсилає повідомлення через widget, воно передається до Chat Service. Після обробки відповідь повертається у widget і відображається в інтерфейсі чату.

Окремо було реалізовано можливість роботи з handoff у widget-каналі. Якщо діалог передано оператору, повідомлення клієнта зберігається в історії, а оператор бачить звернення в CRM-панелі. Оскільки web-widget не має зовнішнього push-механізму, як Telegram, отримання нових повідомлень у браузері може виконуватися через періодичне оновлення або інший механізм синхронізації.

Gmail-сценарій у системі використовується для перевірки email-like обробки повідомлень. Його реалізація дозволяє приймати повідомлення у форматі, схожому на поштове звернення, та передавати його до загального Chat Service. Це потрібно для того, щоб перевірити роботу AI-асистента не лише з месенджерами або widget, а й з email-каналом.

У поточній реалізації Gmail Service може працювати через mock-forwarding endpoint. Такий endpoint імітує отримання email-повідомлення та передає його в систему як звичайне клієнтське звернення. Повідомлення прив'язується до бізнесу,

зовнішнього користувача та thread, після чого обробляється через той самий механізм, що й Telegram або widget.

Перевагою такого підходу є можливість перевірити загальну архітектуру email-інтеграції ще до повного production-підключення Gmail OAuth, Google Pub/Sub або Gmail watch. Надалі цей модуль може бути розширений до повноцінної інтеграції з поштовою скринькою бізнесу, збереженням refresh token, отриманням подій через Gmail API і відправленням відповідей через поштовий сервіс.

Окремою важливою частиною реалізації є Stripe Billing. Оскільки вебплатформа працює за SaaS-моделлю, система повинна підтримувати тарифні плани, підписки, оплату, зміну тарифу та перевірку доступу до платних функцій. Для цього було використано Stripe SDK.

У системі реалізовано тарифні плани, які зберігаються в базі даних. Кожен план має код, назву, опис, вартість, ліміти та перелік доступних функцій. Бізнес може мати активну підписку, яка визначає, які можливості йому доступні. Наприклад, тариф може впливати на кількість асистентів, доступність handoff, кількість лідів або підключення каналів комунікації.

Процес оформлення підписки починається з CRM-панелі. Користувач обирає тариф, після чого frontend надсилає запит на backend. Backend перевіряє права користувача, визначає бізнес, знаходить відповідний план і створює Stripe Checkout Session. Після цього користувач перенаправляється на сторінку оплати Stripe.

Важливою частиною платіжної логіки є те, що доступ до платних функцій не відкривається лише після переходу на success URL. Success URL означає тільки те, що користувач повернувся зі Stripe. Реальний стан підписки оновлюється після обробки Stripe webhook. Це дозволяє уникнути ситуації, коли доступ відкривається без підтвердження платежу.

Stripe webhook endpoint приймає події від Stripe, перевіряє підпис події, дедуплікує її за event id та оновлює локальний стан підписки. Для цього в базі даних зберігаються Stripe events. Якщо подія вже була оброблена раніше, система не повинна повторно виконувати ті самі зміни. Це захищає від дублювання платіжних операцій.

У межах реалізації також передбачено зміну тарифного плану. Якщо бізнес переходить на дорожчий план, система може ініціювати upgrade. Якщо бізнес переходить на дешевший план, така зміна може бути запланована на кінець поточного періоду. Для цього в підписці зберігається інформація про pending plan. Так користувач може бачити, що зміна тарифу вже запланована, але ще не застосована.

Також реалізовується можливість скасування запланованої зміни тарифу. Якщо користувач передумав до завершення поточного періоду, система може скасувати pending downgrade, оновити дані в Stripe і очистити відповідні поля в локальній підписці. Це робить платіжну модель більш гнучкою для бізнесу.

Для контролю платіжних дій використовується billing audit log. У ньому зберігаються службові записи про створення checkout session, portal session, зміну тарифу, синхронізацію підписки, обробку webhook-подій або виникнення помилок. Такий журнал допомагає аналізувати стан платіжної системи та швидше знаходити причини некоректної поведінки.

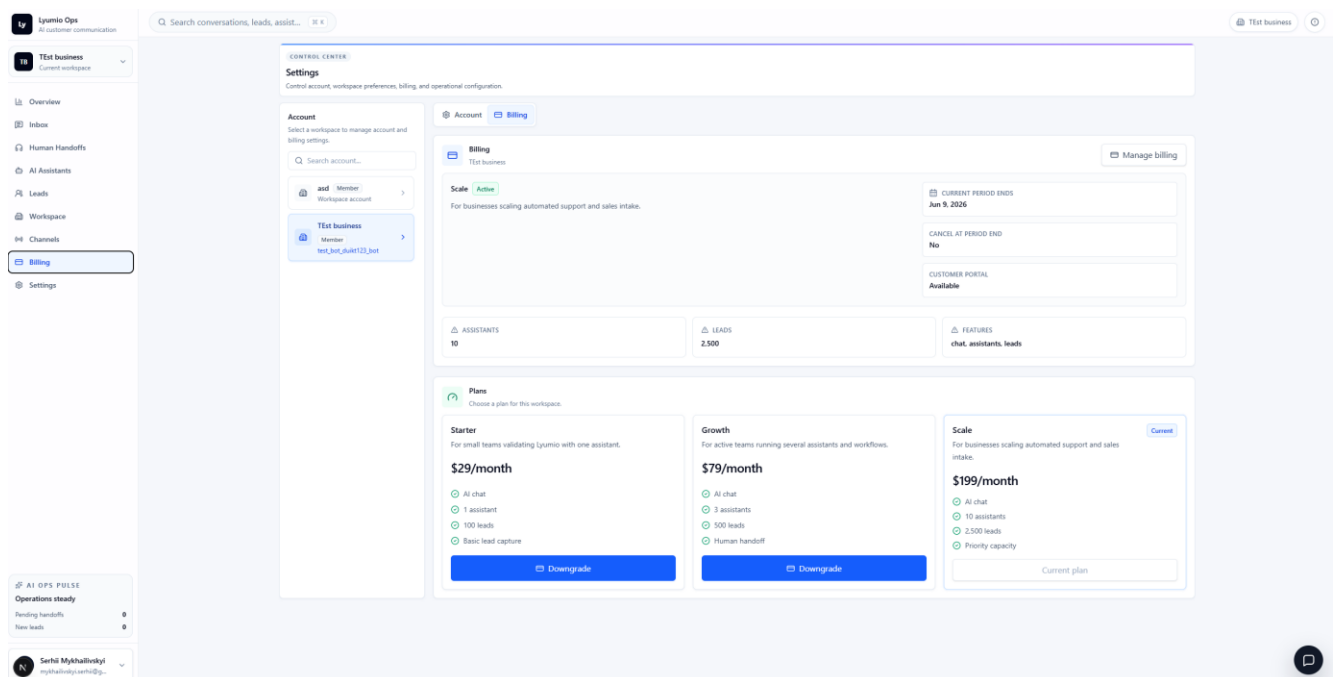


Рис. 4.9. Сторінка керування підпискою

Усі інтеграції реалізовані так, щоб не порушувати загальну архітектуру системи. Telegram, Gmail і widget відповідають за доставку повідомлень, Chat Service за AI-обробку, Businesses Service за бізнес-дані, ліди, handoff і підписки, а Stripe за платіжну частину. Такий поділ дозволяє підтримувати систему простішою для розвитку та тестування.

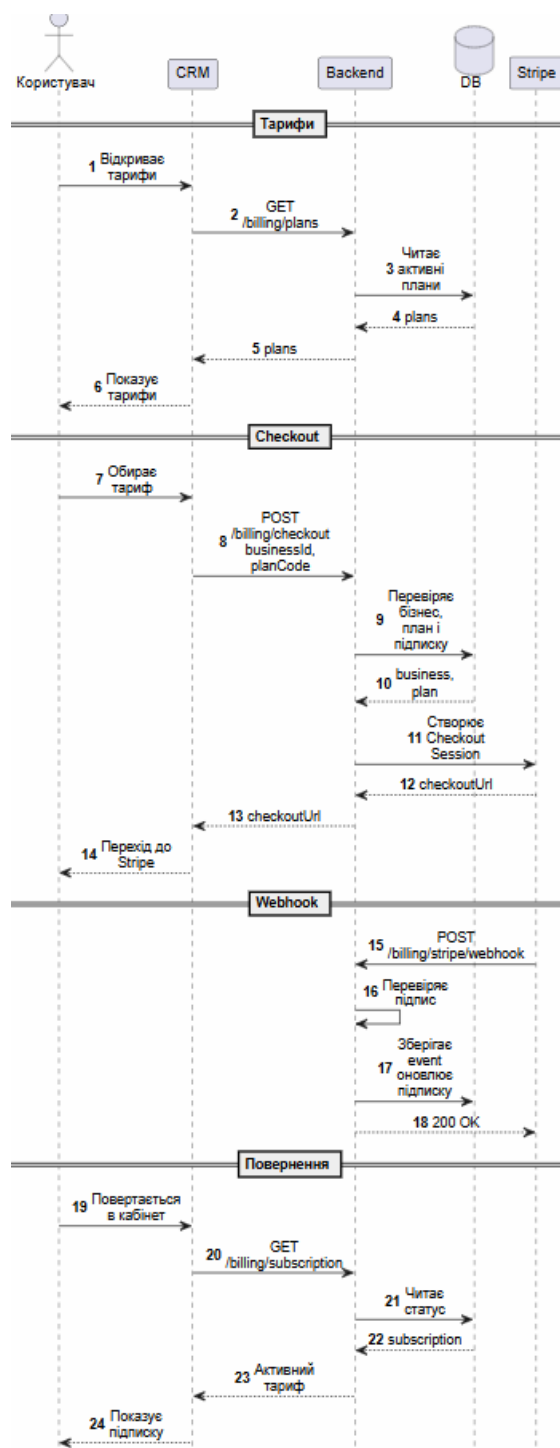


Рис. 4.10. Схема реалізації інтеграції платіжної системи

Таким чином, у межах реалізації було створено інтеграції з каналами комунікації та платіжною системою. Telegram і web-widget дозволяють приймати повідомлення клієнтів, Gmail-сценарій демонструє можливість email-обробки, а Stripe Billing забезпечує роботу тарифів і підписок. Завдяки цьому система може не лише відповідати на повідомлення клієнтів, а й працювати як повноцінна SaaS-платформа для бізнесу.

#### **4.6 Тестування основних сценаріїв роботи системи**

Тестування програмного забезпечення виконувалося для перевірки коректності основних сценаріїв роботи системи. Основну увагу було приділено тим функціям, які безпосередньо впливають на роботу вебплатформи: авторизації користувача, керуванню бізнес-просторами, створенню AI-асистента, обробці повідомлень клієнтів, створенню CRM-лідів, передачі діалогу оператору, роботі інтеграцій та перевірці підписки.

Під час тестування перевірялися не лише окремі сторінки інтерфейсу, а й повні сценарії проходження даних через систему. Наприклад, для сценарію обробки повідомлення перевірялося отримання повідомлення з каналу комунікації, передача його до backend-частини, збереження в історії діалогу, обробка AI-асистентом і повернення відповіді клієнту.

Для перевірки серверної частини використовувалися API-запити та автоматизовані тести окремих endpoint-ів. Для клієнтської частини перевірялася коректність відображення сторінок CRM-панелі, робота форм, списків, таблиць і переходів між розділами. Також було перевірено, що користувач отримує доступ лише до тих бізнес-просторів, у яких він має відповідну роль.

Основні результати тестування наведено в таблиці 4.1.

Таблиця 4.1

## Результати тестування

| Група тестування      | Що перевірялося                                                                                              | Результат |
|-----------------------|--------------------------------------------------------------------------------------------------------------|-----------|
| Авторизація та доступ | Вхід через Auth0,<br>перевірка /auth/me,<br>доступ лише до власних<br>бізнес-просторів                       | Успішно   |
| Бізнеси та ролі       | Створення бізнес-<br>простору, роль OWNER,<br>перевірка доступу за<br>ролями                                 | Успішно   |
| AI-асистенти          | Створення, редагування<br>та використання<br>налаштувань AI-<br>асистента                                    | Успішно   |
| Обробка чатів         | Надсилення<br>повідомлень через<br>Telegram і web-widget,<br>збереження історії та<br>AI-відповідь           | Успішно   |
| CRM-ліди та handoff   | Створення ліда,<br>передача діалогу<br>оператору, відповідь<br>оператора клієнту                             | Успішно   |
| Інтеграції та оплата  | Перевірка widget origins,<br>Telegram webhook, Stripe<br>Checkout Session,<br>webhook і тарифних<br>обмежень | Успішно   |

За результатами тестування було підтверджено, що основні сценарії роботи системи виконуються коректно. Користувач може авторизуватися, працювати з бізнес-просторами, створювати AI-асистентів, переглядати чати, обробляти ліди та керувати інтеграціями. Backend-частина коректно обробляє запити, перевіряє доступ користувача до бізнесу, зберігає дані в базі та взаємодіє із зовнішніми сервісами.

Окремо було перевірено сценарії, пов'язані з AI-асистентом. Система приймає повідомлення клієнта, передає його до механізму обробки чатів, зберігає історію діалогу та формує відповідь. У випадках, коли клієнт залишає контактні дані або просить зв'язатися з менеджером, система може створити CRM-лід або передати діалог оператору підтримки.

Таким чином, тестування підтвердило працездатність основних модулів вебплатформи та коректність взаємодії між CRM-панеллю, API Gateway, внутрішніми сервісами, базою даних, AI-модулем і каналами комунікації.

#### **4.7 Демонстрація результатів роботи програмного забезпечення**

У результаті реалізації було створено вебплатформу для керування AI-асистентами, обробки чатів, роботи з CRM-лідами, інтеграціями та підписками. Демонстрація роботи програмного забезпечення підтверджує, що користувач може виконувати основні дії через CRM-панель: створювати бізнес-простір, налаштовувати AI-асистента, переглядати діалоги з клієнтами, працювати з лідами, керувати каналами комунікації та перевіряти стан підписки.

Основні результати роботи системи відображено на екранних формах CRM-панелі. До них належать сторінки авторизації, вибору або створення бізнесу, налаштування AI-асистентів, перегляду чатів, обробки лідів, керування інтеграціями та billing-сценаріями. Наявність цих сторінок показує, що реалізована система охоплює ключові сценарії використання, визначені у вимогах до програмного забезпечення.

Окремо було продемонстровано сценарій обробки повідомлення клієнта. Повідомлення надходить з каналу комунікації, передається до backend-частини, зберігається в історії діалогу та обробляється AI-асистентом. У разі потреби система може створити CRM-лід або передати діалог оператору підтримки. Такий сценарій підтверджує працездатність основної логіки платформи: взаємодії між каналом комунікації, сервісом чатів, AI-модулем, базою даних і CRM-панеллю.

Також було перевірено відображення даних у CRM-панелі. Користувач має змогу переглядати список чатів, відкривати історію повідомлень, бачити створені ліди та працювати з інтеграціями. Це дозволяє використовувати платформу не лише як AI-чат, а як інструмент централізованого керування клієнтською комунікацією.

Отже, демонстрація результатів підтвердила, що розроблене програмне забезпечення реалізує основні функції, необхідні для автоматизації обробки типових звернень клієнтів. Система поєднує AI-асистентів, CRM-панель, історію чатів, створення лідів, передачу діалогу оператору, інтеграції з каналами комунікації та підписочну модель доступу.

## ВИСНОВКИ

У бакалаврській роботі було розроблено програмне забезпечення для створення AI-асистентів, обробки чатів і централізованого керування бізнесами. Розроблена вебплатформа спрямована на спрощення обслуговування клієнтів бізнесу шляхом автоматизації обробки типових звернень за допомогою AI-асистентів.

У процесі виконання роботи було проведено аналіз предметної області автоматизації взаємодії бізнесу з клієнтами. Було визначено, що для малого та середнього бізнесу актуальними є проблеми швидкої обробки типових звернень, розпорошеності каналів комунікації, відсутності централізованої історії діалогів, ручного створення лідів і складності передачі звернення оператору. Використання AI-асистентів дозволяє частково вирішити ці проблеми за рахунок автоматизованої первинної відповіді, збереження контексту діалогу та виконання окремих бізнес-дій.

Було досліджено існуючі програмні рішення для автоматизації клієнтської підтримки та управління комунікацією, зокрема Intercom, Zendesk, HubSpot і Freshdesk. У результаті порівняння встановлено, що такі платформи мають розвинені можливості для роботи з клієнтами, однак не завжди забезпечують достатню гнучкість backend-логіки, адаптацію під власну архітектуру та просте самостійне створення AI-асистентів у межах окремих бізнес-просторів. Це підтвердило доцільність розробки власного програмного забезпечення.

У роботі було визначено функціональні та нефункціональні вимоги до вебплатформи. До основних функціональних вимог було віднесено реєстрацію та автентифікацію користувачів, створення бізнес-профілів, налаштування AI-асистентів, обробку повідомлень клієнтів, збереження історії діалогів, створення CRM-лідів і передачу діалогу оператору. Серед нефункціональних вимог було визначено продуктивність, стабільність, безпеку, зручність використання, модульність і надійність інтеграцій.

Було спроектовано архітектуру системи, модель доступу користувачів, механізм обробки чатів, структуру бази даних та інтеграції з каналами комунікації. Архітектура системи побудована з використанням API Gateway, внутрішніх сервісів, CRM-панелі, PostgreSQL, Redis та зовнішніх сервісів. Для розмежування доступу було використано модель бізнес-просторів і ролей OWNER, ADMIN та OPERATOR. Такий підхід дозволяє логічно ізолювати дані різних бізнесів і контролювати доступ користувачів до функцій системи.

У межах практичної реалізації було створено основні модулі програмного забезпечення: API Gateway, внутрішні backend-сервіси, CRM-панель, модуль AI-асистента, обробку чатів, створення лідів, передачу діалогу оператору, інтеграції з Telegram, web-widget, email-сценаріями та платіжну модель на основі Stripe. Для реалізації було використано TypeScript, Node.js, Express, React, Next.js, PostgreSQL, Redis, Docker, Auth0/OIDC, OpenAI SDK та Stripe SDK.

Було реалізовано механізм обробки повідомлень клієнтів, за якого повідомлення з різних каналів комунікації приводяться до спільного внутрішнього формату та передаються до Chat Service. Це дозволяє використовувати єдину AI-логіку незалежно від джерела повідомлення. AI-асистент може формувати відповіді, створювати CRM-ліди та ініціювати передачу діалогу оператору підтримки.

Також було реалізовано CRM-панель, яка надає користувачу доступ до основних функцій системи. Через неї можна працювати з бізнесами, AI-асистентами, чатами, лідами, handoff-зверненнями, інтеграціями та підпискою. Такий інтерфейс дозволяє бізнес-користувачу керувати клієнтською комунікацією в одному середовищі без необхідності використовувати кілька окремих інструментів.

Після реалізації було проведено тестування основних сценаріїв роботи системи. Було перевірено авторизацію через Auth0, створення бізнес-просторів, роботу ролей доступу, створення та редагування AI-асистентів, обробку повідомлень через Telegram і web-widget, створення CRM-лідів, передачу діалогу оператору, перевірку widget origins, роботу Stripe Checkout Session та обробку Stripe

webhook. Результати тестування показали, що основні сценарії виконуються коректно.

Отже, мету бакалаврської роботи було досягнуто. Розроблене програмне забезпечення дозволяє спростити обслуговування клієнтів бізнесу шляхом автоматизації обробки типових звернень за допомогою AI-асистентів. Система може бути використана малим і середнім бізнесом для первинної консультації клієнтів, збереження історії діалогів, збору лідів, передачі складних звернень оператору та централізованого керування комунікацією.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Михайлівський С.О., Худік Б.О. Розробка SaaS-Платформи для управління AI асистентами у бізнесі. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, 536 с.
2. Михайлівський С.О., Задонцев Ю.В. Використання AI-Асистентів для автоматизації бізнес-процесів. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. 314-315 с.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Hoboken: Pearson, 2021. 1168 p.
2. Davenport T. H., Ronanki R. Artificial Intelligence for the Real World. Harvard Business Review. 2018. p. 108–116.
3. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 616 p.
4. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 612 p.
5. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley Professional, 2021. 464 p.
6. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
7. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley Professional, 2002. 560 p.
8. McKinsey & Company. The state of AI in 2023: Generative AI's breakout year. 2023. URL: <https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year> (дата звернення: 09.05.2026).
9. Intercom. The only helpdesk designed for the AI Agent era. URL: <https://www.intercom.com/> (дата звернення: 09.05.2026).
10. Zendesk. AI for customer service. URL: <https://www.zendesk.com/service/ai/> (дата звернення: 09.05.2026).
11. HubSpot. AI-powered customer service platform for growing teams. URL: <https://www.hubspot.com/products/service> (дата звернення: 09.05.2026).
12. Freshworks. Freshdesk: AI-powered platform for modern customer service. URL: <https://www.freshworks.com/freshdesk/> (дата звернення: 09.05.2026).
13. TypeScript. The TypeScript Handbook. URL: <https://www.typescriptlang.org/docs/handbook/intro.html> (дата звернення: 09.05.2026).

09.05.2026).

- 14.Node.js. Introduction to Node.js. URL: <https://nodejs.org/learn/getting-started/introduction-to-nodejs> (дата звернення: 09.05.2026).
- 15.Express. Express: Node.js web application framework. URL: <https://expressjs.com/> (дата звернення: 09.05.2026).
- 16.React. Documentation. URL: <https://react.dev/> (дата звернення: 09.05.2026).
- 17.Next.js. App Router Documentation. URL: <https://nextjs.org/docs/app> (дата звернення: 09.05.2026).
- 18.PostgreSQL Global Development Group. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (дата звернення: 09.05.2026).
- 19.Redis. Redis Documentation. URL: <https://redis.io/docs/latest/> (дата звернення: 09.05.2026).
- 20.OpenAI. Agents SDK for TypeScript. URL: <https://openai.github.io/openai-agents-js/> (дата звернення: 09.05.2026).
- 21.Telegram. Telegram Bot API. URL: <https://core.telegram.org/bots/api> (дата звернення: 09.05.2026).
- 22.Google Developers. Gmail API overview. URL: <https://developers.google.com/workspace/gmail/api/guides> (дата звернення: 09.05.2026).
- 23.Auth0. OpenID Connect Protocol. URL: <https://auth0.com/docs/authenticate/protocols/openid-connect-protocol> (дата звернення: 09.05.2026).
- 24.Stripe Documentation URL: <https://docs.stripe.com/> (дата звернення: 09.05.2026).
- 25.Docker. Docker Compose. URL: <https://docs.docker.com/compose/> (дата звернення: 09.05.2026).
- 26.GitHub. GitHub Actions documentation. URL: <https://docs.github.com/actions> (дата звернення: 09.05.2026).

## ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ  
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ  
ТЕХНОЛОГІЙ  
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



### Розробка SaaS-платформи для управління AI-асистентами у B2B моделях

Виконав студент 4 курсу  
групи ПД-44  
Михайлівський Сергій Олександрович  
Керівник роботи  
канд. техн. наук, доц. Задонцев Юрій Вікторович

Київ – 2026

### МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета роботи:** спрощення обслуговування клієнтів бізнесу шляхом автоматизації обробки типових звернень за допомогою AI-асистентів.

**Об'єкт дослідження:** процеси автоматизації взаємодії бізнесу з клієнтами у B2B моделі за допомогою інтелектуальних програмних систем.

**Предмет дослідження:** методи, засоби та архітектурні рішення розробки SaaS-платформи для створення AI-асистентів, обробки чатів і централізованого керування бізнесами у B2B.

## ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної області автоматизації взаємодії бізнесу з клієнтами за допомогою AI-асистентів.
2. Дослідити існуючі програмні рішення для обробки клієнтських звернень, управління чатами та використання AI-асистентів.
3. Визначити функціональні та нефункціональні вимоги до SaaS-платформи.
4. Спроекувати архітектуру системи, модель доступу користувачів, механізм обробки чатів та структуру бази даних.
5. Реалізувати основні модулі програмного забезпечення: API Gateway, внутрішні сервіси, CRM-панель, модуль AI-асистента, створення лідів та передачу діалогу оператору.
6. Провести тестування основних сценаріїв роботи системи.

3

## АНАЛІЗ АНАЛОГІВ

| Показник                          | Intercom                                              | Zendesk                                                    | HubSpot                                                  | Freshdesk                                              | Розроблена платформа                                       |
|-----------------------------------|-------------------------------------------------------|------------------------------------------------------------|----------------------------------------------------------|--------------------------------------------------------|------------------------------------------------------------|
| Основна задача системи            | Підтримка клієнтів у чатах та inbox                   | Обробка звернень через ticketing/support                   | Управління клієнтами, продажами та сервісом              | Омніканальна підтримка клієнтів                        | Створення та керування AI-асистентами для бізнесу          |
| Що є центром роботи системи       | Клієнтська розмова                                    | Тікет / звернення                                          | CRM-контакт, компанія, угода                             | Тікет / звернення                                      | Бізнес-простір, AI-асистент, чат і лід                     |
| Як використовується AI            | Fin AI Agent автоматизує відповіді у customer service | AI Agents автоматизують запити у support                   | Breeze Agent допомагає у CRM, sales, marketing і service | Freddy Agent автоматизує support-звернення             | AI-асистент створюється як окрема сутність бізнесу         |
| Запуск для нового бізнесу         | Налаштувати workspace, inbox, knowledge base та Fin   | Налаштувати support-акаунт, канали, базу знань і AI Agents | Налаштувати CRM, об'єкти, канали та Customer Agent       | Налаштувати helpdesk, канали, knowledge base та Freddy | Створити бізнес-простір, AI-асистента і підключити канал   |
| Результат після звернення клієнта | Діалог у inbox та історія підтримки                   | Тікет або звернення в support-системі                      | Запис у CRM та історія взаємодії з клієнтом              | Тікет або діалог у helpdesk                            | Чат, повідомлення, лід або handoff у межах бізнес-простору |

4

## ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### Функціональні вимоги

1. Реєстрація та автентифікація користувачів.
2. Створення та керування бізнес-профілями.
3. Створення, налаштування та редагування AI-асистентів.
4. Обробка повідомлень клієнтів через внутрішній чат.
5. Збереження історії взаємодій та централізоване керування чатами.
6. Передача діалогу оператору за потреби.

### Нефункціональні вимоги

1. Час відповіді на типовий запит до 2 секунд, AI-відповідь до 5 секунд.
2. Забезпечити безпеку застосуванням HTTPS, автентифікації через Auth0, розмежування доступу, захист конфіденційних даних через рівні доступу в базі даних.
3. Забезпечити зручність використання створенням бізнес-профілю або AI-асистента за 3-5 кроків, зрозумілою навігацією.
4. Розробити окремі модулі для бізнесів, AI-асистентів і чатів.
5. Надійність інтеграцій: коректна обробка помилок у разі недоступності зовнішніх сервісів.

5

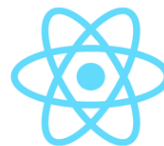
## ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



TypeScript



Node.js



React.js



Next.js

Stripe Node.js  
SDK

PostgreSQL



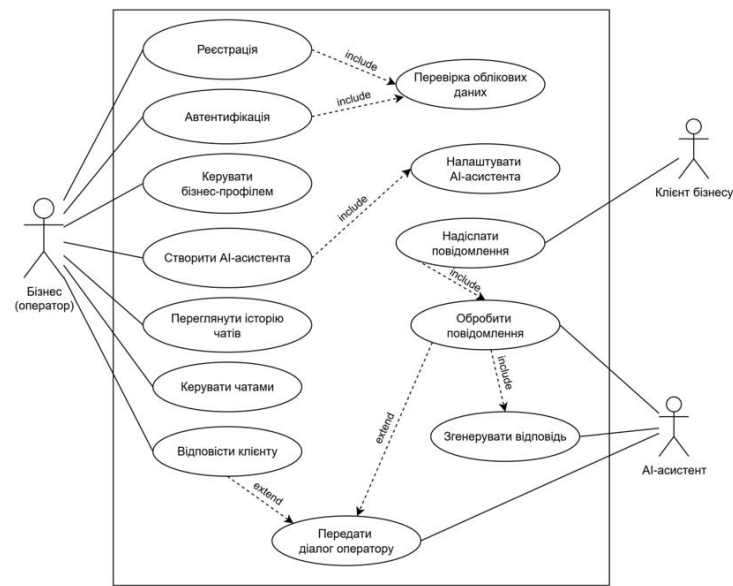
redis



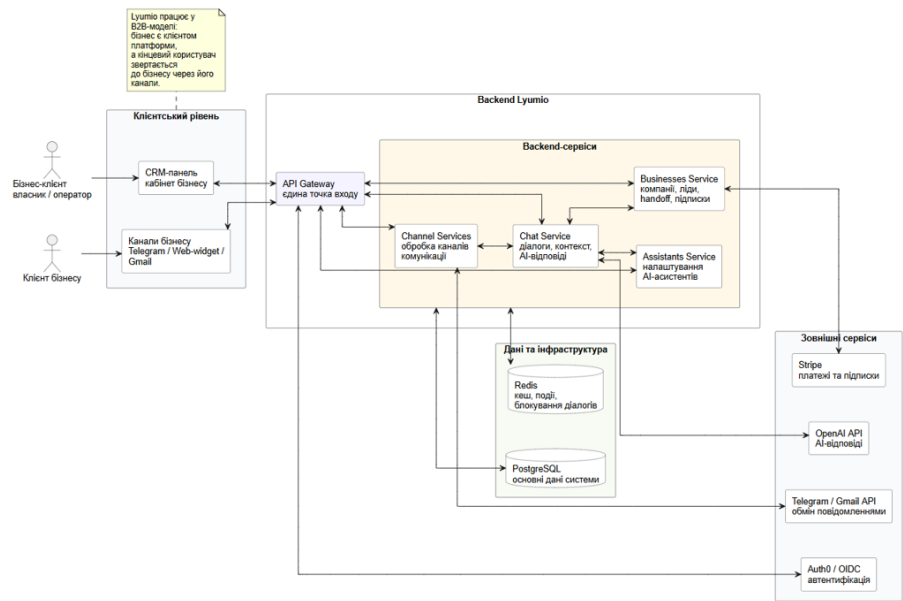
Auth0

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

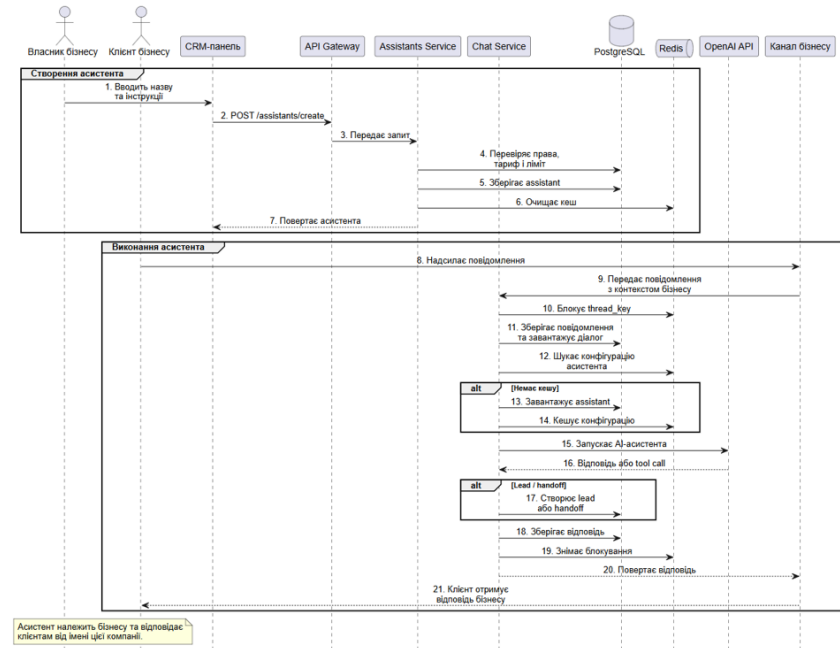


ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ

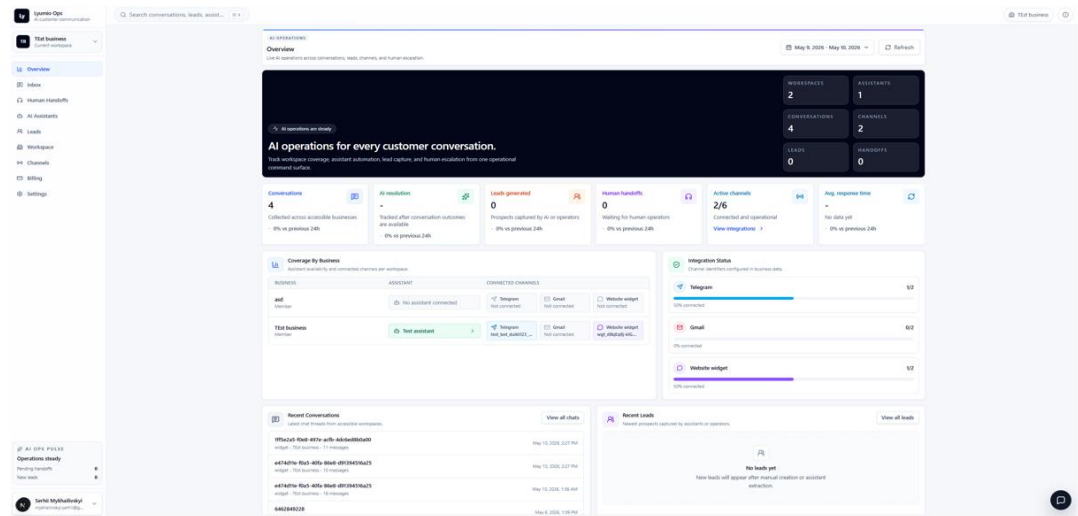




# ДІАГРАМА ПОСЛІДОВНОСТІ СТВОРЕННЯ ТА ВИКОНАННЯ AI-АСИСТЕНТА

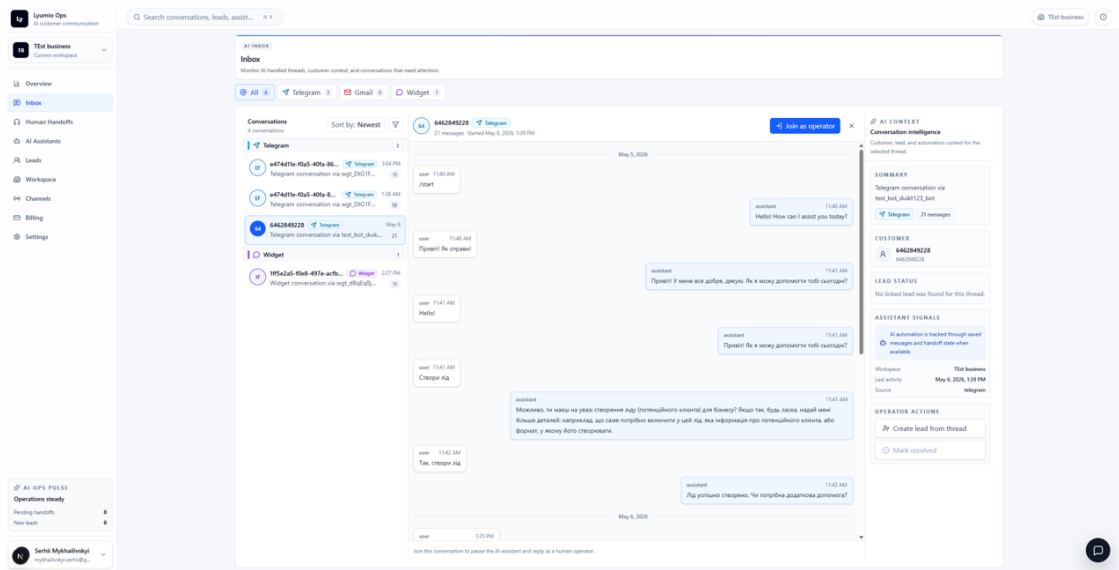


# ЕКРАННІ ФОРМИ



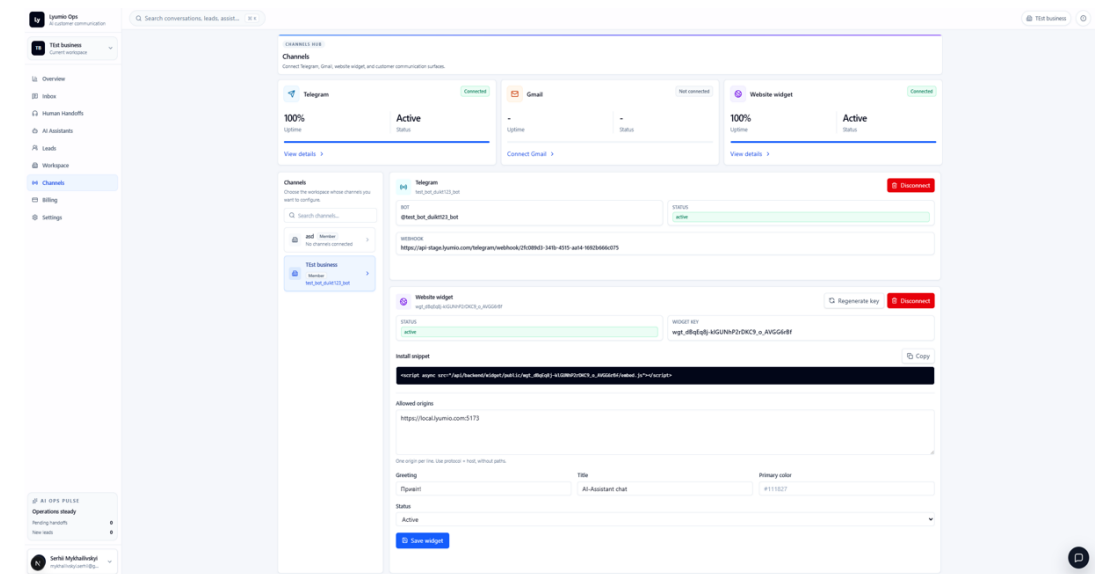
# Головна сторінка CRM-панелі

ЕКРАННІ ФОРМИ



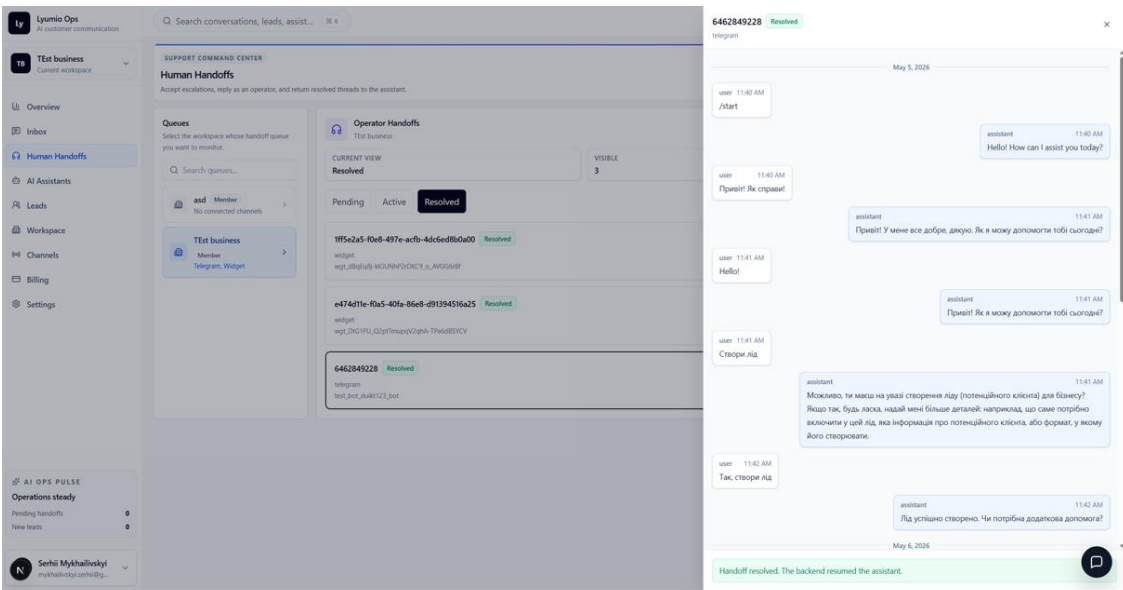
Сторінка перегляду чатів

ЕКРАННІ ФОРМИ



Сторінка інтеграцій

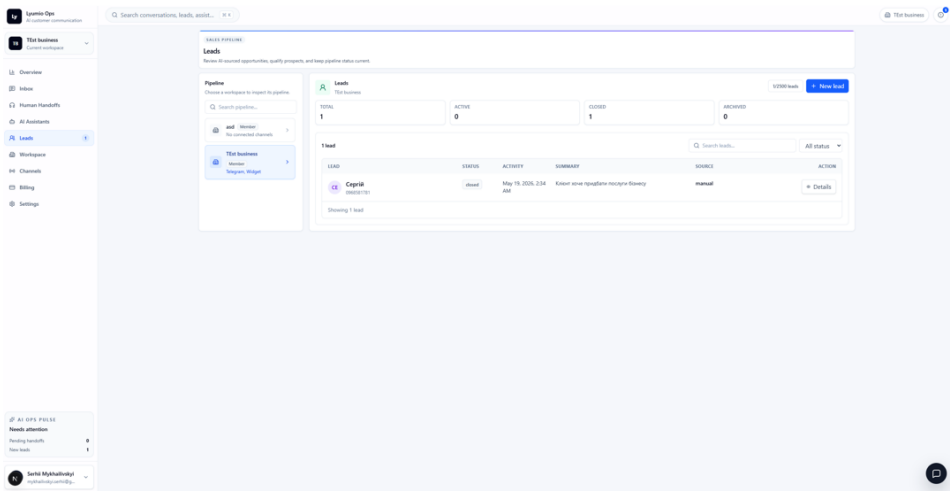
## ЕКРАННІ ФОРМИ



Сторінка звернень, переданих оператору

15

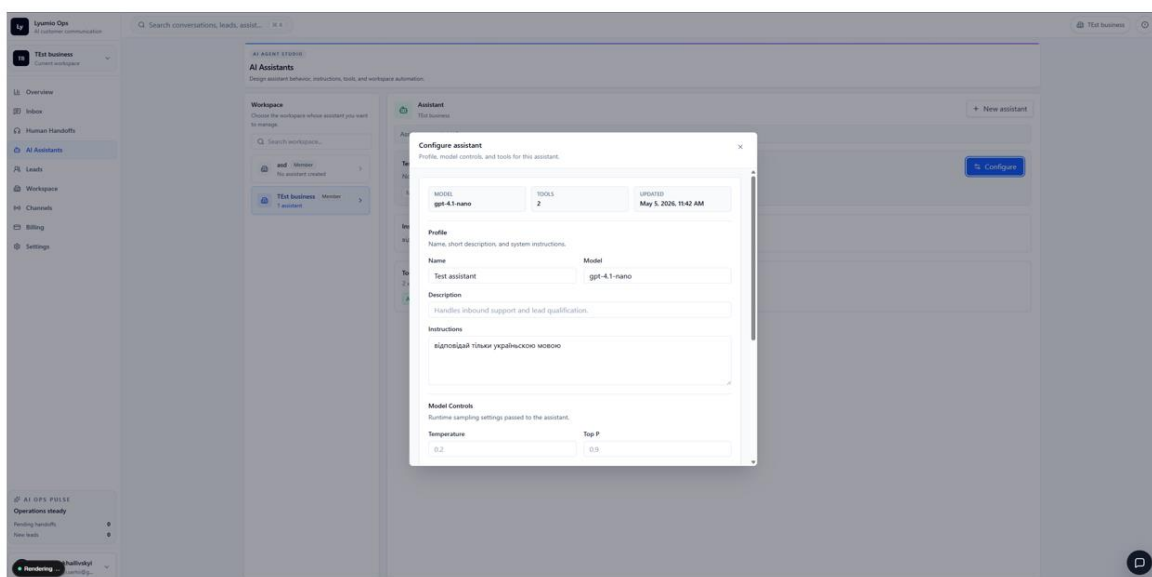
## ЕКРАННІ ФОРМИ



Сторінка перегляду створених лідів

16

## ЕКРАННІ ФОРМИ



Сторінка створення та  
налаштування AI-асистента

17

## АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Михайлівський С.О., Худік Б.О. Розробка SaaS-Платформи для управління AI асистентами у бізнесі. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, 536 с.
2. Михайлівський С.О., Задонцев Ю.В. Використання AI-Асистентів для автоматизації бізнес-процесів. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. 314-315 с.

18

## ВИСНОВКИ

1. Проведено аналіз предметної області автоматизації взаємодії бізнесу з клієнтами за допомогою AI-асистентів у B2B-моделях, у результаті чого визначено основні проблеми клієнтської комунікації.
2. Досліджено аналоги [Intercom](#), [Zendesk](#), [HubSpot](#) та [Freshdesk](#), у результаті чого визначено їхні можливості та обмеження щодо обробки клієнтських звернень, управління чатами, CRM-функціональності, інтеграцій та налаштування AI-асистентів.
3. Визначено функціональні та нефункціональні вимоги до [SaaS-платформи](#).
4. Проведено аналіз та обрані технології, методи та інструменти розробки [SaaS-платформи](#) для створення AI-асистентів зокрема [TypeScript](#), [React.js](#), [Node.js](#) та інші.
5. Спроектовано архітектуру системи, модель доступу користувачів, механізм обробки чатів та структуру бази даних.
6. Реалізовано основні модулі системи, у результаті чого отримано веб-платформу для керування бізнес-просторами, AI-асистентами, чатами, [CRM-лідами](#), інтеграціями, підписками, а також для передачі діалогу оператору.
7. Проведено тестування основних сценаріїв роботи шляхом модульного та інтеграційного тестування з використанням [Vitest](#) і [Supertest](#), у результаті чого підтверджено відповідність реалізованої системи визначеним вимогам.

## ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

### Assistants controller

```

import { Request, Response, Router } from "express";
import {
  getAuthenticatedUserContext,
  getErrorMessage,
  getErrorStatusCode,
  getRequestLogger,
  logControllerError,
  requireInternalAuthContext,
} from "shared-lib";
import { AssistantUseCase } from "../use-cases/Assistants/assistant.use-case";
import { AssistantRequestValidationError } from "../validation/Assistants/assistant.validation";
import {
  parseAssistantCreate,
  parseAssistantId,
  pickAssistantUpdates,
} from "../validation/Assistants/assistant.request";
import { assistantsLogger } from "../logger";

const router: Router = Router();

router.use(requireInternalAuthContext);

function getErrorName(error: unknown): string | undefined {
  if (typeof error !== "object" || error === null) {
    return undefined;
  }

  if (!("name" in error)) {
    return undefined;
  }

  return String((error as { name?: unknown }).name);
}

function getDatabaseErrorCode(error: unknown): string | undefined {
  if (typeof error !== "object" || error === null) {
    return undefined;
  }

  const original = (error as { original?: { code?: unknown } }).original;
  if (typeof original?.code === "string") {
    return String(original.code);
  }

  const parent = (error as { parent?: { code?: unknown } }).parent;
  if (typeof parent?.code === "string") {
    return String(parent.code);
  }

  return undefined;
}

function resolveControllerError(error: unknown): { statusCode: number; message: string } {
  const explicitStatusCode = getErrorStatusCode(error, 0);
  if (explicitStatusCode > 0) {
    return {
      statusCode: explicitStatusCode,
      message: getErrorMessage(error, "Unexpected server error"),
    };
  }

  if (error instanceof AssistantRequestValidationError) {
    return {
      statusCode: error.statusCode,
      message: error.message,
    };
  }

  const errorName = getErrorName(error);
  const errorCode = getDatabaseErrorCode(error);

  if (errorName === "SequelizeUniqueConstraintError" || errorCode === "23505") {

```

```

return {
  statusCode: 409,
  message: "Assistant already exists for this business",
};
}

if (errorName === "SequelizeForeignKeyConstraintError" ||
errorCode === "23503") {
  return {
    statusCode: 404,
    message: "Business not found",
  };
}

return {
  statusCode: 500,
  message: "Unexpected server error",
};
}

async function getAssistants(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, assistantsLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const response = await
AssistantUseCase.getAll(userContext);
    res.json(response);
  } catch (error) {
    const resolvedError = resolveControllerError(error);
    logControllerError({
      logger,
      event: "assistants.list.failed",
      error,
      statusCode: resolvedError.statusCode,
      failureMessage: "Assistants controller request failed",
    });
    res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
  }
}

```

```

async function createAssistant(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, assistantsLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const payload = parseAssistantCreate(req.body);
    const databaseAssistant = await
AssistantUseCase.create(payload, userContext);
    const assistantId = databaseAssistant.get("id");
    const businessId = databaseAssistant.get("business_id");

    logger.info(
      {
        event: "assistant.created",
        assistantId,
        businessId,
      },
      "Assistant created"
    );

    res.status(201).json(databaseAssistant);
    return;
  } catch (error) {
    const resolvedError = resolveControllerError(error);
    logControllerError({
      logger,
      event: "assistant.create.failed",
      error,
      statusCode: resolvedError.statusCode,
      failureMessage: "Assistants controller request failed",
    });
    res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
    return;
  }
}

```

```

async function updateAssistant(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, assistantsLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);

```

```

const assistantId = parseAssistantId(req.params.id);
const updates = pickAssistantUpdates(req.body);

if (Object.keys(updates).length === 0) {
  res.status(400).json({ error: "No valid fields provided for
update" });
  return;
}

const updatedAssistant = await AssistantUseCase.update(
  assistantId,
  updates,
  userContext
);

if (!updatedAssistant) {
  res.status(404).json({ error: "Assistant not found" });
  return;
}

logger.info(
  {
    event: "assistant.updated",
    assistantId,
    updatedFields: Object.keys(updates),
  },
  "Assistant updated"
);

res.json(updatedAssistant);
return;
} catch (error) {
  const resolvedError = resolveControllerError(error);
  logControllerError({
    logger,
    event: "assistant.update.failed",
    error,
    statusCode: resolvedError.statusCode,
    failureMessage: "Assistants controller request failed",
  });
  res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
  return;
}

```

```

}

async function deleteAssistant(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, assistantsLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const assistantId = parseAssistantId(req.params.id);
    const response = await AssistantUseCase.delete(assistantId,
userContext);

    if (!response) {
      res.status(404).json({ error: "Assistant not found" });
      return;
    }

    logger.info(
      {
        event: "assistant.deleted",
        assistantId,
      },
      "Assistant deleted"
    );

    res.sendStatus(200);
    return;
  } catch (error) {
    const resolvedError = resolveControllerError(error);
    logControllerError({
      logger,
      event: "assistant.delete.failed",
      error,
      statusCode: resolvedError.statusCode,
      failureMessage: "Assistants controller request failed",
    });
    res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
    return;
  }
}

async function getAssistantById(req: Request, res: Response):
Promise<void> {

```

```

const logger = getRequestLogger(res, assistantsLogger);

try {
  const userContext = getAuthenticatedUserContext(res);
  const assistantId = parseAssistantId(req.params.id);
  const response = await
AssistantUseCase.getById(assistantId, userContext);

  if (!response) {
    res.status(404).json({ error: "Assistant not found" });
    return;
  }

  res.json(response);
  return;
} catch (error) {
  const resolvedError = resolveControllerError(error);
  logControllerError({
    logger,

```

## Billing controller

```

import { Request, Response, Router } from "express";
import {
  getAuthenticatedUserContext,
  getErrorMessage,
  getErrorStatusCode,
  getRequestLogger,
  logControllerError,
  requireInternalAuthContext,
} from "shared-lib";
import { businessesLogger } from "../logger";
import { BillingUseCase } from "../use-
cases/Billing/billing.use-case";
import {
  parseBillingCancelPlanChange,
  parseBillingCheckout,
  parseBillingPlanChange,
  parseBillingPortal,
  parseBillingSubscriptionQuery,
  parseStripeSignature,
} from "../validation/Billing/billing.request";

const billingRouter = Router();

```

```

  event: "assistant.get_by_id.failed",
  error,
  statusCode: resolvedError.statusCode,
  failureMessage: "Assistants controller request failed",
});
res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
return;
}
}

```

```

router.get("/", getAssistants);
router.post("/create", createAssistant);
router.patch("/:id", updateAssistant);
router.delete("/:id", deleteAssistant);
router.get("/:id", getAssistantById);

```

```
export default router;
```

```

const billingWebhookRouter = Router();

billingRouter.use(requireInternalAuthContext);

async function getPlans(_req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const plans = await BillingUseCase.listPlans();
    res.status(200).json(plans);
  } catch (error) {
    const statusCode = getErrorStatusCode(error);
    logControllerError({
      logger,
      event: "billing.plans.failed",
      error,
      statusCode,
      failureMessage: "Billing plans request failed",
    });
    res.status(statusCode).json({ error: getErrorMessage(error,
"Billing plans failed") });
  }
}

```

```

    }
  }

  async function getSubscription(req: Request, res: Response):
  Promise<void> {
    const logger = getRequestLogger(res, businessesLogger);

    try {
      const userContext = getAuthenticatedUserContext(res);
      const payload = parseBillingSubscriptionQuery(req.query);
      const subscription = await
BillingUseCase.getSubscription(payload, userContext);

      res.status(200).json(subscription);
    } catch (error) {
      const statusCode = getErrorStatusCode(error);
      logControllerError({
        logger,
        event: "billing.subscription.failed",
        error,
        statusCode,
        failureMessage: "Billing subscription request failed",
      });
      res.status(statusCode).json({ error: getErrorMessage(error,
"Billing subscription failed") });
    }
  }

  async function createCheckoutSession(req: Request, res:
Response): Promise<void> {
    const logger = getRequestLogger(res, businessesLogger);

    try {
      const userContext = getAuthenticatedUserContext(res);
      const payload = parseBillingCheckout(req.body);
      const checkoutSession = await
BillingUseCase.createCheckoutSession(
        payload,
        userContext,
        logger
      );

      res.status(201).json(checkoutSession);
    } catch (error) {

```

```

      const statusCode = getErrorStatusCode(error);
      logControllerError({
        logger,
        event: "billing.checkout.failed",
        error,
        statusCode,
        failureMessage: "Billing checkout request failed",
      });
      res.status(statusCode).json({ error: getErrorMessage(error,
"Billing checkout failed") });
    }
  }

  async function changeSubscriptionPlan(req: Request, res:
Response): Promise<void> {
    const logger = getRequestLogger(res, businessesLogger);

    try {
      const userContext = getAuthenticatedUserContext(res);
      const payload = parseBillingPlanChange(req.body);
      const planChange = await
BillingUseCase.changeSubscriptionPlan(
        payload,
        userContext,
        logger
      );

      res.status(200).json(planChange);
    } catch (error) {
      const statusCode = getErrorStatusCode(error);
      logControllerError({
        logger,
        event: "billing.subscription_change.failed",
        error,
        statusCode,
        failureMessage: "Billing subscription plan change request
failed",
      });
      res.status(statusCode).json({
        error: getErrorMessage(error, "Billing subscription plan
change failed"),
      });
    }
  }

```

```

async function cancelSubscriptionPlanChange(req: Request,
res: Response): Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const payload = parseBillingCancelPlanChange(req.body);
    const result = await
BillingUseCase.cancelSubscriptionPlanChange(
  payload,
  userContext,
  logger
);

    res.status(200).json(result);
  } catch (error) {
    const statusCode = getErrorStatusCode(error);
    logControllerError({
      logger,
      event: "billing.subscription_change_cancel.failed",
      error,
      statusCode,
      failureMessage: "Billing subscription plan change
cancellation request failed",
    });
    res.status(statusCode).json({
      error: getErrorMessage(error, "Billing subscription plan
change cancellation failed"),
    });
  }
}

```

```

async function createPortalSession(req: Request, res:
Response): Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const payload = parseBillingPortal(req.body);
    const portalSession = await
BillingUseCase.createPortalSession(payload, userContext,
logger);

```

```

    res.status(201).json(portalSession);
  } catch (error) {
    const statusCode = getErrorStatusCode(error);
    logControllerError({
      logger,
      event: "billing.portal.failed",
      error,
      statusCode,
      failureMessage: "Billing portal request failed",
    });
    res.status(statusCode).json({ error: getErrorMessage(error,
"Billing portal failed") });
  }
}

```

```

function getRawWebhookBody(req: Request): Buffer {
  if (Buffer.isBuffer(req.body)) {
    return req.body;
  }

  const error = new Error("Stripe webhook requires raw body")
as Error & { statusCode: number };
  error.statusCode = 400;
  throw error;
}

```

```

async function receiveStripeWebhook(req: Request, res:
Response): Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const signature = parseStripeSignature(req.header("stripe-
signature"));
    const result = await BillingUseCase.receiveStripeWebhook(
      getRawWebhookBody(req),
      signature,
      logger
    );

    res.status(200).json(result);
  } catch (error) {
    const statusCode = getErrorStatusCode(error);
    logControllerError({
      logger,

```

```

    event: "billing.stripe_webhook.failed",
    error,
    statusCode,
    failureMessage: "Stripe billing webhook request failed",
  });
  res.status(statusCode).json({ error: getErrorMessage(error,
"Stripe webhook failed") });
}
}

```

```

billingRouter.get("/plans", getPlans);
billingRouter.get("/subscription", getSubscription);

```

### Business controller

```

import { Request, Response, Router } from "express";
import {
  getAuthenticatedUserContext,
  getRequestLogger,
  logControllerError,
} from "shared-lib";
import { businessesLogger } from "../logger";
import { BusinessUseCase } from "../use-cases/Business/business.use-case";
import {
  parseBusinessCreate,
  parseBusinessId,
  pickBusinessUpdates,
} from "../validation/Business/business.request";
import { resolveControllerError } from "../errors";

const router = Router();

async function getBusinesses(_req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const allBusinesses = await
BusinessUseCase.getAll(userContext);
    res.json(allBusinesses);
  } catch (error) {
    const resolvedError = resolveControllerError(error);

```

```

    billingRouter.post("/subscription/change",
changeSubscriptionPlan);
    billingRouter.post("/subscription/change/cancel",
cancelSubscriptionPlanChange);
    billingRouter.post("/checkout", createCheckoutSession);
    billingRouter.post("/portal", createPortalSession);
    billingWebhookRouter.post("/", receiveStripeWebhook);

```

```

export { billingWebhookRouter };
export default billingRouter;

```

```

logControllerError({
  logger,
  event: "businesses.list.failed",
  error,
  statusCode: resolvedError.statusCode,
  failureMessage: "Businesses controller request failed",
});
res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
}
}

```

```

async function createBusiness(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const payload = parseBusinessCreate(req.body);
    const newBusiness = await BusinessUseCase.create(payload,
userContext);
    const businessId = newBusiness.get("id");
    const platformUsernames =
newBusiness.get("platform_usernames");

```

```

    logger.info(
      {
        event: "business.created",
        businessId,
        configuredPlatforms: Object.keys(platformUsernames ||
{}).filter(function hasValue(
          key: string

```

```

    ): boolean {
        return Boolean((platformUsernames as Record<string,
unknown>)?.[key]);
    },
    },
    "Business created"
);

res.status(201).json(newBusiness);
return;
} catch (error) {
    const resolvedError = resolveControllerError(error);
    logControllerError({
        logger,
        event: "business.create.failed",
        error,
        statusCode: resolvedError.statusCode,
        failureMessage: "Businesses controller request failed",
    });
    res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
    return;
}
}

```

```

async function deleteBusiness(req: Request, res: Response):
Promise<void> {
    const logger = getRequestLogger(res, businessesLogger);

    try {
        const userContext = getAuthenticatedUserContext(res);
        const businessId = parseBusinessId(req.params.id);
        const deleted = await BusinessUseCase.delete(businessId,
userContext);

        if (!deleted) {
            res.status(404).json({ error: "Business not found" });
            return;
        }

        logger.info(
            {
                event: "business.deleted",
                businessId,

```

```

    },
    "Business deleted"
);

res.sendStatus(200);
return;
} catch (error) {
    const resolvedError = resolveControllerError(error);
    logControllerError({
        logger,
        event: "business.delete.failed",
        error,
        statusCode: resolvedError.statusCode,
        failureMessage: "Businesses controller request failed",
    });
    res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
    return;
}
}

async function updateBusiness(req: Request, res: Response):
Promise<void> {
    const logger = getRequestLogger(res, businessesLogger);

    try {
        const userContext = getAuthenticatedUserContext(res);
        const businessId = parseBusinessId(req.params.id);
        const updates = pickBusinessUpdates(req.body);

        if (Object.keys(updates).length === 0) {
            res.status(400).json({ error: "No valid fields provided for
update" });
            return;
        }

        const updated = await BusinessUseCase.update(businessId,
updates, userContext);

        if (!updated) {
            res.status(404).json({ error: "Business not found" });
            return;
        }
    }
}

```

```

logger.info(
  {
    event: "business.updated",
    businessId,
    updatedFields: Object.keys(updates),
  },
  "Business updated"
);

res.json(updated);
return;
} catch (error) {
  const resolvedError = resolveControllerError(error);
  logControllerError({
    logger,
    event: "business.update.failed",
    error,
    statusCode: resolvedError.statusCode,
    failureMessage: "Businesses controller request failed",
  });
  res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
  return;
}
}

```

```

async function getBusinessById(req: Request, res: Response):
Promise<void> {
  const logger = getRequestLogger(res, businessesLogger);

  try {
    const userContext = getAuthenticatedUserContext(res);
    const businessId = parseBusinessId(req.params.id);
    const foundBusiness = await
BusinessUseCase.getById(businessId, userContext);

    if (!foundBusiness) {
      res.status(404).json({ error: "Business not found" });
      return;
    }

    res.json(foundBusiness);
    return;
  } catch (error) {

```

```

const resolvedError = resolveControllerError(error);
logControllerError({
  logger,
  event: "business.get_by_id.failed",
  error,
  statusCode: resolvedError.statusCode,
  failureMessage: "Businesses controller request failed",
});
res.status(resolvedError.statusCode).json({ error:
resolvedError.message });
return;
}
}

router.get("/", getBusinesses);
router.post("/create", createBusiness);
router.delete("/:id", deleteBusiness);
router.patch("/:id", updateBusiness);
router.get("/:id", getBusinessById);

export default router;

```