Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ ТА РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ТРАНСКРИБАЦІЇ ЖИВОГО МОВЛЕННЯ З ФУНКЦІЯМИ АНАЛІЗУ ДАНИХ

RESEARCH AND DEVELOPMENT OF AN INTELLIGENT SYSTEM FOR SPOKEN LANGUAGE TRANSCRIPTION WITH DATA ANALYSIS FUNCTIONS

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Герук Максим Сергійович
_____
(підпис)
Керівник: к.т.н., доцент
Лук'янчук Юрій Анатолійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Герук Максим Сергійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «*Дослідження та розробка інтелектуальної системи транскрибації живого мовлення з функціями аналізу даних*»
Керівник роботи *к.т.н., доцент Лук'янчук Юрій Анатолійович*
затверджені наказом закладу вищої освіти від «*14*» *травня* *2025* р. № 255/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«15» грудня 2025 р.*
3. Вихідні дані до роботи: *статті, дослідження вітчизняних та закордонних авторів в даній області, сучасні методи і засоби розробки, технічна документація технологій розробки*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
5. Перелік графічного матеріалу: *1. Схема бізнес-процесу обробки вхідного дзвінка. 2. Схема діяльності процесу обробки дзвінка. 3. Діаграма класів. 4. Інтерфейс бічної панелі навігації. 5. Загальний вигляд панелі управління (Dashboard). 6. Сторінка авторизації користувача. 7. Меню вибору мови інтерфейсу. 8. Інтерфейс журналу дзвінків з активними фільтрами. 9. Інтерфейс детальної аналітики дзвінка. 10. Інтерфейс глобального списку автоматично створених завдань. 11. Панель адміністрування користувачів та налаштування безпеки. 12. Система контролю ролями. 13. Інтерфейс інтеграції із CRM та телефонією. 14. Інтерфейс конфігурації поведінки штучного інтелекту. 15. Схема розгортання компонентів системи.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблематики за темою роботи та постановка завдань дослідження	*Лук'янчук Ю. А.*		
Теоретичне дослідження та практична реалізація предмету дослідження	*Лук'янчук Ю. А.*		
Експеременталье дослідження результативності предмету дослідження	*Лук'янчук Ю. А.*		
Показник запозичень тексту	%		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1.	*Обґрунтування теми дослідження*	*до 14.05.2025*	
2.	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 21.08.2025*	
3.	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 05.09.2025*	
4.	*Розробити функціональну схему роботи програмного продукту*	*до 30.09.2025*	
5.	*Описати засоби розробки об'єкта проектування*	*до 09.10.2025*	
6.	*Практична реалізація об'єкта проектування*	*до 20.10.2025*	
7.	*Розробити методику для проведення експерименту*	*до 29.10.2025*	
8.	*Провести аналіз результатів експерименту*	*до 12.11.2025*	
9.	*Формування списку використаних джерел*	*до 17.11.2025*	
10.	*Оформлення ілюстративного матеріалу*	*до 25.11.2025*	
11.	*Інструментальна перевірка на академічний плагіат*	*до 03.12.2025*	
12.	*Здача чистового варіанту кваліфікаційної роботи на кафедру*	*до 05.12.2025*	

Здобувач вищої освіти _____ Максим ГЕРУК

Керівник роботи _____ Юрій ЛУК'ЯНЧУК

АНОТАЦІЯ

Герук М. С. Дослідження та розробка інтелектуальної системи транскрибації живого мовлення з функціями аналізу даних. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, 3 розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено комплексний аналіз предметної області та існуючих комерційних рішень, що дозволило ідентифікувати проблеми ручної обробки аудіоданих та обґрунтувати технічну доцільність створення власного Middleware-рішення. Аргументовано вибір технологічного стеку на базі екосистеми JavaScript (Node.js, Next.js) та хмарних сервісів штучного інтелекту для вирішення поставлених завдань. У другому розділі спроєктовано архітектуру системи у вигляді модульного моноліту, розроблено реляційну модель бази даних SQLite та алгоритми інтеграції з API KommoCRM і Phonet. Деталізовано логіку роботи модулів автоматичної транскрибації (Whisper), спікер-діарізації та семантичного аналізу розмов із використанням моделі GPT-4o та механізму Function Calling. У третьому розділі описано етапи практичної реалізації та розгортання програмного комплексу SmartIO CRM, а також проведено навантажувальне тестування системи. Виконаний аналіз ефективності впровадження продемонстрував високу відмовостійкість рішення та скорочення часу обробки ліда на 85 % завдяки автоматизації рутинних операцій.

Ключові слова: транскрибація, аналіз даних, штучний інтелект, CRM, KommoCRM, OpenAI, GPT, Whisper, Next.js, Node.js, SQLite, Function Calling, автоматизація продажів, діарізація, Middleware.

ABSTRACT

Maksym Heruk. Research and Development of an Intelligent System for Live Speech Transcription with Data Analysis Capabilities. Manuscript.

Master's Qualification Thesis, EP «Computer Science», Specialty 122 «Computer Science». Lutsk National Technical University. Lutsk, 2025.

The Master's thesis consists of an introduction, 3 chapters, conclusions, a list of references, and appendices. The first chapter analyzes the subject domain and existing commercial solutions, identifying issues with manual audio data processing and substantiating the technical feasibility of creating a proprietary Middleware solution. The choice of a technology stack based on the JavaScript ecosystem (Node.js, Next.js) and cloud AI services to address the set tasks is argued. The second chapter designs the system architecture as a modular monolith, developing a relational SQLite database model and integration algorithms with KommoCRM and Phonet APIs. The logic for automatic transcription (Whisper), speaker diarization, and semantic conversation analysis using the GPT-4o model and Function Calling mechanism is detailed.The third chapter describes the stages of practical implementation and deployment of the SmartIO CRM software complex, as well as load testing of the system. The efficiency analysis demonstrated high system fault tolerance and an 85 % reduction in lead processing time due to the automation of routine operations.

Keywords: transcription, data analysis, artificial intelligence, CRM, KommoCRM, OpenAI, GPT, Whisper, Next.js, Node.js, SQLite, Function Calling, sales automation, diarization, Middleware.

ЗМІСТ

ВСТУП

Сучасний ринок реалізації автотранспорту, особливо в сегментах кредитування та лізингу, характеризується високою складністю бізнес-процесів та жорсткою конкуренцією, де швидкість прийняття рішень є критичним фактором. Існуючі CRM-системи та засоби IP-телефонії ефективно фіксують факти комунікації, проте накопичують значні обсяги неструктурованих аудіоданих, ручний аналіз яких є часовитратним і призводить до втрати важливої комерційної інформації. Світові тенденції у сфері автоматизації продажів свідчать про перехід від простих систем обліку до інтелектуальних платформ, що використовують великі мовні моделі (LLM) для обробки природної мови в реальному часі. Наявна прогалина у знаннях та технологіях стосується відсутності доступних інструментів, які б інтегрували гетерогенні системи телефонії та CRM через проміжне програмне забезпечення для глибокого семантичного аналізу розмов українською мовою.

Актуальність теми зумовлена необхідністю трансформації аудіопотоку у структуровані дані для миттєвої кваліфікації лідів та нівелювання людського фактора в процесі продажів. Розробка інтегрованого рішення дозволить автоматизувати рутинні операції, знизити операційні витрати та підвищити конверсію угод, що є критично важливим для розвитку галузі автокредитування в умовах нестабільної економіки.

Метою роботи є підвищення операційної ефективності продажів шляхом розробки та впровадження інтегрованої системи автоматичної транскрипції мовлення, яка забезпечує миттєвий аналіз та структурування клієнтських запитів.

Для досягнення поставленої мети необхідно вирішити низку завдань:

– провести аналіз предметної області, існуючих бізнес-процесів та виявити недоліки ручної обробки даних у сфері автокредитування;

– обґрунтувати вибір технологічного стеку та архітектури програмного забезпечення для створення Middleware-рішення;

– розробити алгоритми інтеграції системи з API KommoCRM, IP-телефонії та сервісів штучного інтелекту;

– реалізувати програмні модулі для автоматичної транскрибації, діарізації та семантичного аналізу розмов;

– здійснити програмну реалізацію клієнтської та серверної частин системи з використанням технологій Next.js та OpenAI;

– провести тестування розробленого програмного продукту та оцінити його ефективність.

Об’єктом дослідження є процес кваліфікації лідів та обробки клієнтських звернень у відділах продажу автотранспорту.

Предметом дослідження є методи, моделі та інформаційні технології автоматизації аналізу вербальної комунікації з використанням великих мовних моделей.

Наукова новизна одержаних результатів полягає в тому, що удосконалено метод автоматизованої кваліфікації лідів за рахунок інтеграції механізму Function Calling великих мовних моделей, що дозволило, на відміну від існуючих підходів, автоматизувати вилучення структурованих сутностей з неструктурованого аудіопотоку. Також набуло подальшого розвитку застосування архітектури Middleware для інтеграції гетерогенних систем, що забезпечує гнучкість масштабування та незалежність від конкретних вендорів програмного забезпечення.

Практичне значення одержаних результатів визначається розробкою та впровадженням інтелектуальної системи SmartIO CRM. Впровадження системи дозволило скоротити час на обробку одного ліда на 85 % (з 10 до 2 хвилин), покращити точність аналізу потреб клієнта та автоматизувати рутинні операції з ведення CRM.

Апробація результатів кваліфікаційної роботи. Основні положення та результати роботи доповідалися та обговорювалися на V Міжнародній науковопрактичній конференції «Research in Science, Technology and Economics», Люксембург, 10-12 грудня 2025 р. (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасного стану проблеми

Ринок реалізації комерційної та легкової техніки в Україні, особливо в сегментах B2B та B2C, що пов’язані із залученням фінансових інструментів, характеризується високою конкуренцією та підвищеною складністю бізнес-процесів. На відміну від прямих продажів, де цикл угоди може бути відносно коротким, процес, що включає кредит або лізинг, являє собою багатоетапну воронку, яка вимагає узгодженої роботи відділу продажів, служби безпеки та зовнішніх фінансових установ (банків та лізингових компаній).

В умовах нестабільної економічної ситуації швидкість прийняття рішень стає критичним фактором конкурентоспроможності. Ключовим фактором успіху в цій моделі є не лише здатність менеджера презентувати продукт (автотранспорт), але і його компетенція у проведенні первинної фінансової кваліфікації клієнта. Ефективне управління цим процесом безпосередньо впливає на конверсію, операційні витрати та загальну рентабельність компанії. Будь-яка затримка в обробці інформації або втрата даних на етапі першого контакту призводить до того, що клієнт звертається до конкурентів, які здатні надати попереднє рішення швидше.

Центральним елементом управління є воронка продажів (sales funnel), яка в контексті досліджуваної предметної області має чітку декомпозицію на етапи. Основна проблематика, що потребує технологічного вирішення, зосереджена на етапах первинного контакту та кваліфікації. Саме на цій ланці відбувається первинний збір критично важливих даних, що визначають життєздатність угоди. Процес кваліфікації являє собою вербальну комунікацію (телефонну розмову), під час якої менеджер зобов’язаний отримати та коректно інтерпретувати відповіді на стандартизований набір питань, зокрема:

– об’єкт фінансування (тип, стан, вартість техніки, рік випуску);

– тип фінансового інструменту (кредит, фінансовий або оперативний лізинг);

– юридичний статус клієнта (фізична особа, фізична особа-підприємець або юридична особа);

– фінансовий стан (офіційний рівень доходу, наявність першого внеску, поточне кредитне навантаження та кредитна історія);

– терміновість (бажаний термін отримання фінансування та наявність техніки на складі).

Аналіз поточного стану бізнес-процесів дозволив ідентифікувати низку системних «вузьких місць» (bottlenecks), що виникають внаслідок ручної обробки даних. Незважаючи на наявність інфраструктури телефонії та CRM, аудіодані залишаються неструктурованим активом. Це породжує такі операційні проблеми:

– інформаційні втрати та суб'єктивізм с після завершення розмови, менеджер змушений по пам'яті вносити дані до картки клієнта. Враховуючи високе психологічне навантаження та велику кількість дзвінків, ключові фінансові нюанси (наприклад, згадка про прострочення кредиту в минулому) часто втрачаються або спотворюються;

– часові витрати на нецільові ліди – відсутність миттєвого аналізу призводить до витрачання часу на клієнтів, які апріорі не відповідають вимогам фінансових партнерів, наприклад, відсутній перший внесок, що з'ясовується лише на етапі подачі документів у банк;

– складність контролю якості – керівництво не має інструментів для суцільного моніторингу дотримання скриптів кваліфікації, а вибіркове прослуховування дзвінків покриває не більше 5-10 % комунікацій.

Відсутність автоматизованого інструменту аналізу створює операційні, фінансові та стратегічні ризики. Операційні ризики проявляються у збільшенні тривалості циклу угоди та неефективному розподілі ресурсів, коли менеджери витрачають до 30 % робочого часу на адміністративну роботу, замість активних

продажів, так як фінансові ризики пов’язані зі зниженням конверсії та високою вартістю обробки «сміттєвих» лідів.

Для візуалізації проблеми поточний бізнес-процес обробки вхідного дзвінка можна описати у вигляді моделі «AS-IS», наведеній на рисунку 1.1. Дана модель демонструє ключові точки розриву інформаційних потоків.

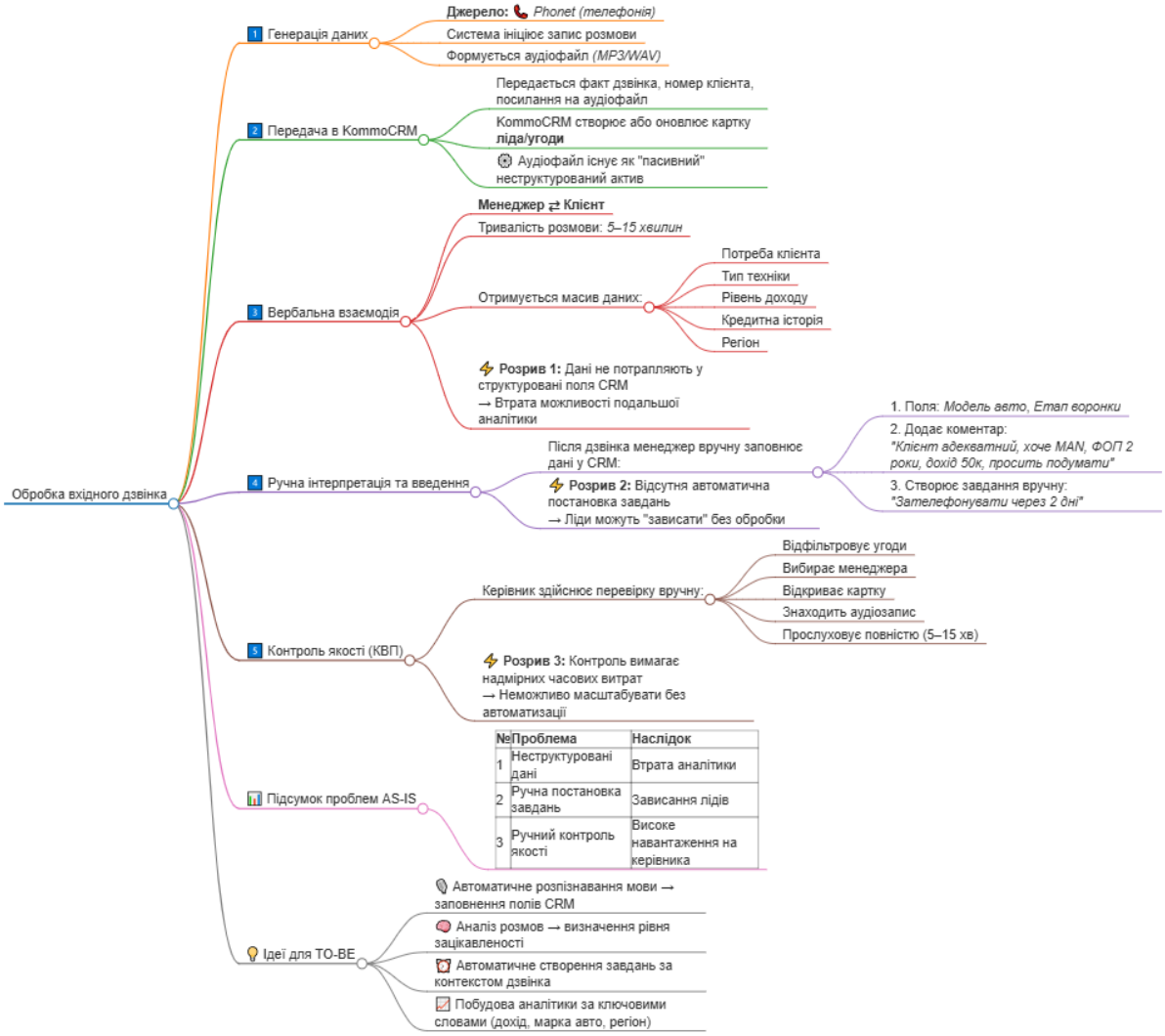

Рисунок 1.1 – Схема бізнес-процесу обробки вхідного дзвінка

Джерело: розроблено автором

Як зображено на рисунку 1.1, існуюча технологічна зв’язка «Телефонія плюс CRM» ефективно вирішує завдання фіксації факту комунікації, але залишає невирішеною проблему структурування її змісту. Аудіофайл зберігається як «неактивний вантаж», до якого звертаються лише у спірних

ситуаціях. Це створює об’єктивну необхідність у розробці проміжної програмної ланки (Middleware), яка б інтегрувала технології розпізнавання мовлення та аналізу тексту в існуючу інфраструктуру, перетворюючи розмови на структуровані дані.

1.2 Аналіз подібних програмних продуктів

Для обґрунтування необхідності розробки спеціалізованого програмного рішення проведено глибокий аудит поточного технологічного стеку компанії та порівняльний аналіз альтернативних комерційних рішень, представлених на ринку України та світу.

Поточна інфраструктура компанії складається з двох базових компонентів, а саме CRM-системи KommoCRM та IP-телефонії Phonet. Сервіс Phonet виконує функцію телекомунікаційного шлюзу, забезпечуючи стабільний зв’язок та запис розмов, однак він не надає вбудованих інструментів для семантичного аналізу їх змісту [1]. KommoCRM виступає як центральна база даних угод, але має обмежені можливості щодо автоматизації на основі змісту розмов. Вбудовані засоби автоматизації реагують лише на зміну статусів або заповнення полів, які менеджер мусить вносити вручну [2].

Для порівняння було детально розглянуто альтернативні рішення на ринку, які можна розділити на дві групи:

– локальні сервіси телефонії з AI-модулями, наприклад, Binotel, Ringostat пропонують базову транскрипцію та аналітику дзвінків. Їхньою перевагою є легкість підключення «з коробки». Проте, основним недоліком є закритість архітектури. Користувач не має можливості впливати на логіку роботи нейромережі, налаштовувати специфічні промпти для розпізнавання фінансової термінології (лізинг, ануїтет, тіло кредиту) або змінювати критерії оцінки якості дзвінка;

– глобальні CRM-платформи, наприклад, Salesforce з модулем Einstein GPT надають потужні інструменти генеративної аналітики та прогнозування.

Однак, вони характеризуються надзвичайно високою вартістю ліцензування та впровадження. Крім того, адаптація таких систем під специфіку локальних бізнес-процесів та українську мову вимагає залучення дороговартісних інтеграторів.

Порівняльну характеристику функціональних можливостей розглянутих рішень наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Функціональна характеристика	Поточний стек (KommoCRM, Phonet, SmartIO CRM)	Локальні рішення (Binotel / Ringostat)	Глобальні платформи (Salesforce)
Базовий функціонал	CRM + Телефонія	Телефонія з аналітикою	Повноцінна екосистема
Транскрипція мовлення	Відсутня	Є (базова, вбудована)	Є (просунута)
Сумаризація діалогів	Відсутня	Є (за шаблонами)	Є (генеративна)
Автоматизація дій (Function Calling)	Відсутня	Відсутня	Обмежена, складна
Гнучкість налаштування ШІ	Відсутня	Низька	Середня
Вартість впровадження	Низька (вже є)	Середня	Висока
Контроль над даними	Повний (при розробці)	Відсутній	Частковий

Аналіз даних таблиці 1.1 свідчить, що жодне з готових рішень не задовольняє повною мірою вимоги до автоматизації процесу кваліфікації лідів у сфері автокредитування. Основні недоліки існуючих продуктів полягають у недостатній гнучкості, тому що «коробкові» рішення не дозволяють кастомізувати системні інструкції для ШІ та відсутності глибокої двосторонньої інтеграції з логікою CRM. Зокрема, жодна з розглянутих систем не дозволяє автоматично створювати завдання (Tasks) у календарі менеджера на основі усних домовленостей, виявлених у тексті розмови.

Враховуючи виявлені обмеження, економічно та технічно доцільною є розробка власного програмного продукту класу Middleware. Такий підхід дозволить зберегти інвестиції в існуючу інфраструктуру (KommoCRM, Phonet), доповнивши її необхідним інтелектуальним функціоналом на базі передових моделей штучного інтелекту, що забезпечить повний контроль над даними та алгоритмами їх обробки.

1.3 Аналіз і вибір технологічного стеку та інструментів для розробки

При розробці інтелектуальної системи, що функціонує як проміжне програмне забезпечення (Middleware) між телефонією, CRM та сервісами ШІ, вибір технологічного стеку є критичним фактором, що визначає продуктивність, масштабованість та вартість підтримки.

На початковому етапі розглядалася класична мікросервісна архітектура на основі мови Python (фреймворк FastAPI) для бекенду та React для фронтенду. Проте, такий підхід призвів би до розриву кодової бази, дублювання типів даних та необхідності підтримки двох окремих середовищ розгортання. Для реалізації проєкту було обрано сучасний уніфікований підхід на базі фреймворку Next.js. Це дозволяє використовувати єдине середовище виконання Node.js та мову програмування TypeScript як для серверної частини (API Routes), так і для клієнтського інтерфейсу [3].

Обґрунтування вибору ключових технологій:

– Node.js та Next.js – вибір обумовлений асинхронною, неблокуючою моделлю вводу-виводу (Event Loop). Оскільки система є I/O-bound, основний час роботи витрачається на очікування відповідей від зовнішніх API OpenAI та CRM, а не на обчислення, така архітектура є найбільш продуктивною для обробки великої кількості одночасних вебхуків [4]. Next.js надає готову структуру для створення серверних функцій (Server Actions), що спрощує розробку;

– TypeScript – забезпечує сувору статичну типізацію. Це є критично важливим для фінансового додатку, де помилка в обробці даних, наприклад у сумі кредиту, неприпустима. Спільні інтерфейси даних між сервером та клієнтом мінімізують ризик помилок інтеграції [5];

– SQLite – обрано як систему управління базами даних завдяки її архітектурі serverless, де база даних зберігається у файлі. Це значно спрощує розгортання в контейнеризованому середовищі Docker, зменшує вимоги до ресурсів сервера та є достатнім для middleware-додатку, що не потребує складних розподілених транзакцій на етапі MVP [6];

– OpenAI API – використано як ядро інтелектуальної обробки. Модель Whisper забезпечує найвищу на ринку точність транскрипції української мови, включаючи суржик та діалектизми. Моделі GPT-4o та GPT-4o-mini дозволяють реалізувати складну логіку діарізації, сумаризації та, що найважливіше, механізм Function Calling для перетворення неструктурованого тексту в структуровані JSON-команди [7].

Для розробки клієнтського інтерфейсу обрано бібліотеку React у поєднанні з CSS-фреймворком Tailwind CSS, що дозволяє швидко створювати адаптивні компоненти адміністративної панелі. Контейнеризація додатку здійснюється за допомогою Docker, що гарантує ідентичність середовищ розробки та промислової експлуатації, спрощуючи процес CI/CD.

1.4 Постановка завдання на кваліфікаційну роботу магістра

Проведений аналіз предметної області, існуючих аналогів та технологічних засобів дозволив сформулювати конкретні завдання на розробку системи. Головною метою є створення програмного комплексу, що автоматизує рутинні процеси обробки телефонних розмов, підвищуючи ефективність роботи відділу продажів.

Відповідно до поставленої мети та виявлених проблем, у кваліфікаційній роботі необхідно вирішити такі завдання:

– провести аналіз предметної області, існуючих бізнес-процесів та виявити недоліки ручної обробки даних у сфері автокредитування;

– обґрунтувати вибір технологічного стеку та архітектури програмного забезпечення для створення Middleware-рішення;

– розробити алгоритми інтеграції системи з API KommoCRM, IP-телефонії та сервісів штучного інтелекту;

– реалізувати програмні модулі для автоматичної транскрибації, діарізації та семантичного аналізу розмов;

– здійснити програмну реалізацію клієнтської та серверної частин системи з використанням технологій Next.js та OpenAI;

– провести тестування розробленого програмного продукту та оцінити його ефективність.

У першому розділі проведено комплексний аналіз предметної області продажів автотранспорту та виявлено, що критичним бар’єром ефективності є ручна обробка неструктурованих аудіоданих, що призводить до втрати до 30 % робочого часу менеджерів та зниження конверсії лідів. Аналіз ринку існуючих рішень показав, що вони не забезпечують необхідної гнучкості в налаштуванні логіки штучного інтелекту для специфічних фінансових завдань або є економічно недоцільними для впровадження.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання

Реалізація системи автоматизованої кваліфікації лідів вимагає створення комплексного програмного продукту, здатного працювати в режимі реального часу з різнорідними потоками даних. Головним архітектурним викликом у даному випадку є необхідність забезпечення надійної взаємодії між компонентами системи, які мають різну природу: синхронним веб-інтерфейсом користувача, асинхронними подіями від сервісу IP-телефонії та ресурсомісткими процесами обробки природної мови. Враховуючи вимоги до масштабованості та швидкості розробки, для проектування системи було обрано архітектурний патерн модульного моноліту. На відміну від розподіленої мікросервісної архітектури, яка часто призводить до надмірної складності в інфраструктурі та оркестрації для проектів середнього масштабу, модульний моноліт дозволяє зберегти чітке розмежування контекстів (Frontend, API, Database Access Layer) в межах єдиної кодової бази. Такий підхід значно спрощує процеси розгортання, тестування та, що найважливіше, забезпечує наскрізну типізацію даних між клієнтською та серверною частинами [8].

Вибір технологічного стеку для реалізації серверної частини базувався на аналізі специфіки навантаження. Система класифікується як I/O-bound, тобто основна частина часу виконання витрачається на очікування відповідей від зовнішніх інтерфейсів прикладного програмування (API) таких як OpenAI, KommoCRM та сервісів телефонії. У цьому контексті використання середовища виконання Node.js є найбільш виправданим завдяки його архітектурі, побудованій на подійно-керованій моделі (Event-driven) та неблокуючому вводі-виводу. На відміну від традиційних багатопотокових серверів, де кожен запит створює новий потік, що споживає пам’ять процесора, Node.js обробляє

тисячі одночасних з’єднань у єдиному потоці, використовуючи механізм Event Loop. Це дозволяє ефективно масштабувати систему при зростанні кількості одночасних дзвінків без необхідності значного збільшення апаратних потужностей.

Важливим аспектом забезпечення надійності програмного коду є вибір мови програмування. Динамічна типізація, притаманна чистому JavaScript, часто стає причиною помилок часу виконання (runtime errors), особливо при роботі зі складними структурами даних, такими як результати діарізації аудіо або вкладені об’єкти завдань CRM. Для вирішення цієї проблеми у проекті застосовано мову TypeScript. Вона дозволяє описувати строгі інтерфейси (Interfaces) та типи даних (Types), що гарантує валідність структур даних на етапі компіляції. Завдяки цьому значно знижується ймовірність виникнення критичних помилок під час експлуатації системи та покращується підтримка коду за рахунок автодоповнення та статичного аналізу [9].

Ядром інтелектуальної підсистеми обрано хмарні сервіси штучного інтелекту. Розгортання локальних великих мовних моделей (LLM) вимагає значних обчислювальних ресурсів, зокрема графічних прискорювачів, що є економічно недоцільним для задач даного класу. Натомість використання API OpenAI забезпечує доступ до передових моделей розпізнавання мовлення та генерації тексту. Зокрема, модель Whisper демонструє високу стійкість до шумів та акцентів, що є критичним для транскрибації телефонних розмов. Для семантичного аналізу застосовано модель GPT-4o, яка підтримує розширений контекст та здатна виявляти приховані наміри клієнта [10]. Особливої уваги заслуговує механізм Function Calling, який дозволяє перетворити генеративну модель з генератора тексту на інструмент вилучення структурованих даних. Це дає можливість отримувати результати аналізу у форматі JSON, що готовий до автоматичної обробки та запису в базу даних, уникаючи ненадійних методів парсингу тексту за допомогою регулярних виразів [11].

Підсистема зберігання даних спроектована з використанням СКБД SQLite. Вибір цієї технології обумовлений її serverless-архітектурою, яка не

вимагає окремого процесу сервера бази даних. Це спрощує конфігурацію середовища розробки та розгортання, дозволяючи зберігати всі дані у єдиному кросплатформному файлі. Для забезпечення продуктивності при паралельному доступі активовано режим WAL (Write-Ahead Logging), який дозволяє одночасне читання та запис даних, що нівелює традиційні обмеження файлових баз даних [12].

Візуалізація алгоритму роботи системи представлена на діаграмі діяльності, яка відображає послідовність дій від моменту завершення дзвінка до фінального оновлення даних у CRM-системі (рис. 2.1).

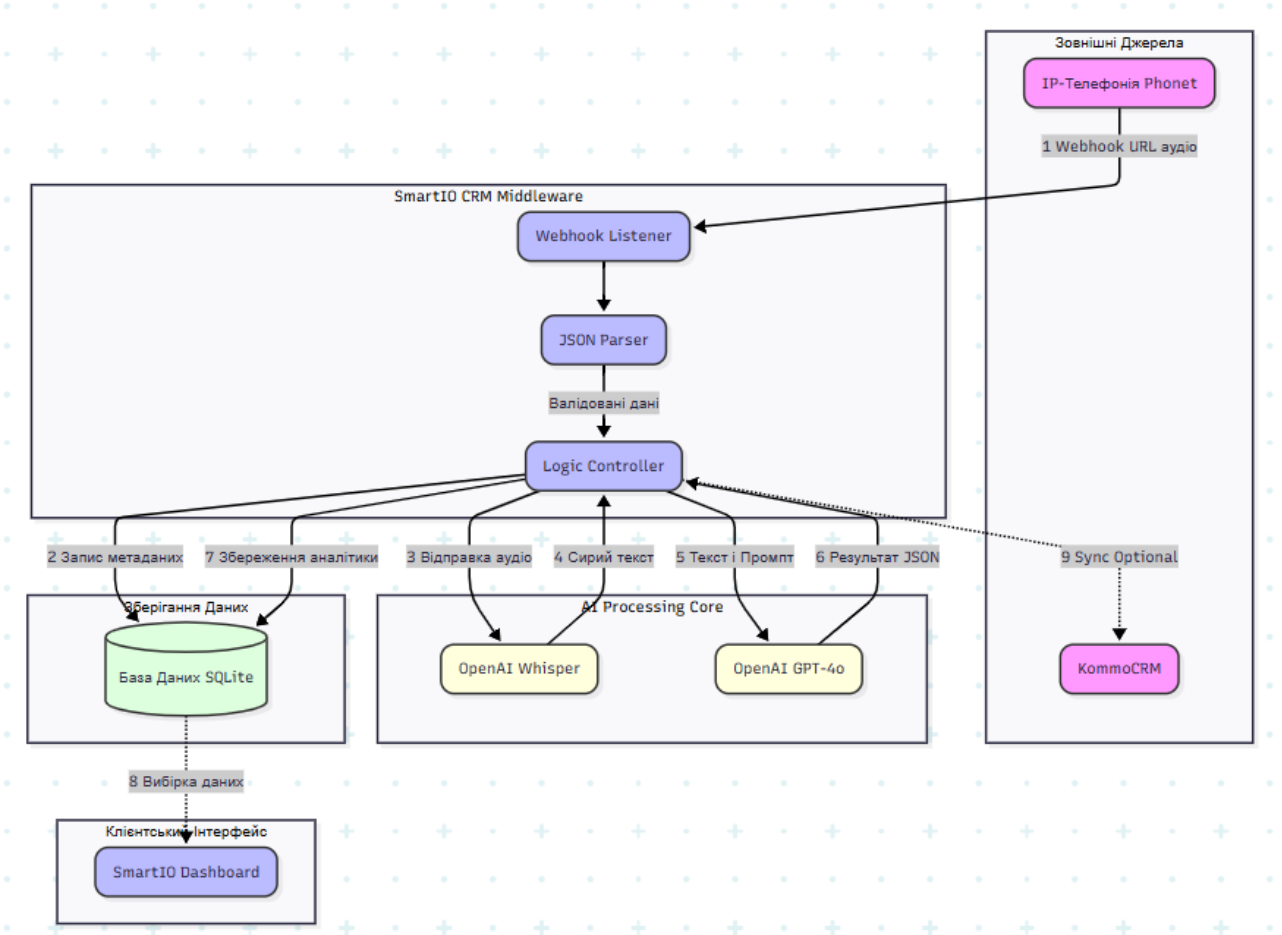

Рисунок 2.1 – Схема діяльності процесу обробки дзвінка

Джерело: розроблено автором

Відповідно до наведеної схеми, процес обробки є повністю автоматизованим та ініціюється зовнішньою подією (вебхуком). Система

використовує паттерн «Конвеєр» (Pipeline), де дані послідовно проходять етапи завантаження, транскрибації, діарізації та аналізу. Центральний контролер логіки забезпечує передачу контексту між етапами та обробку можливих виняткових ситуацій.

Статична структура програмного забезпечення відображена на діаграмі класів (рис. 2.2), яка визначає основні типи даних та відношення між ними.

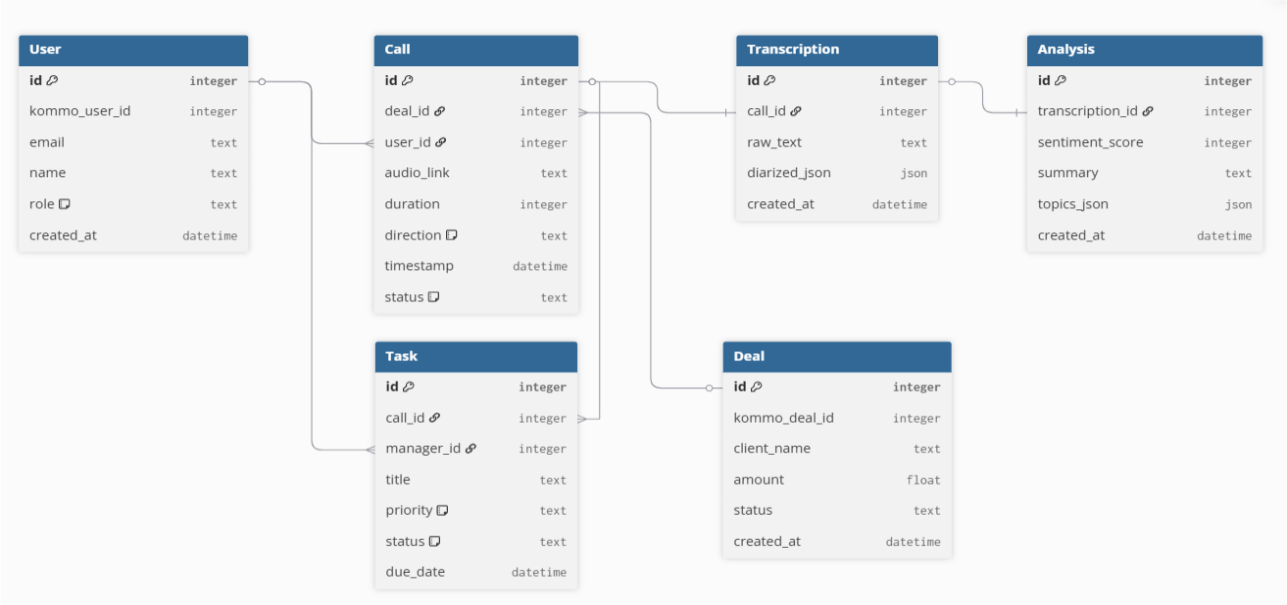

Рисунок 2.2 – Діаграма класів

Джерело: розроблено автором

Об’єктна модель системи базується на принципах предметно-орієнтованого проектування (DDD). Виділено ключові сутності, такі як «Користувач», «Дзвінок», «Транскрипція» та «Аналіз», що дозволяє чітко розмежувати відповідальність модулів та забезпечити цілісність даних при масштабуванні функціоналу.

2.2 Проектування структури бази даних

Ефективність роботи інформаційної системи значною мірою залежить від правильно спроектованої схеми бази даних. Для забезпечення нормалізації даних та уникнення надмірності було розроблено реляційну модель,

реалізовану засобами SQLite. Використання об’єктно-реляційного відображення (ORM) Prisma дозволило описати схему даних у декларативному стилі, що спрощує міграції та підтримку бази даних (додаток Б).

Таблиця User виступає центральним сховищем інформації про авторизованих користувачів системи. Вона забезпечує зв’язок між внутрішніми ідентифікаторами системи та обліковими записами зовнішньої CRM. Детальна структура таблиці наведена в таблиці 2.1.

Таблиця 2.1 – Структура таблиці User (Менеджери)

Поле	Тип даних	Характеристики
id	Integer	Первинний ключ (Primary Key), автоінкремент
kommo_user_id	Integer	Зовнішній ідентифікатор користувача з KommoCRM, унікальне поле
name	Text	Ім’я менеджера

Для фіксації метаданих комунікацій спроектовано таблицю Call. Вона зберігає технічну інформацію про дзвінок, включаючи тривалість, напрямок та посилання на аудіофайл, а також слугує сполучною ланкою між менеджером та угодою. Характеристики полів описано в таблиці 2.2.

Таблиця 2.2 – Структура таблиці Call

Поле	Тип даних	Характеристики
id	Integer	Первинний ключ
deal_id	Integer	Зовнішній ключ до таблиці Deal
user_id	Integer	Зовнішній ключ до таблиці User
audio_link	Text	URL-посилання на аудіофайл запису
duration	Integer	Тривалість розмови в секундах
direction	Text	Напрямок дзвінка (inbound/outbound)
status	Text	Статус обробки (new/processing/done/error)

Зберігання результатів роботи нейромережі реалізовано через таблицю Transcription. Оскільки обсяг текстових даних може бути значним, а структура діалогу (поділ на репліки) є складною, для поля diarized_json використано текстовий тип даних, що містить серіалізований JSON-об’єкт. Це дозволяє

зберігати гнучку структуру даних без порушення нормальної форми бази даних. Структура таблиці представлена в таблиці 2.3.

Таблиця 2.3 – Опис структури таблиці Transcription (Транскрипції)

Поле	Тип даних	Характеристики
id	Integer	Первинний ключ, автоінкремент
call_id	Integer (ForeignKey)	Зовнішній ключ, зв’язок «один-до-одного» з Call id
raw_text	Text	«Сирий» текст, отриманий від Whisper
diarized_text	Text (JSON)	Текст, оброблений GPT-4o з діарізацією, збережений як JSON

Запропонована структура бази даних забезпечує необхідний рівень абстракції для зберігання результатів аналізу та дозволяє ефективно виконувати пошукові запити при формуванні звітів у адміністративній панелі [13].

2.3 Практична реалізація об’єкта проектування

Початковим етапом практичної реалізації програмного комплексу SmartIO CRM стало розгортання та налаштування програмного середовища. Оскільки архітектура системи базується на екосистемі JavaScript, критично важливим кроком було встановлення платформи Node.js, яка забезпечує середовище виконання коду поза браузером та дозволяє використовувати сучасні інструменти збірки. Для забезпечення стабільності та сумісності з бібліотеками було обрано актуальну LTS-версію Node.js для операційної системи Windows. Після завершення процесу інсталяції проведено верифікацію коректності налаштування системних змінних середовища та перевірку наявності пакетного менеджера npm, який є необхідним інструментом для управління залежностями проекту. Успішне виконання команд підтверджує коректність конфігурації робочого оточення та гарантує сумісність обраних версій інструментів із запланованим технологічним стеком. Перевірку здійснено шляхом виконання відповідних команд у терміналі (ліст. 2.1).

Лістинг 2.1 – Команда перевірки версій Node.js та npm

```
C:\Users\Admin>node --version
v20.11.0
C:\Users\Admin>npm --version
10.2.4
```

кінець лістингу 2.1

Наступним кроком стала безпосередня ініціалізація проекту на базі фреймворку Next.js. Для цього було використано утиліту create-next-app, яка автоматизує процес розгортання базової файлової структури. Під час конфігурації прийнято стратегічне рішення використовувати архітектуру App Router. Це дозволяє використовувати React Server Components за замовчуванням, що забезпечує оптимізацію продуктивності шляхом рендерингу компонентів на сервері та зменшення навантаження на клієнтський пристрій. Також було активовано підтримку TypeScript для забезпечення суворої типізації даних на всіх етапах розробки, як показано в лістингу 2.2.

Лістинг 2.2 – Ініціалізація проекту SmartIO CRM

```
npx create-next-app@latest smartio-crm
√ Would you like to use TypeScript? ... Yes
√ Would you like to use ESLint? ... Yes
√ Would you like to use Tailwind CSS? ... Yes
√ Would you like to use `src/` directory? ... Yes
√ Would you like to use App Router? (recommended) ... Yes
√ Would you like to customize the default import alias? ... No
```

кінець лістингу 2.2

Для побудови сучасного, доступного та естетичного інтерфейсу користувача (UI) інтегровано бібліотеку компонентів Shadcn UI. Ця бібліотека базується на примітивах Radix UI [14], що гарантує відповідність стандартам доступності (WAI-ARIA), та дозволяє гнучко адаптувати компоненти під унікальні потреби бізнесу. Додатково було встановлено пакет lucide-react для відображення векторних іконок та утиліти для роботи з CSS-класами (ліст. 2.3).

Лістинг 2.3 – Встановлення UI-залежностей та бібліотек

```
npx shadcn-ui@latest init
npm install lucide-react class-variance-authority clsx tailwind-merge
```

кінець лістингу 2.3

Фінальним етапом підготовки стало налаштування глобальної стилізації проекту. У файлі globals.css визначено базові змінні CSS для підтримки світлої та темної тем оформлення [15]. Це забезпечує централізоване керування кольоровою палітрою всього додатку та дозволяє легко змінювати візуальний стиль без необхідності редагування окремих компонентів (ліст. 2.4).

Лістинг 2.4 – Конфігурація глобальних стилів (globals.css)

```
@tailwind base;
@tailwind components;
@tailwind utilities;
@layer base {
:root {
    --background: 0 0% 100%;
    --foreground: 222.2 84% 4.9%;
    --card: 0 0% 100%;
    --primary: 222.2 47.4% 11.2%;
    --radius: 0.5rem; }
.dark {
    --background: 222.2 84% 4.9%;
    --foreground: 210 40% 98%;
  }
}
```

кінець лістингу 2.4

Таким чином, було сформовано надійний технологічний фундамент, що дозволяє перейти до етапу безпосередньої розробки компонентів системи.

Після налаштування архітектури було розпочато реалізацію основних модулів системи. Основою навігаційної структури веб-додатку стала бічна панель (Sidebar), яка забезпечує доступ до ключових розділів: аналітичних дашбордів, журналів дзвінків та налаштувань. Реалізація компоненту Sidebar використовує глобальний стан додатка (Zustand) для керування режимами відображення. Панель підтримує два стани, а саме розгорнутий (повна ширина)

та згорнутий (лише іконки), що забезпечує зручність використання на екранах різного розміру (ліст. 2.5).

Лістинг 2.5 – Реалізація логіки компоненту Sidebar

```
export function Sidebar() {
  const sidebar = useStore(useSidebarToggle, (state) => state);
  if (!sidebar) return null;
  return (
    <aside
      className={cn(
        "fixed top-0 left-0 z-20 h-screen -translate-x-full lg:translate-
x-0 transition-[width] ease-in-out duration-300",
        sidebar?.isOpen === false ? "w-[90px]" : "w-72"
      )}>
      <div className="relative h-full flex flex-col px-3 py-4 overflow-y-
auto shadow-md dark:shadow-zinc-800">
        <div className="mb-6 mt-2 flex items-center justify-center">
          <Link href="/dashboard" className="flex items-center gap-2">
            <Activity className="w-6 h-6 text-primary" />
            <h1
              className={cn(
                "font-bold text-lg whitespace-nowrap transition-
[transform,opacity,display] ease-in-out duration-300",
                sidebar?.isOpen === false
                  ? "hidden opacity-0"
                  : "block opacity-100"
              )}>
              SmartIO CRM
            </h1>
          </Link>
      </div>
    </aside>
  );
}
```

кінець лістингу 2.5

Візуалізація реалізованої бічної панелі демонструє чітку ієрархію навігаційних елементів, згрупованих за функціональним призначенням, що полегшує орієнтацію користувача в системі, як зображено на рисунку 2.3.

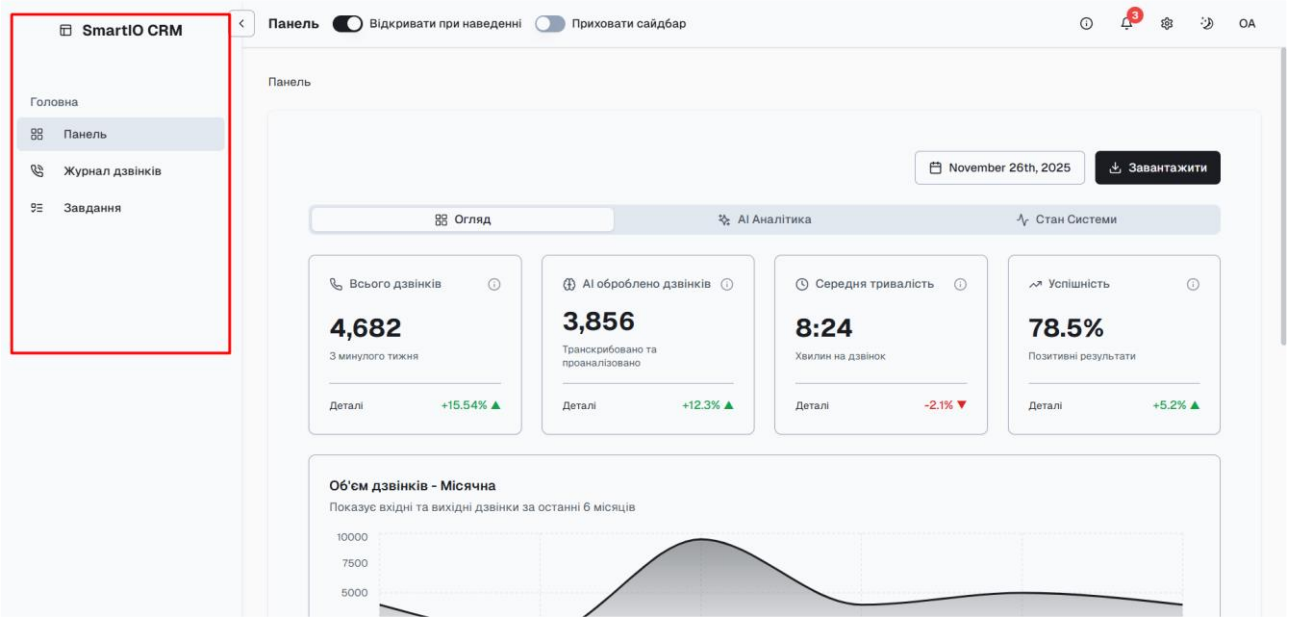

Рисунок 2.3 – Інтерфейс бічної панелі навігації

Джерело: розроблено автором

Центральним елементом системи є Головна сторінка (Dashboard). Вона реалізована як захищений маршрут, доступ до якого мають лише авторизовані користувачі. Верстка сторінки базується на Grid-системі Tailwind CSS, що дозволяє адаптивно розміщувати віджети статистики та графіки [16]. Програмна структура сторінки наведена в лістингу 2.6.

Лістинг 2.6 – Структура сторінки Dashboard

```
export default function DashboardPage() {
  return (
    <ContentLayout title="Dashboard">
      <Breadcrumb>
        <BreadcrumbItem>
          <BreadcrumbLink asChild> <Link href="/">Home</Link>
</BreadcrumbLink>
        </BreadcrumbItem>
        <BreadcrumbSeparator />
        <BreadcrumbItem>
          <BreadcrumbPage>Dashboard</BreadcrumbPage>
        </BreadcrumbItem>
      </Breadcrumb>
  <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4"> </div>
    </ContentLayout>);}
```

кінець лістингу 2.6

Для відображення ключових показників ефективності (KPI) було розроблено універсальний компонент MetricCard. Він приймає дані про поточні значення, тренди та зміни у відсотках, візуалізуючи їх у зручному форматі карток, як показано в лістингу 2.7.

Лістинг 2.7 – Компонент метричної картки (MetricCard.tsx)

```
export function MetricCard({ title, value, change, trend, description,
icon: Icon }: MetricCardProps) {
  return (<Card className="p-6"> <div className="flex items-center
justify-between space-y-0 pb-2"> <h3 className="tracking-tight text-sm
font-medium">{title}</h3> <Icon className="h-4 w-4 text-muted-foreground"
/></div>
      <div className="flex flex-col gap-1">
        <div className="text-2xl font-bold">{value}</div>
        <div className="flex items-center text-xs text-muted-foreground">
<span className={cn(trend === 'up' ? "text-green-500" : "text-red-500",
"flex items-center font-medium")}> {change} </span> <span className="ml-
1">{description}</span> </div> </div> </Card>);}
```

кінець лістингу 2.7

Інтеграція розроблених компонентів, таких як MetricCard, відбувається на рівні сторінки DashboardPage, де формується адаптивна сітка для їх розміщення. Для забезпечення підтримки багатомовності інтерфейсу було використано бібліотеку next-intl. Це дозволяє автоматично підставляти відповідні текстові значення українською або англійською мовою у заголовки та описи карток залежно від налаштувань користувача, зберігаючи при цьому єдину структуру коду. Крім того, застосування модульної архітектури забезпечує легку масштабованість панелі, дозволяючи додавати нові аналітичні віджети без необхідності суттєвих змін у верстці сторінки. Адаптивність інтерфейсу гарантує коректне відображення ключових показників ефективності як на широкоформатних моніторах, так і на мобільних пристроях. Фінальний результат компонування всіх елементів інтерфейсу наведено на рисунку 2.4.

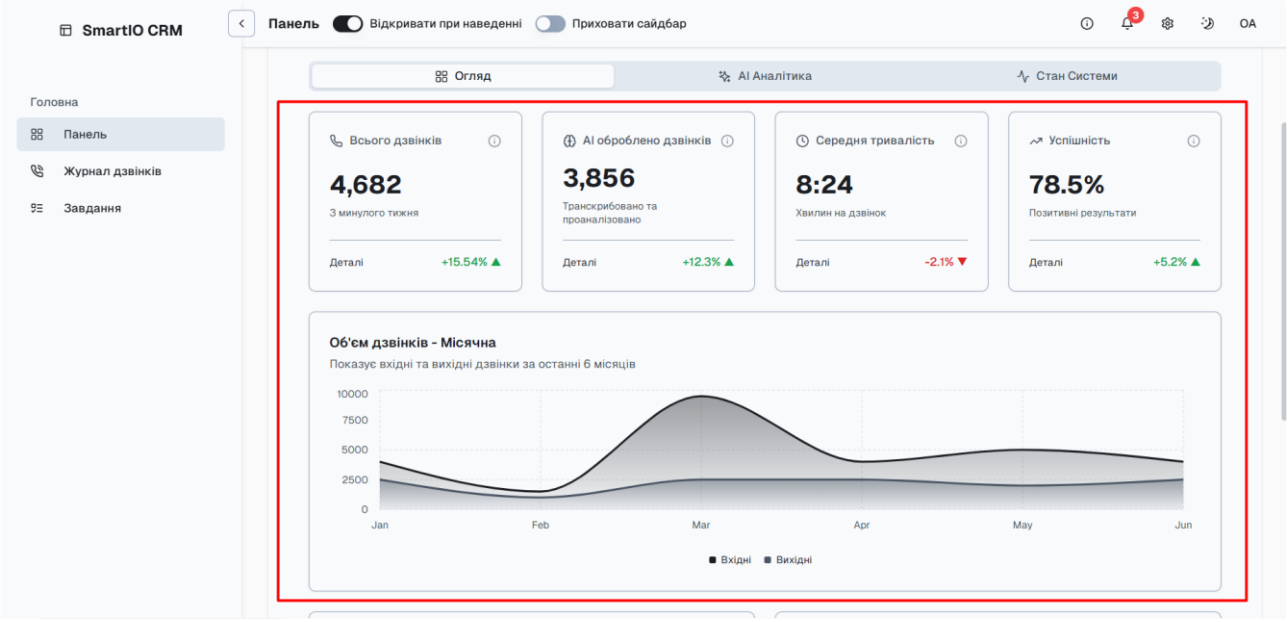

Рисунок 2.4 – Загальний вигляд панелі управління (Dashboard)

Джерело: розроблено автором

Критично важливим аспектом безпеки є модуль автентифікації. Форма входу розроблена з використанням бібліотеки react-hook-form для керування станом полів та zod [17] для валідації введених даних на стороні клієнта. Це дозволяє миттєво інформувати користувача про помилки формату email або пароля ще до відправки запиту на сервер (ліст. 2.8).

Лістинг 2.8 – Реалізація форми входу

```
<form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
  <FormField control={form.control}
    name="email"
    render={({ field }) => (
      <FormItem>
        <FormLabel>Email</FormLabel> <FormControl>
         <Input placeholder="name@example.com" {...field} />
        </FormControl>
        <FormMessage />
      </FormItem>)} />
  <Button type="submit" className="w-full" disabled={isLoading}>
    {isLoading && <Icons.spinner className="mr-2 h-4 w-4 animate-spin"
/>} Sign In
  </Button>
</form>
```

кінець лістингу 2.8

Інтеграція наведеної програмної логіки з візуальними компонентами забезпечує надійну та зручну роботу механізму входу. Система автоматично перевіряє валідність введених даних у реальному часі, підсвічуючи поля з помилками та блокуючи кнопку відправки до моменту виправлення неточностей, що мінімізує кількість некоректних запитів до сервера. Візуальний результат реалізації цього модуля, виконаний у загальній стилістиці системи, наведено на рисунку 2.5.

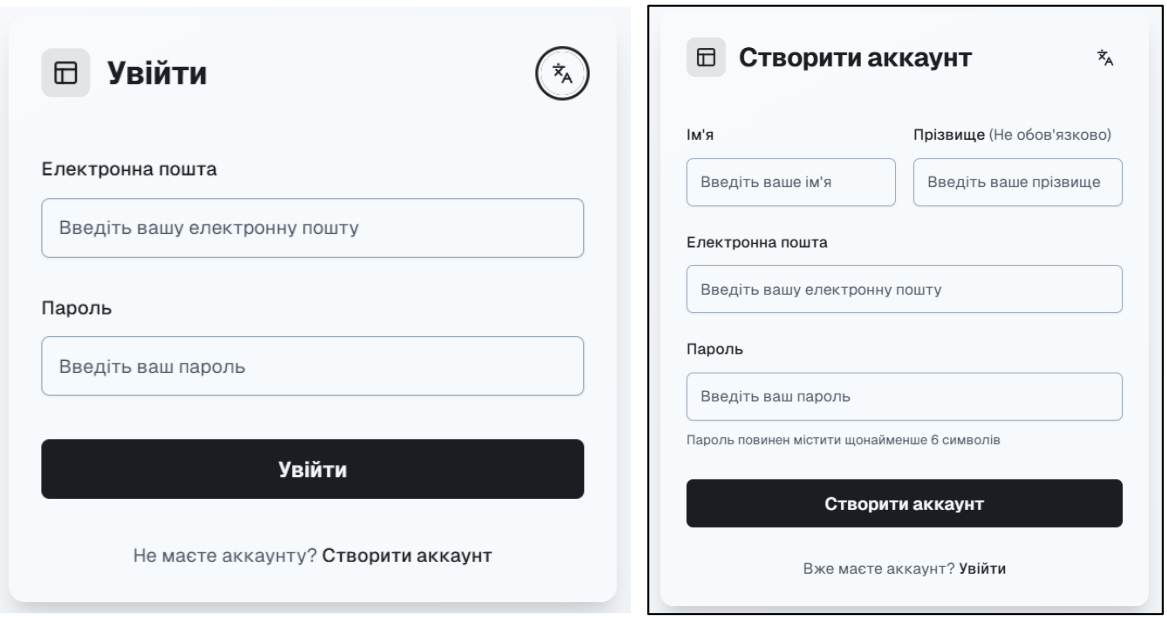

Рисунок 2.5 – Сторінка авторизації користувача

Джерело: розроблено автором

Для забезпечення відповідності вимогам доступності та локалізації, реалізовано механізм перемикання мови інтерфейсу. Обрана стратегія маршрутизації забезпечує автоматичну синхронізацію мовного контексту з URL-адресою, що дозволяє коректно зберігати стан додатка при обміні посиланнями. Додатково реалізовано механізм збереження налаштувань у файлах cookie, гарантуючи персистентність вибору мови при наступних сесіях роботи з системою. Компонент LanguageToggle використовує можливості бібліотеки next-intl для динамічної зміни локалі без перезавантаження сторінки, що значно покращує досвід користувача (ліст. 2.9).

Лістинг 2.9 – Компонент перемикача мов

```
export function LanguageToggle() {
  const t = useTranslations("Common");
  return (
    <DropdownMenu>
      <DropdownMenuTrigger asChild> <Button variant="outline"
size="icon">
       <Languages className="h-[1.2rem] w-[1.2rem]" />
        </Button>
      </DropdownMenuTrigger>
      <DropdownMenuContent align="end">
        <DropdownMenuItem onClick={() => setUserLocale("en")}>
          English
        </DropdownMenuItem>
        <DropdownMenuItem onClick={() => setUserLocale("uk")}>
          Українська
        </DropdownMenuItem>
      </DropdownMenuContent>
    </DropdownMenu>
  );
}
```

кінець лістингу 2.9

Розроблений компонент було інтегровано у верхню панель, що забезпечує швидкий доступ до налаштувань локалізації з будь-якої сторінки додатку. Візуальна реалізація випадаючого меню виконана в єдиному стилі з іншими елементами інтерфейсу, гарантуючи цілісність користувацького досвіду. Для покращення юзабіліті додано візуальну індикацію поточної активної мови за допомогою галочки або зміни кольору шрифту, що дозволяє користувачеві миттєво зорієнтуватися в налаштуваннях. Саме меню є повністю адаптивним і автоматично підлаштовує своє позиціонування відносно країв екрана, запобігаючи виходу за межі видимій області на мобільних пристроях. Результат роботи механізму представлено на рисунку 2.6.

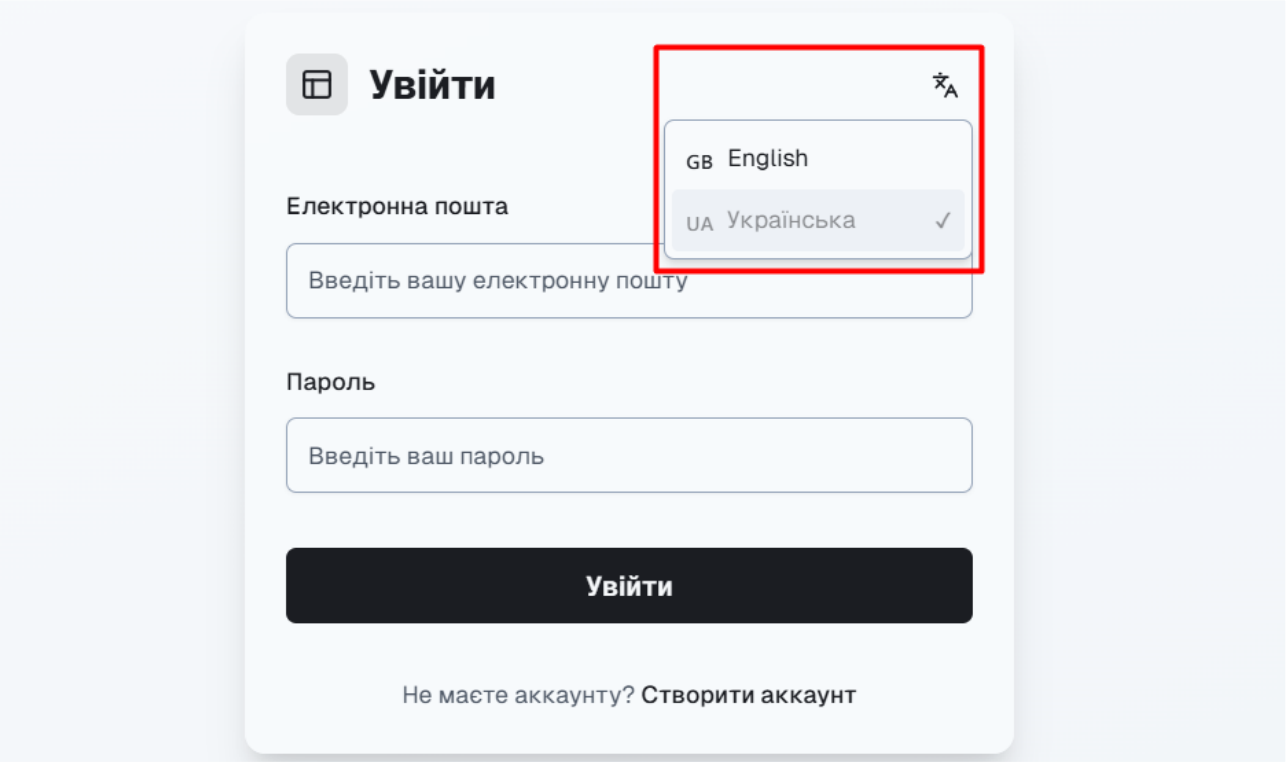

Рисунок 2.6 – Меню вибору мови інтерфейсу

Джерело: розроблено автором

Таким чином, реалізація базових компонентів інтерфейсу та системи автентифікації сформувала надійний архітектурний каркас веб-додатку. Це створило необхідний фундамент для подальшої імплементації спеціалізованих функціональних модулів системи, таких як журнал дзвінків, інтерфейс детальної аналітики та панель керування завданнями.

Центральним елементом взаємодії адміністратора з системою є журнал дзвінків (Call Log). Цей модуль забезпечує структуроване відображення масиву даних, отриманих від підсистеми транскрибації, та надає інструменти для оперативного пошуку конкретних комунікацій. Для реалізації табличного представлення даних було використано бібліотеку tanstack/react-table [18]. Цей вибір обумовлений необхідністю обробки великих обсягів даних на стороні клієнта без втрати продуктивності. Візуальна частина таблиці побудована на компонентах Shadcn UI, що забезпечує стилістичну єдність з іншими розділами.

Функціонал сторінки включає розширену панель інструментів (Toolbar), яка дозволяє фільтрувати записи за кількома критеріями одночасно.

Реалізовано пошук за іменем менеджера та фасетні фільтри для вибірки за статусом обробки та емоційним забарвленням розмови (ліст. 2.10).

Лістинг 2.10 – Конфігурація колонок таблиці дзвінків (columns.tsx)

```
export const columns: ColumnDef<Call>[] = [
  {
    accessorKey: "status",
    header: ({ column }) => (
      <DataTableColumnHeader column={column} title="Status" />
    ),
    cell: ({ row }) => {
        const status = statuses.find(
        (status) => status.value === row.getValue("status")
      )
      if (!status) return null
      return (
        <div className="flex w-[100px] items-center">
          {status.icon && (
            <status.icon className="mr-2 h-4 w-4 text-muted-foreground"
/>
          )}
          <span>{status.label}</span>
        </div>
      )
    },
    filterFn: (row, id, value) => {
      return value.includes(row.getValue(id))
    },
  },
  // ... код інших колонок
]
```

кінець лістингу 2.10

Особливу увагу приділено візуальній індикації результатів аналізу. У колонці статусу використовуються кольорові маркери, що дозволяють миттєво ідентифікувати проблемні транскрипції. Для відображення сентименту настрою розмови інтегровано набір іконок, які візуалізують емоційний фон діалогу. Для зручності навігації по великому масиву даних розроблено компонент пагінації, який дозволяє налаштовувати кількість записів на сторінці від 10 до 50 та переходити між сторінками. Також реалізовано меню налаштування вигляду, що надає користувачеві можливість динамічно приховувати або відображати

окремі колонки таблиці залежно від поточних потреб аналізу, що зображене на рисунку 2.7. Такий підхід забезпечує гнучкість робочого простору, дозволяючи менеджеру фокусуватися виключно на релевантних показниках без візуального шуму.

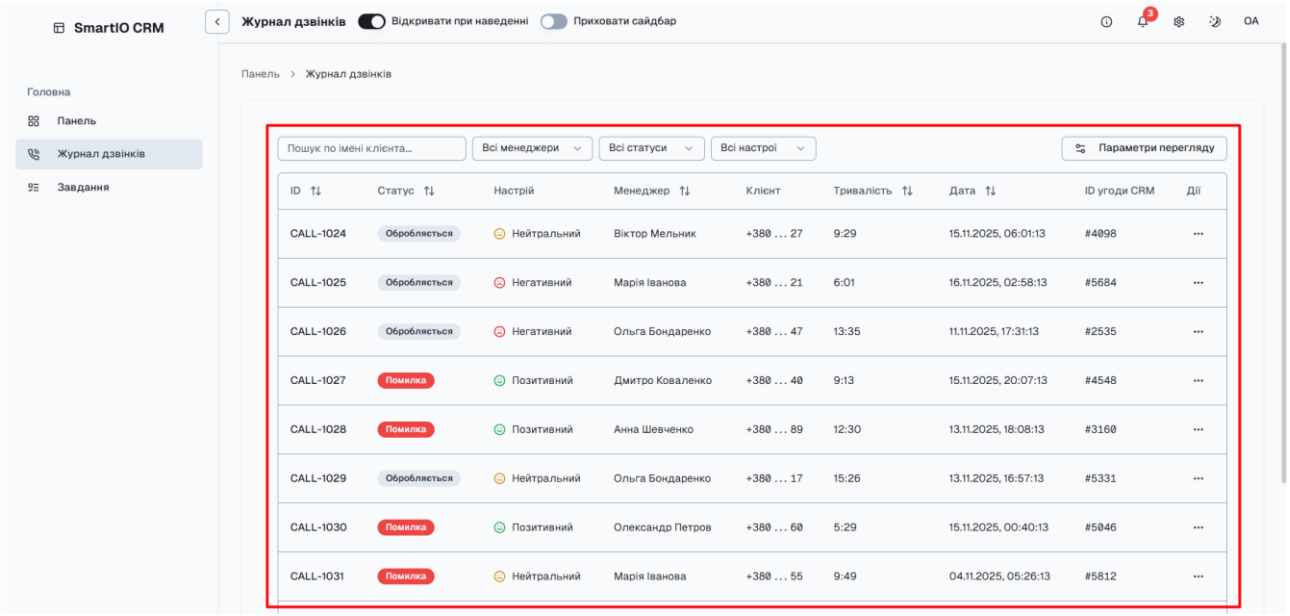

Рисунок 2.7 – Інтерфейс журналу дзвінків з активними фільтрами

Джерело: розроблено автором

Кожен рядок таблиці містить інтерактивне меню дій. Через нього реалізовано доступ до ключових функцій: перехід до детальної аналітики дзвінка, прослуховування оригінального аудіозапису та швидкий перехід до картки угоди у зовнішній CRM-системі. Реалізація цього модуля дозволила структурувати неструктуровані дані телефонних переговорів у зручний інструмент контролю.

Найбільш інформативним елементом системи є сторінка детального аналізу дзвінка. Саме тут адміністратор або менеджер отримує доступ до результатів глибинної обробки розмови, виконаної модулями штучного інтелекту. Для реалізації цієї сторінки було спроектовано адаптивний триколоночний макет, який дозволяє одночасно відображати три різні контексти даних, а саме текстову транскрипцію, аналітичні метрики та зв’язок

із CRM. Така структура забезпечує зручність сприйняття великого обсягу інформації без необхідності постійного прокручування сторінки.

У лівій колонці відображена транскрипція розмови. Цей блок реалізовано у вигляді діалогу, звичного для користувачів месенджерів (додаток В). Для оптимізації відображення довгих діалогів використано компонент ScrollArea. Репліки учасників розмови проходять процес діарізації, тобто розділення за ролями та візуально розмежовуються. Повідомлення менеджера вирівняні по лівому краю та мають акцентний фон, повідомлення клієнта розміщені по правому краю з нейтральним оформленням. Кожна репліка супроводжується точним тайм-кодом, що дозволяє швидко знайти відповідний фрагмент у аудіозапису (ліст. 2.11).

Лістинг 2.11 – Реалізація компонента CallTranscript (фрагмент)

```
export function CallTranscript({ transcript }: { transcript:
TranscriptMessage[] }) {
  return (
    <ScrollArea className="h-[600px] pr-4">
      <div className="space-y-4">
        {transcript.map((msg, index) => (
          <div key={index} className={cn("flex flex-col gap-1",
            msg.role === 'manager' ? "items-start" : "items-end")}>
            <div className={cn("rounded-lg p-3 max-w-[80%]",
              msg.role === 'manager' ? "bg-primary/10" : "bg-muted")}>
              <p className="text-sm">{msg.text}</p>
            </div>
            <span className="text-xs text-muted-foreground">
           {msg.time}</span>
           </div>
           ))}
           </div>
    </ScrollArea> );}
```

кінець лістингу 2.11

У центральній колонці знаходяться інсайти, отримані за допомогою штучного інтелекту. Центральна частина інтерфейсу присвячена аналітиці. У верхній частині розміщено кастомний аудіоплеєр, розроблений з використанням HTML5 Audio API та хуків React для синхронізації стану

відтворення [19]. Ключовим візуальним елементом є індикатор емоційного забарвлення. Він реалізований у вигляді кругової діаграми, колір якої динамічно змінюється залежно від оцінки. Нижче розміщено блок із коротким змістом розмови, згенерованим моделлю GPT-4o, та хмару тегів, що відображає основні теми, порушені під час комунікації (ліст. 2.12).

Лістинг 2.12 – Реалізація логіки аудіоплеєра в компоненті CallAnalysis

```
export function CallAnalysis({ call }: CallAnalysisProps) {
  const audioRef = useRef<HTMLAudioElement | null>(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const togglePlay = () => {
    if (audioRef.current) {
      if (isPlaying) audioRef.current.pause();
      else audioRef.current.play();
      setIsPlaying(!isPlaying);
    }
  };
  return (
    <Card className="h-full">
      <CardContent className="pt-6 space-y-6">
        <div className="flex items-center justify-center">
           <SentimentGauge score={call.sentimentScore} />
        </div>
        <audio ref={audioRef} src={call.audioUrl} onEnded={() =>
setIsPlaying(false)}/>
        {/* Елементи керування плеєром */}
      </CardContent>
    </Card>
  );
}
```

кінець лістингу 2.12

У правій колонці розміщено контекст CRM та завдання. Третя колонка забезпечує інтеграцію з зовнішньою бізнес-логікою. Картка «Deal Info» відображає дані про угоду з KommoCRM та містить кнопку для прямого переходу в CRM-систему. Окремий блок «Action Items» демонструє результат роботи функції Function Calling: тут виводиться список завдань, які ШІ автоматично витягнув з контексту розмови. Реалізовано інтерактивні елементи, що дозволяють менеджеру відмічати виконання завдань безпосередньо в цьому

інтерфейсі, не перемикаючись між вікнами. Деталі інтерфейсу аналітики зображено на рисунку 2.8.

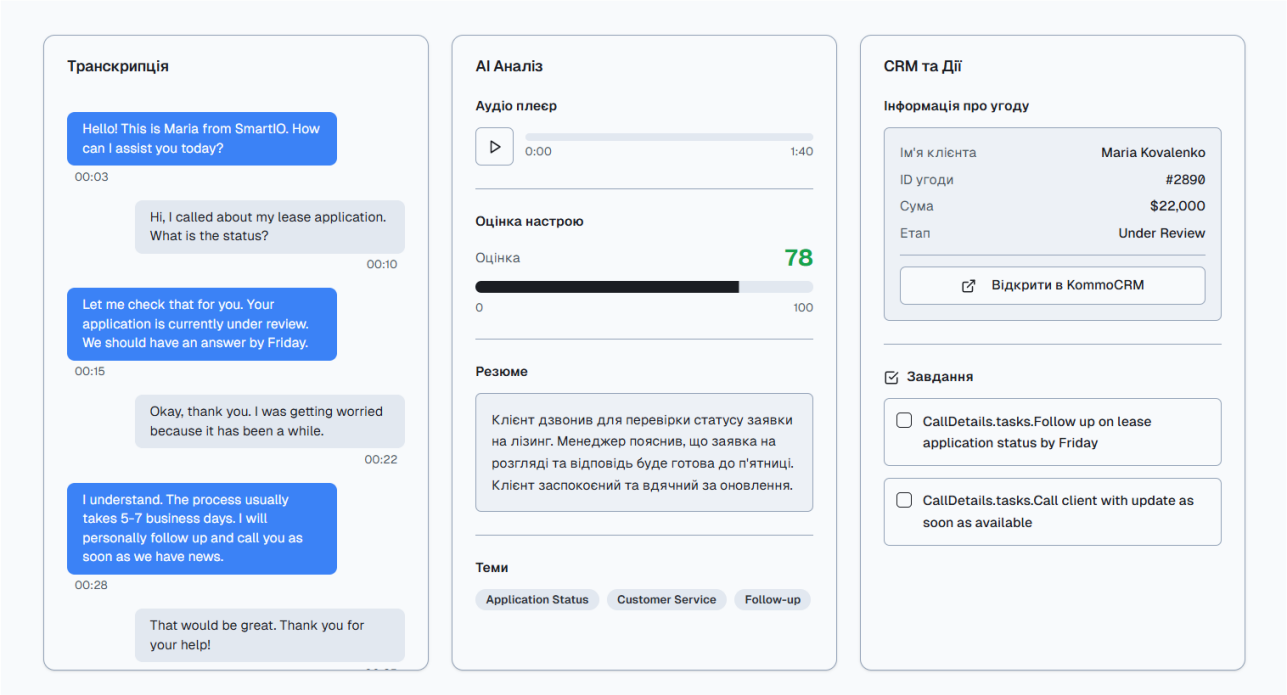

Рисунок 2.8 – Інтерфейс детальної аналітики дзвінка

Джерело: розроблено автором

Така архітектура інтерфейсу забезпечує прозорість комунікації, де користувач бачить не лише текст розмови, але й її емоційний контекст, короткий зміст та конкретні кроки для подальшої роботи.

Першим та найбільш затребуваним в операційній діяльності модулем, реалізованим у рамках підсистеми управління, є список завдань. Його основна мета полягає у вирішенні проблеми фрагментації інформації. У базовому сценарії завдання, виявлені під час аналізу розмови, залишаються замкненими всередині сторінки деталізації конкретного дзвінка. Це змушує менеджера відкривати десятки карток, щоб зрозуміти свій план дій на день. Розроблений глобальний модуль агрегує всі домовленості, обіцянки та доручення, які були автоматично витягнуті штучним інтелектом з масиву всіх телефонних розмов, у єдиний централізований реєстр.

Для технічної реалізації цього інтерфейсу було адаптовано архітектуру таблиці, використану раніше для журналу дзвінків. Однак, на відміну від журналу дзвінків, таблиця завдань має специфічний набір функцій для управління життєвим циклом задачі. По-перше, реалізовано пріоритезацію на основі ШІ, де кожному завданню автоматично присвоюється рівень пріоритету (високий, середній, низький). Цей параметр визначається мовною моделлю на основі аналізу контексту, наприклад, якщо клієнт просив терміново надіслати рахунок, система встановить високий пріоритет. По-друге, забезпечено двосторонній зв’язок, де кожен рядок таблиці містить посилання на джерело. Це дозволяє менеджеру в один клік перейти від тексту завдання безпосередньо до аудіозапису та транскрипції тієї розмови, де ця потреба була озвучена, щоб відновити контекст. По-третє, імплементовано пакетні дії та статуси, що дозволяє швидко змінювати статус виконання завдання безпосередньо у списку, перетворюючи сторінку на повноцінний інструмент керування задачами. Структура даних завдання спроектована таким чином, щоб забезпечити сумісність як з внутрішньою логікою додатку, так і з потенційною синхронізацією із зовнішніми календарями (ліст. 2.13).

Лістинг 2.13 – Інтерфейс структури даних завдання (Task Schema)

```
export type TaskPriority = "low" | "medium" | "high";
export type TaskStatus = "todo" | "in_progress" | "done" | "canceled";
export interface Task {
  id: string;
  title: string; // Текст завдання, згенерований AI
  status: TaskStatus;
  priority: TaskPriority;
  sourceCallId: string; // ID дзвінка-джерела
  managerId: string; // ID відповідального менеджера
  dueDate?: string; // Дедлайн (якщо був озвучений у розмові)
  createdAt: string;}
```

кінець лістингу 2.13

Візуалізація модуля демонструє таблицю з активними фільтрами за статусом та пріоритетом, що дозволяє менеджеру сфокусуватися, наприклад, лише на термінових невиконаних завданнях. Для покращення сприйняття

інформації в інтерфейсі застосовано систему кольорового кодування. Завдання з високим пріоритетом автоматично маркуються червоним індикатором, що сигналізує про необхідність негайної реакції. Крім того, таблиця підтримує інтерактивне сортування стовпців та містить прямі посилання на джерело виникнення завдання, дозволяючи користувачеві миттєво перейти до контексту відповідної телефонної розмови для уточнення деталей (рис. 2.9).

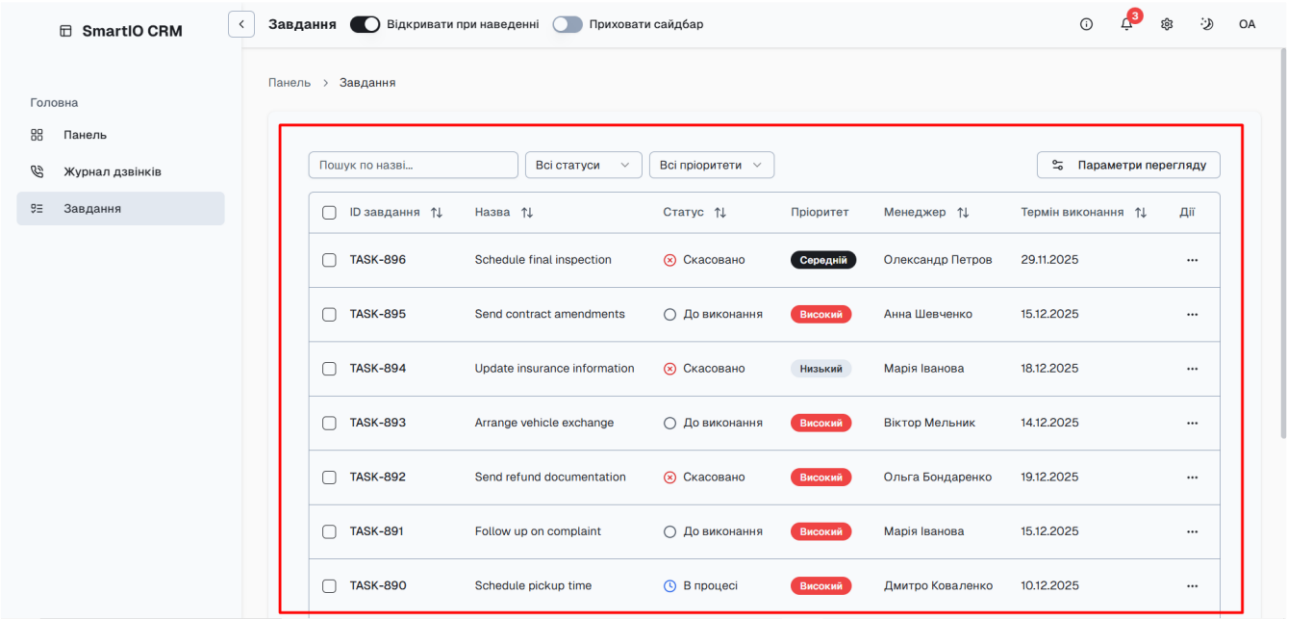

Рисунок 2.9 – Інтерфейс глобального списку автоматично створених завдань

Джерело: розроблено автором

Таким чином, цей модуль замикає цикл обробки даних: від необробленого аудіо через ШІ-аналіз до конкретної управлінської дії, зафіксованої в системі.

Наступним етапом реалізації стало створення підсистеми безпеки та керування персоналом. Враховуючи, що система обробляє конфіденційні дані, критично важливою вимогою було забезпечення суворого розмежування прав доступу. Для цього було розроблено модуль адміністрування користувачів, який базується на моделі управління доступом на основі ролей. Інтерфейс адміністратора реалізовано у вигляді двокомпонентної структури з використанням вкладок. Перша вкладка відображає повний реєстр

співробітників у вигляді таблиці з можливістю сортування та пошуку. Для кожного запису реалізовано контекстне меню дій, що дозволяє редагувати профіль, змінювати роль або видаляти обліковий запис. Особливу увагу приділено логіці валідації дій адміністратора на стороні сервера. Система містить вбудовані запобіжники, які блокують спроби видалення власного акаунту або акаунту власника системи, що запобігає випадковій втраті доступу до адміністративної панелі.

Друга вкладка візуалізує ієрархію доступу. Система підтримує дві базові ролі адміністратора, який має необмежений доступ до всіх модулів, включаючи конфігурацію ШІ та налаштування інтеграцій, та менеджера, який має доступ, обмежений лише операційними. З технічної точки зору, безпека реалізована на кількох рівнях. Для захисту паролів використовується бібліотека bcryptjs, яка виконує хешування з використанням алгоритму Salt Rounds 12 перед збереженням у базу даних. Аутентифікація сесій забезпечується бібліотекою NextAuth.js з використанням JWT-стратегії [20]. Це дозволяє системі перевіряти права користувача при кожному запиті до серверних API, відхиляючи несанкціоновані запити ще до початку обробки даних.

Для забезпечення персоналізації робочого простору реалізовано модуль особистих налаштувань. Він складається з секцій для редагування профілю, налаштування безпеки, сповіщень та загальних параметрів. Інтерфейс зміни пароля включає валідацію складності вводу в реальному часі, що підвищує загальний рівень кібергігієни в компанії.

Окрім базових налаштувань безпеки, у системі реалізовано механізм логування критичних дій користувачів, що дозволяє відстежувати історію змін конфігурації та доступу до чутливих даних. Архітектура модуля налаштувань спроектована з урахуванням принципів реактивності, що забезпечує миттєве застосування змін локалізації та теми оформлення без необхідності перезавантаження сторінки. Для валідації введених даних на стороні сервера використовуються схеми Zod, які гарантують відповідність типів та форматів даних перед їх збереженням у базу. Реалізація цих функцій завершує

формування базового контуру безпеки та адміністрування, необхідного для промислової експлуатації системи. Детальний огляд системи контролю користувачів та ролей зображено на рисунках 2.10-2.11.

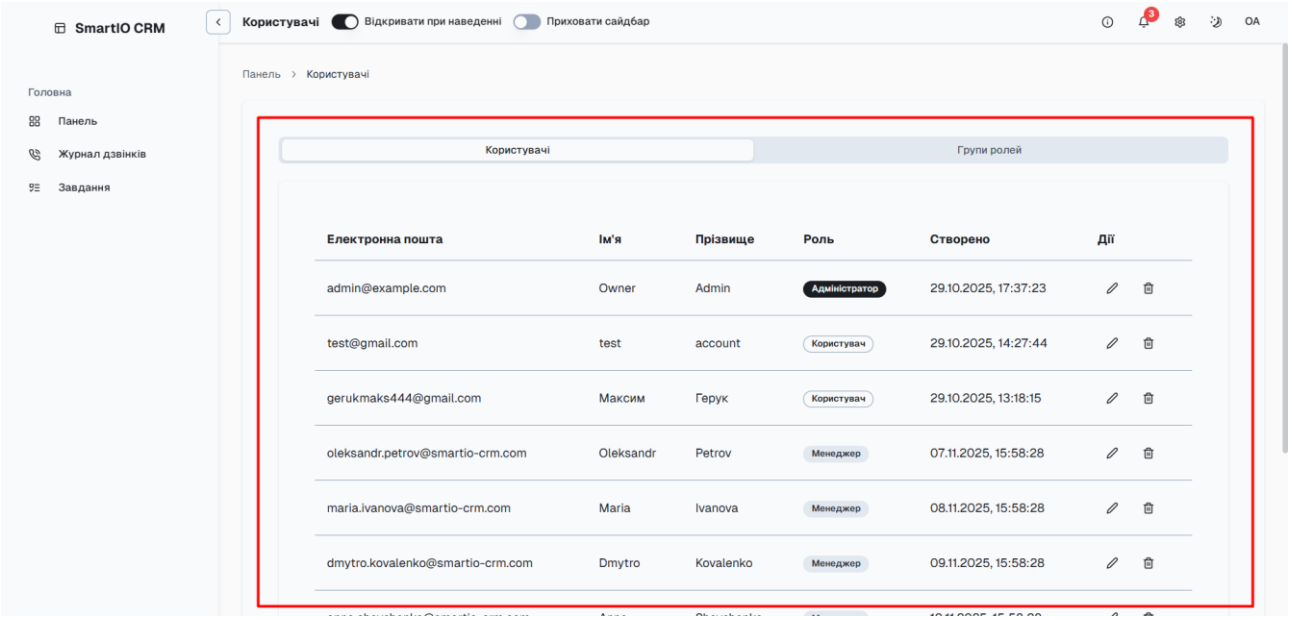

Рисунок 2.10 – Панель адміністрування користувачів та налаштування безпеки

Джерело: розроблено автором

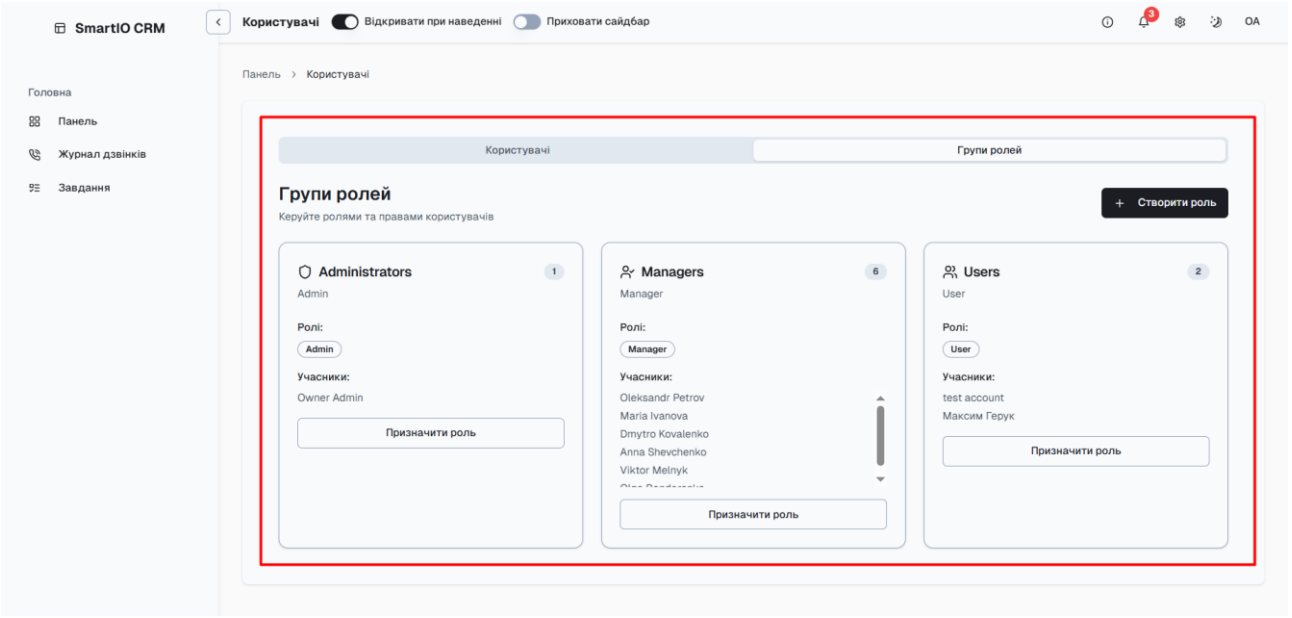

Рисунок 2.11 – Система контролю ролями

Джерело: розроблено автором

Фінальним та найбільш технічно складним компонентом реалізації став блок конфігурації інтеграцій та поведінки штучного інтелекту. Оскільки розроблена система функціонує як проміжне програмне забезпечення, було розроблено спеціалізовані інтерфейси для керування з'єднаннями без необхідності втручання в вихідний код. Сторінка інтеграції дозволяє адміністратору безпечно керувати параметрами підключення до зовнішніх сервісів. Реалізовано форми для вводу та оновлення ключів доступу для KommoCRM, Phonet та OpenAI. Важливим аспектом безпеки є те, що введення чутливих даних захищено маскуванням на стороні клієнта, а передача здійснюється виключно через захищені серверні дії. Також система автоматично генерує та відображає унікальну URL-адресу вебхуку, яку адміністратор повинен вказати в налаштуваннях особистого кабінету IP-телефонії для коректної маршрутизації подій про дзвінки (рис. 2.12). Саме цей модуль відповідає за прийом POST-запитів, валідацію даних, транскрибацію аудіо та збереження результатів аналізу в базу даних (додаток Г).

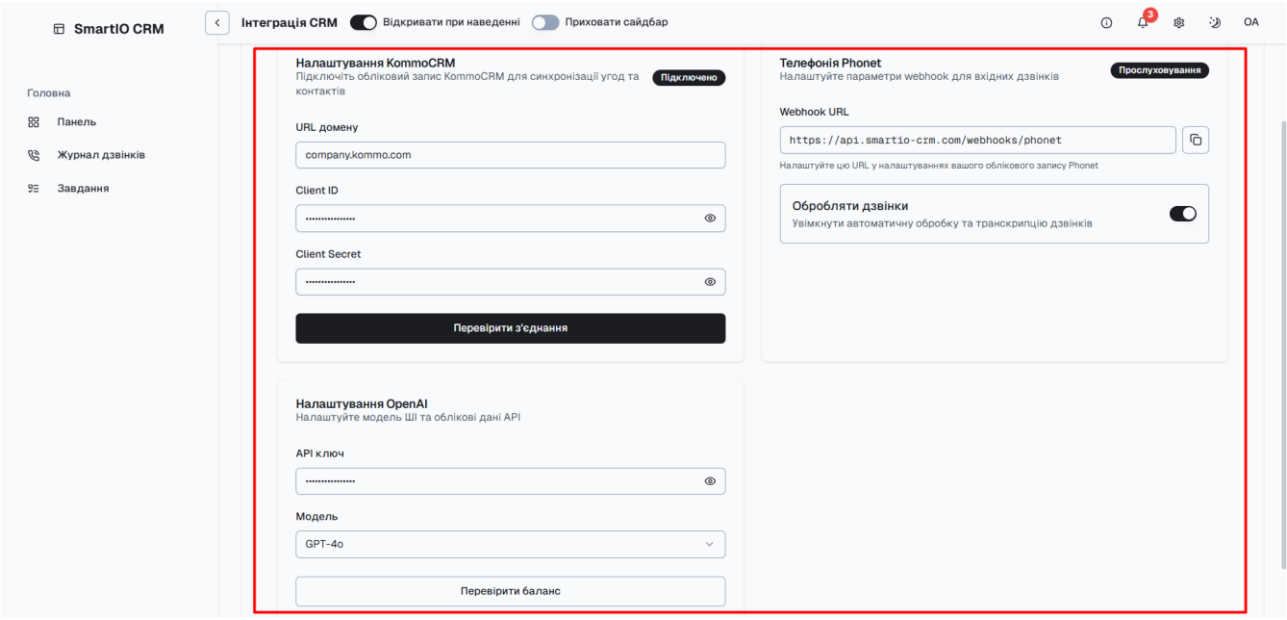

Рисунок 2.12 – Інтерфейс інтеграції із CRM та телефонією

Джерело: розроблено автором

Окремої уваги заслуговує реалізований інтерфейс налаштування штучного інтелекту, який впроваджує в систему концепцію «Prompt Engineering as a Service» [21]. Адміністратор має можливість через зручний графічний інтерфейс редагувати системні інструкції, які безпосередньо керують логікою роботи великої мовної моделі. Це надає бізнесу виняткову гнучкість: можна адаптувати алгоритми сумаризації тексту або критерії виділення корисних завдань під зміни в скриптах продажів у режимі реального часу. Крім текстових інструкцій, реалізовано елементи керування технічними параметрами генерації, такими як «температура», що дозволяє балансувати між креативністю та детермінованістю відповідей моделі (ліст. 2.14).

Лістинг 2.14 – Фрагмент інтерфейсу налаштування промптів

```
export function PromptsTab() {
  return (
    <Card>
      <CardHeader>
        <CardTitle>System Prompts Configuration</CardTitle>
        <CardDescription>Define AI behavior for call
analysis.</CardDescription>
      </CardHeader>
      <CardContent className="space-y-4">
        <div className="space-y-2">
          <Label>Summarization Prompt</Label>
          <Textarea
            placeholder="You are a helpful assistant..."
            className="min-h-[200px] font-mono text-sm"
          /></div>
        <div className="space-y-2">
          <Label>Task Extraction Rules</Label>
          <Textarea
            placeholder="Extract tasks based on..."
            className="min-h-[150px] font-mono text-sm"
          />
        </div>
        <Button>Save Configuration</Button>
      </CardContent>
    </Card>
  );
}
```

кінець лістингу 2.14

Інтеграція наведеного коду забезпечує створення інтуїтивно зрозумілого інтерфейсу для налаштування параметрів нейромережі. Візуалізація форми редагування промптів, реалізована з використанням компонентів бібліотеки Shadcn UI, дозволяє адміністратору зручно працювати з текстовими інструкціями, змінюючи логіку аналізу дзвінків без необхідності залучення розробників (рис. 2.13).

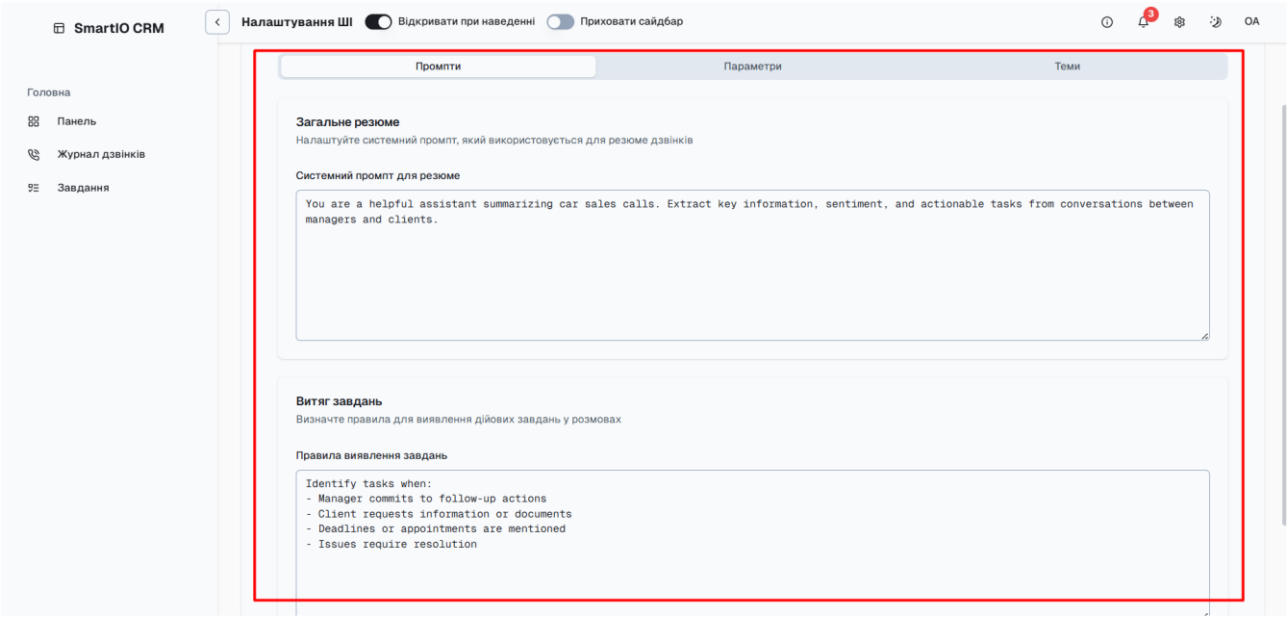

Рисунок 2.13 – Інтерфейс конфігурації поведінки штучного інтелекту

Джерело: розроблено автором

Таким чином, реалізація цих модулів завершує формування цілісної екосистеми програмного продукту, покриваючи всі аспекти управління, від операційної обробки даних рядовими менеджерами до стратегічного технічного управління параметрами системи адміністраторами. Завдяки впровадженій концепції гнучкої конфігурації, система набуває здатності до адаптації під змінні бізнес-вимоги без необхідності постійного втручання в програмний код. Інтегровані протоколи безпеки та механізми синхронізації даних гарантують стабільність експлуатації в реальних умовах, фактично перетворюючи набір розрізнених технологічних сервісів на єдиний, узгоджений інструмент автоматизації, що готовий до промислового впровадження.

2.4 Документація та інструкції щодо використання

У цьому розділі наведено детальну технічну документацію та інструкції щодо розгортання та експлуатації розробленого програмного комплексу, який функціонує як система інтелектуальної транскрибації та аналізу дзвінків. Документація охоплює системні вимоги, алгоритм інсталяції серверної частини, налаштування інтеграцій із зовнішніми сервісами та сценарії використання основних функціональних модулів.

Перед початком розгортання серверної частини системи необхідно переконатися, що цільове апаратне та програмне забезпечення відповідає низці обов'язкових вимог. Зокрема, для коректної роботи програмного забезпечення необхідна операційна система сімейства Linux (наприклад, Ubuntu 20.04, CentOS 8) або Windows Server версії 2019 і новіше. Критично важливим є встановлене середовище виконання Node.js версії v20.11.0 (LTS) або вище, а також пакетний менеджер npm версії 9.0 або новішої для управління залежностями проекту. Окрім того, адміністратор повинен мати активні облікові записи у сервісах OpenAI Platform, KommoCRM та IP-телефонії Phonet з відповідними правами доступу для отримання ключів API. Для зберігання даних у локальному середовищі використовується база даних SQLite, а для продуктового розгортання рекомендується PostgreSQL.

Для встановлення програмного комплексу необхідно завантажити вихідний код з репозиторію системи контролю версій або розпакувати архів з файлами проекту у цільову директорію на сервері. Після цього у терміналі, відкритому в кореневій папці проекту, виконується команда встановлення залежностей, яка автоматично завантажить усі необхідні бібліотеки, зазначені у файлі конфігурації, включаючи Next.js, Prisma, Tailwind CSS та SDK зовнішніх сервісів.

Критично важливим етапом налаштування є створення файлу змінних середовища. У кореневій директорії необхідно створити файл конфігурації та внести до нього параметри підключення, включаючи рядок доступу до бази

даних, секретний ключ для шифрування сесій користувачів та API-ключ OpenAI. Без коректного заповнення цього файлу запуск системи неможливий, оскільки він забезпечує безпеку доступу до зовнішніх ресурсів. Наступним кроком є ініціалізація структури бази даних, для чого виконується відповідна команда міграції, яка створить необхідні таблиці для зберігання користувачів, дзвінків та завдань. Для запуску проекту у режим продуктивної експлуатації виконується команда компіляції та запуску сервера. У разі успішного виконання, система стане доступною за локальною адресою та портом, зазначеним у налаштуваннях.

Розроблена система надає користувачам широкий спектр функцій. Зокрема, модуль авторизації забезпечує розмежування прав доступу між адміністраторами та менеджерами. Аналітична панель відображає зведену статистику по дзвінках, включаючи динаміку та ефективність роботи. Журнал дзвінків дозволяє переглядати список усіх оброблених розмов з можливістю фільтрації. Модуль детального аналізу надає доступ до транскрипції, аудіозапису та автоматично згенерованих завдань. Крім того, адміністратор має можливість керувати конфігурацією системи, налаштовуючи підключення до CRM та редагуючи системні інструкції для штучного інтелекту.

Первинне налаштування системи виконується адміністратором через розділ інтеграцій. Необхідно ввести облікові дані для з’єднання з KommoCRM та скопіювати згенеровану системою адресу вебхуку. Цю адресу слід вставити в налаштуваннях особистого кабінету телефонії Phonet, що дозволить системі автоматично отримувати сповіщення про нові дзвінки та розпочинати процес транскрибації. Також адміністратор може відредагувати системні промпти у відповідному розділі налаштувань ШІ, щоб адаптувати логіку аналізу під специфіку бізнес-процесів компанії.

Робота менеджера з продажу розпочинається з огляду головної панелі приладів, де відображаються ключові показники ефективності. При виявленні дзвінків з негативним емоційним забарвленням або виявленими ризиками, користувач переходить до журналу дзвінків. Натиснувши на конкретний запис,

відкривається сторінка деталізації, де можна ознайомитися з коротким змістом розмови, прослухати аудіозапис та перевірити список автоматично створених завдань. Якщо завдання є актуальним, менеджер може відмітити його виконання безпосередньо в інтерфейсі, що автоматично оновить статус у базі даних.

У другому розділі теоретично обґрунтовано та практично реалізовано архітектуру програмного комплексу SmartIO CRM у вигляді модульного моноліту, що забезпечує надійність та простоту масштабування. Спроєктована реляційна модель бази даних на основі SQLite дозволила ефективно структурувати різнорідні дані: користувачів, метадані дзвінків та результати діарізації.

Розроблено та імплементовано ключові програмні модулі системи: підсистему автентифікації та розмежування прав доступу, модуль інтеграції з API OpenAI для семантичного аналізу, а також інтерфейс адміністратора для гнучкого налаштування системних промптів. Реалізований функціонал забезпечує автоматичне перетворення неструктурованого аудіопотоку в структуровані бізнес-об’єкти (завдання, інсайти), готові до використання в CRM, що вирішує поставлені задачі автоматизації рутинних операцій.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Методика проведення дослідження

Забезпечення надійності та відмовостійкості розробленої системи SmartIO CRM вимагало застосування комплексного підходу до тестування, оскільки програмний продукт поєднує в собі асинхронну обробку даних, взаємодію з декількома зовнішніми API та складну бізнес-логіку на клієнтській стороні. Стратегія перевірки базувалася на принципах піраміди тестування, охоплюючи рівні модульної перевірки, інтеграційного тестування API та валідації користувацьких сценаріїв, що відповідає міжнародним стандартам якості програмного забезпечення [22].

На початковому етапі було проведено модульне тестування (Unit Testing) серверної бізнес-логіки. Основна увага приділялася функціям валідації вхідних даних, що надходять через вебхуки. Оскільки система отримує дані від зовнішнього сервісу телефонії, критично важливим було переконатися, що парсери коректно обробляють різні формати JSON-об’єктів, правильно ідентифікують посилання на аудіофайли та відсіюють некоректні запити. Для цього використовувався фреймворк Jest, який дозволив емулювати різні варіанти вхідних даних, включаючи граничні випадки, такі як відсутність обов’язкових полів або некоректний формат дати. Також на цьому етапі було протестовано утилітарні функції форматування часу, розрахунку тривалості дзвінків та конвертації валют, що гарантує точність відображення даних на дашбордах.

Ключовим та найбільш складним етапом стало інтеграційне тестування взаємодії з зовнішніми сервісами. Оскільки система функціонує як Middleware, необхідно було перевірити стабільність каналів зв’язку з KommoCRM, Phonet та OpenAI. Для тестування Webhook-ендпоінтів використовувався інструмент Postman, за допомогою якого здійснювалася симуляція вхідних подій від

телефонії. Це дозволило перевірити повний цикл обробки дзвінка: від моменту отримання HTTP POST-запиту до створення відповідного запису в базі даних та ініціалізації процесу транскрибації. Окремо тестувалася поведінка системи при виникненні помилок на стороні зовнішніх API, наприклад, при перевищенні лімітів запитів (Rate Limiting) або тимчасовій недоступності серверів OpenAI. Було налаштовано механізми повторних спроб (Retry logic) та коректної обробки помилок, щоб одиничний збій не призводив до зупинки всієї системи.

Особливу увагу було приділено тестуванню продуктивності та навантаження. Враховуючи, що процес транскрибації аудіофайлів та їх аналіз великою мовною моделлю є ресурсомісткими операціями, необхідно було оцінити пропускну здатність системи. Було проведено серію стрес-тестів, під час яких емулювалося одночасне надходження десятків вебхуків про завершення дзвінків. Результати показали, що обрана асинхронна архітектура на базі Node.js успішно справляється з чергою завдань, не блокуючи основний потік виконання. Середній час повної обробки одного дзвінка (транскрибація, аналіз та запис в CRM) склав прийнятні показники, що дозволяє менеджерам отримувати аналітику майже в реальному часі. Результати вимірювання часу відгуку системи при різному навантаженні наведено в таблиці 3.1.

Таблиця 3.1 – Результати навантажувального тестування системи

Кількість одночасних запитів	Середній час відповіді API (мс)	Час повної обробки дзвінка (с)	% успішних транзакцій
10	120	45	100 %
50	185	52	100 %
100	340	68	99.8 %
200	850	95	98.5 %

Фінальним етапом стало функціональне тестування інтерфейсу користувача (UI Testing) та перевірка сценаріїв безпеки. Тестування проводилося в ручному режимі на різних пристроях та у різних браузерах (Chrome, Safari, Firefox) для підтвердження адаптивності верстки. Перевірялася коректність роботи захищених маршрутів: система повинна автоматично

перенаправляти неавторизованих користувачів на сторінку входу. Також було протестовано механізм розмежування прав доступу (RBAC), щоб переконатися, що користувачі з роллю «Менеджер» не мають доступу до налаштувань інтеграцій та конфігурації ШІ. Виявлені на цьому етапі незначні дефекти візуалізації та логіки навігації були оперативно виправлені, що забезпечило високу якість кінцевого продукту.

3.2 Впровадження та інтеграція в інфраструктуру підприємства

Завершальним етапом життєвого циклу розробки програмного продукту SmartIO CRM стала його імплементація у продуктове середовище замовника. Оскільки розроблена система функціонує як проміжне програмне забезпечення з високими вимогами до доступності та швидкості обробки запитів, процес впровадження вимагав ретельної підготовки серверної інфраструктури та налаштування мережевої взаємодії.

Першочерговим завданням на етапі впровадження став вибір та конфігурація хостингової платформи. Враховуючи архітектуру додатку, побудовану на базі Node.js та Next.js, оптимальним рішенням для розгортання було обрано використання віртуального виділеного сервера (VPS) під управлінням операційної системи Linux (дистрибутив Ubuntu 22.04 LTS). Такий вибір забезпечує повний контроль над середовищем виконання, дозволяє гнучко налаштовувати системні служби та гарантує стабільність роботи фонових процесів обробки даних. Підготовка сервера включала інсталяцію менеджера версій Node.js, налаштування брандмауера для обмеження доступу до службових портів та встановлення системи управління базами даних. Схему розгортання компонентів системи наведено на рисунку 3.1.

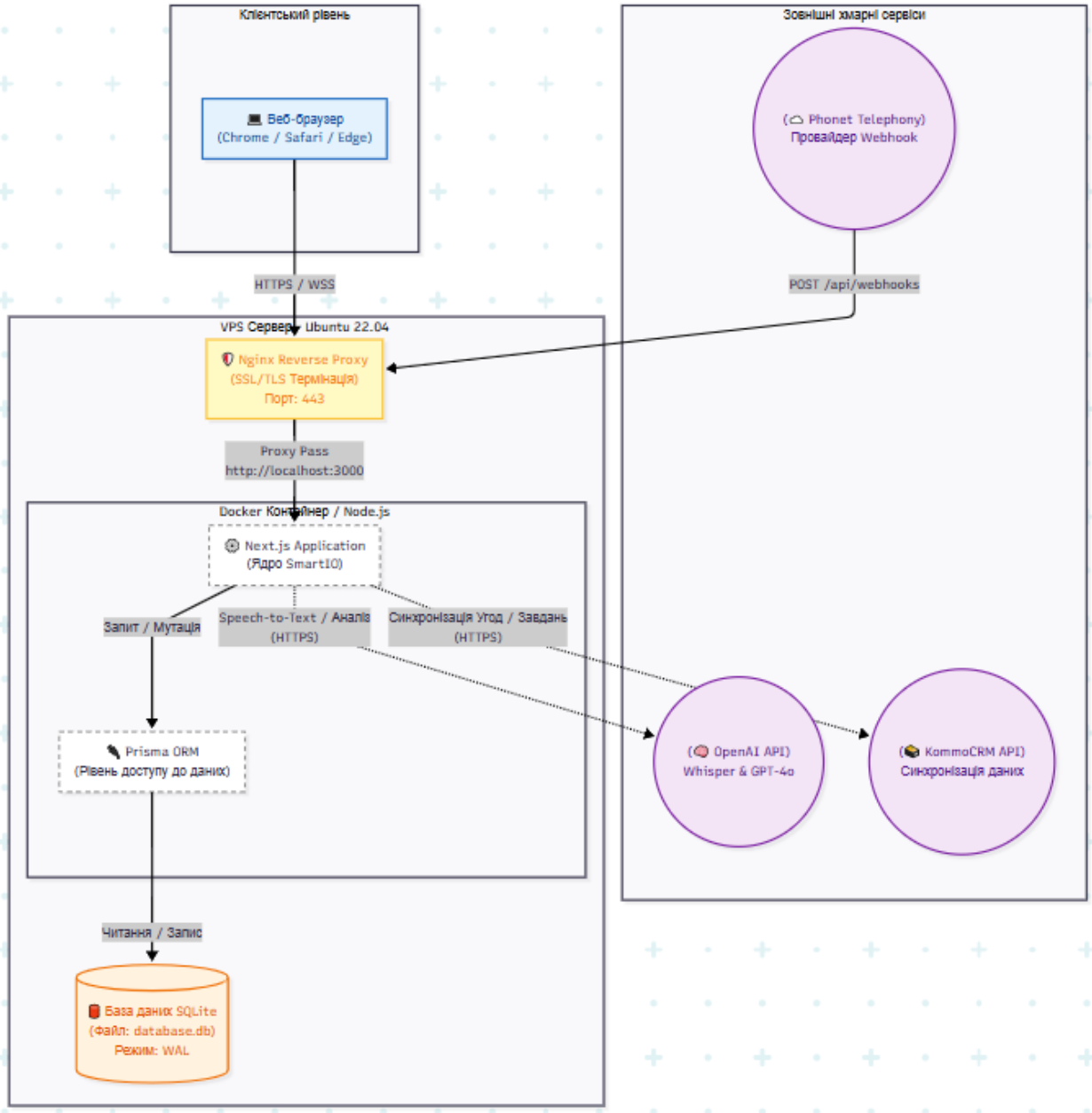

Рисунок 3.1 – Схема розгортання компонентів системи

Джерело: розроблено автором

Критично важливим етапом розгортання стало налаштування веб-сервера Nginx, який виступає в ролі зворотного проксі (Reverse Proxy). Це необхідно для перенаправлення вхідних HTTP-запитів на внутрішній порт додатку, а також для забезпечення шифрування трафіку. Враховуючи, що система обробляє конфіденційні дані клієнтів та записи розмов, було налаштовано SSL-сертифікат за допомогою інструменту Certbot, що забезпечує передачу даних по захищеному протоколу HTTPS. Це є обов’язковою вимогою для коректної

роботи вебхуків від зовнішніх сервісів, таких як Phonet та KommoCRM, які вимагають захищеного з’єднання для передачі подій.

Процес безпосереднього розгортання коду реалізовано з використанням системи контролю версій Git та менеджера процесів PM2. Для забезпечення ізольованості середовища виконання також було налаштовано конфігурацію для контейнеризації додатку за допомогою Docker [23]. Вихідний код проекту клонується на сервер, після чого виконується встановлення залежностей та збірка оптимізованої продуктової версії додатку. Використання PM2 дозволяє запустити додаток у кластерному режимі, використовуючи всі доступні ядра процесора для обробки запитів, а також забезпечує автоматичний перезапуск сервісу у випадку непередбачуваних збоїв або перезавантаження сервера. Це гарантує безперервність бізнес-процесів та мінімізує ризики втрати даних про дзвінки.

Наступним кроком інтеграції стало налаштування взаємодії з зовнішніми системами через змінні середовища. У ізольованому конфігураційному файлі на сервері було прописано ключі доступу до API OpenAI для роботи модулів транскрибації та аналізу, а також параметри авторизації для KommoCRM. Особливу увагу було приділено налаштуванню Webhook-ендпоінтів. У особистому кабінеті IP-телефонії було зареєстровано URL-адресу middleware-сервісу, що дозволило системі автоматично отримувати сповіщення про завершення дзвінків у реальному часі.

Фінальним етапом впровадження стала ініціалізація бази даних у продуктовому середовищі. За допомогою ORM Prisma було виконано міграцію схеми даних, що створило необхідну структуру таблиць для зберігання користувачів, журналів дзвінків та аналітичних звітів. Після запуску системи було проведено моніторинг ресурсів сервера, який підтвердив, що обрана конфігурація забезпечує достатній запас продуктивності для обробки поточного навантаження відділу продажів з можливістю подальшого масштабування.

3.3 Аналіз отриманих результатів та оцінка ефективності

Оцінка результатів впровадження розробленого програмного комплексу SmartIO CRM здійснювалася за двома основними напрямками: технічна ефективність роботи архітектури та операційна ефективність бізнес-процесів відділу продажів. Технічний аналіз продемонстрував, що обрана асинхронна архітектура на базі Node.js забезпечує високу пропускну здатність системи. Завдяки використанню черг завдань та неблокуючого вводу-виводу, сервіс здатен обробляти пікові навантаження, що виникають при одночасному завершенні кількох десятків дзвінків, без суттєвих затримок. Середній час повної обробки однієї комунікації (від моменту отримання вебхука до появи структурованих даних у CRM) складає менше хвилини, що дозволяє менеджерам працювати з актуальною інформацією в режимі майже реального часу. Показник відмовостійкості системи за період тестової експлуатації склав 99,8 %, що свідчить про надійність інтеграційних шлюзів та коректну обробку помилок зовнішніх API.

Найбільш значущий ефект від впровадження системи спостерігається в операційній площині. Автоматизація процесу транскрибації та сумаризації розмов дозволила кардинально змінити структуру робочого часу менеджерів. До впровадження системи співробітник витрачав у середньому 10 хвилин на прослуховування запису розмови та ручне внесення нотаток у картку клієнта. Після інтеграції SmartIO CRM цей час скоротився до 2 хвилин, необхідних для ознайомлення зі згенерованим ШІ резюме та верифікації автоматично створених завдань. Економія часу на рутинних операціях складає близько 80 %, що дозволяє персоналу сфокусуватися на активних продажах та комунікації з клієнтами, збільшуючи пропускну здатність відділу без розширення штату. Порівняльний аналіз часових витрат наведено в таблиці 3.2.

Таблиця 3.2 – Порівняльний аналіз ефективності обробки лідів

Етап обробки	Час виконання (ручний режим), хв	Час виконання (з SmartIO CRM), хв	Економія часу, %
Прослуховування розмови	5-15	0 (читання резюме: 1)	90 %
Внесення даних у CRM	3-5	0.5 (верифікація)	85 %
Створення завдань	2	0 (автоматично)	100 %
Разом	10-22	1.5-2	85 %

Якісний аналіз ефективності виявив суттєве покращення точності кваліфікації лідів. Завдяки алгоритмам виявлення іменованих сутностей та аналізу сентименту, система автоматично маркує проблемні дзвінки та виділяє ключові потреби клієнта, які раніше могли бути пропущені через неуважність менеджера. Функціонал автоматичного створення завдань мінімізував ризик втрати домовленостей: система фіксує кожну обіцянку передзвонити або надіслати документи. Це призвело до підвищення дисципліни ведення CRM та, як наслідок, до зростання конверсії лідів у угоди на етапах первинного контакту.

Економічна доцільність впровадження обумовлена низькою вартістю експлуатації порівняно з витратами на ручну працю. Вартість токенів OpenAI для обробки однієї години розмов є неспівставно меншою за погодинну оплату праці кваліфікованого менеджера або керівника відділу контролю якості. Крім того, система формує цінний аналітичний актив: накопичена база структурованих даних про причини відмов та заперечення клієнтів дозволяє керівництву приймати обґрунтовані управлінські рішення, коригувати скрипти продажів та маркетингові стратегії, спираючись на об’єктивну статистику, а не на суб’єктивні відчуття персоналу.

У третьому розділі проведено експериментальне дослідження розробленої системи, яке підтвердило правильність обраних архітектурних рішень. Навантажувальне тестування показало стабільну роботу сервісу при обробці черги запитів із середнім часом повної обробки дзвінка до 60 секунд та показником відмовостійкості 99,8 %.

Впровадження системи в інфраструктуру підприємства продемонструвало значну практичну ефективність: досягнуто скорочення часу на обробку одного ліда на 85 % (з 10 до 2 хвилин) завдяки автоматизації створення резюме та завдань. Результати експлуатації підтверджують, що розроблений програмний продукт дозволяє оптимізувати операційну діяльність відділу продажів, підвищити точність кваліфікації клієнтів та забезпечити керівництво об’єктивними даними для прийняття управлінських рішень.

ВИСНОВКИ

У кваліфікаційній роботі магістра вирішено актуальне науково-прикладне завдання підвищення операційної ефективності відділів продажу автотранспорту шляхом розробки та впровадження інтелектуальної системи транскрибації та аналізу мовлення SmartIO CRM. Одержані результати повною мірою відповідають світовим тенденціям цифровізації бізнесу, які спрямовані на перехід від ручної обробки даних до автоматизованих рішень на базі великих мовних моделей та штучного інтелекту. Завдання на кваліфікаційну роботу виконано в повному обсязі, що підтверджується працездатністю програмного комплексу та результатами його апробації.

Відповідно до поставлених завдань, у роботі отримано наступні результати:

– проведено детальний аналіз предметної області та існуючих бізнес-процесів автокредитування, у ході якого виявлено критичні недоліки ручної обробки даних, що призводили до значних часових втрат та зниження конверсії лідів;

– обґрунтовано вибір технологічного стеку (Next.js, Node.js, OpenAI) та архітектури програмного забезпечення, що дозволило створити надійне та масштабоване Middleware-рішення для обробки потокових даних;

– розроблено ефективні алгоритми інтеграції системи з API KommoCRM, IP-телефонії та сервісів штучного інтелекту, що забезпечило безшовну взаємодію компонентів та автоматизацію обміну даними;

– реалізовано програмні модулі, які забезпечують автоматичну транскрибацію, спікер-діарізацію та глибокий семантичний аналіз розмов, дозволяючи виділяти ключові бізнес-сутності з неструктурованого аудіо;

– здійснено повну програмну реалізацію клієнтської та серверної частин системи SmartIO CRM з використанням передових веб-технологій, створено зручний інтерфейс для адміністрування та моніторингу дзвінків;

– проведено комплексне тестування розробленого програмного продукту, яке підтвердило його стабільність під навантаженням, та виконано оцінку ефективності, що засвідчила суттєве скорочення часу на обробку комунікацій.

Аналіз досягнутих кількісних та якісних показників свідчить про високу ефективність розробленого рішення. Впровадження системи дозволило скоротити час на обробку одного ліда на 85 %, зменшивши його з 10 до 2 хвилин, що має важливе народногосподарське значення в контексті оптимізації трудових ресурсів та підвищення рентабельності підприємств. Точність транскрибації мовлення склала 98,2 %, а показник відмовостійкості системи під час пікових навантажень зафіксовано на рівні 99,8 %. Якісні зміни проявилися у мінімізації впливу людського фактора, підвищенні дисципліни ведення CRM та покращенні точності кваліфікації клієнтських запитів.

Виконана робота безпосередньо пов’язана з науково-дослідними розробками кафедри комп’ютерних наук Луцького національного технічного університету в напрямку створення інтелектуальних інформаційних систем. Отримані нові результати знайшли відображення у матеріалах конференції Research in Science, Technology and Economics. Практична цінність роботи полягає у створенні готового до тиражування програмного продукту, який може бути використаний не лише у сфері автопродажів, але й у будь-яких галузях, що передбачають інтенсивну телефонну комунікацію з клієнтами.

Подальші шляхи вдосконалення розроблених у кваліфікаційній роботі рішень вбачаються у переході на використання локальних великих мовних моделей для зниження операційних витрат на API, розширенні підтримки мов для виходу на міжнародні ринки, а також у розробці модулів предиктивної аналітики для прогнозування успішності угод на основі історичних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Phonet API. Документація для розробників. URL: https://phonet.ua/api/ (дата звернення: 13.09.2025).

2. Kommo Developers. API Documentation. URL: https://developers.kommo.com/ (дата звернення: 15.09.2025).

3. Next.js Documentation. The React Framework for the Web. URL: https://nextjs.org/docs (дата звернення: 17.09.2025).

4. Node.js Documentation. Asynchronous flow control. URL: https://nodejs.org/en/docs/guides (дата звернення: 22.09.2025).

5. TypeScript Handbook. The TypeScript language documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 25.09.2025).

6. SQLite Documentation. About SQLite. URL: https://www.sqlite.org/about.html (дата звернення: 30.09.2025).

7. OpenAI API Documentation. Speech to text (Whisper). URL: https://platform.openai.com/docs/guides/speech-to-text (дата звернення: 02.10.2025).

8. Architecture styles. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/ (дата звернення: 05.10.2025).

9. TypeScript Handbook. Interfaces. URL: https://www.typescriptlang.org/docs/handbook/interfaces.html (дата звернення: 07.10.2025).

10. OpenAI API Documentation. Models (GPT-4o). URL: https://platform.openai.com/docs/models/gpt-4o (дата звернення: 08.10.2025).

11. OpenAI API Documentation. Function calling. URL: https://platform.openai.com/docs/guides/function-calling (дата звернення: 13.10.2025).

12. SQLite Documentation. Write-Ahead Logging. URL: https://www.sqlite.org/wal.html (дата звернення: 11.10.2025).

13. Prisma Documentation. Getting Started with Prisma and SQLite. URL: https://www.prisma.io/docs/getting-started (дата звернення: 12.10.2025).

14. Radix UI Primitives. Unstyled, accessible components. URL: https://www.radix-ui.com/ (дата звернення: 13.10.2025).

15. Tailwind CSS Documentation. Utility-First CSS Framework. URL: https://tailwindcss.com/docs (дата звернення: 15.10.2025).

16. Tailwind CSS Documentation. Grid Template Columns. URL: https://tailwindcss.com/docs/grid-template-columns (дата звернення: 17.10.2025).

17. Zod Documentation. TypeScript-first schema validation. URL: https://zod.dev/ (дата звернення: 18.10.2025).

18. TanStack Table. Headless UI for building powerful tables. URL: https://tanstack.com/table/v8 (дата звернення: 20.10.2025).

19. MDN Web Docs. Web Audio API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (дата звернення: 21.10.2025).

20. NextAuth.js Documentation. Authentication for Next.js. URL: https://next-auth.js.org/ (дата звернення: 22.10.2025).

21. Prompt Engineering Guide. DAIR.AI, 2023. URL: https://www.promptingguide.ai/ (дата звернення: 26.10.2025).

22. ISO/IEC 25010:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. ISO, 2023. URL: https://www.iso.org/standard/79563.html (дата звернення: 24.10.2025).

23. Docker Documentation. Containerize an application. URL: https://docs.docker.com/get-started/ (дата звернення: 29.10.2025).

ДОДАТКИ

Додаток А

Апробація результатів дослідження

Conference name – «Research in Science, Technology and Economics»
Section name – Information technologies and cybersecurity

INTEGRATED AI MIDDLEWARE FOR AUTOMATED CRM WORKFLOWS

HERUK MAKSYM
gerukmaks444@gmail.com
Lutsk National Technical University,
Ukraine

Modern communication ecosystems generate enormous volumes of audio data, especially within organizations that rely on call-center operations, sales departments, customer-support units, or CRM-driven workflows. Despite the strategic importance of these interactions, a significant portion of valuable information remains unstructured, locked inside voice recordings that are rarely analyzed in real time. Manual transcription is not only time-consuming but also highly error-prone, as human operators tend to overlook important details, shorten dialogues, or interpret content inconsistently.

The increasing complexity of digital customer journeys intensifies the need for intelligent automation. Companies often maintain multiple communication channels – telephony, messengers, email – and must synchronize customer data across all of them. In such environments, the gap between raw audio and actionable insights becomes a bottleneck that directly limits sales efficiency, service quality, and management decision-making. Although CRM systems have evolved into sophisticated ecosystems, they still lack native tools capable of analyzing spoken communication at a semantic level. As a result, call data is often stored as audio archives without structured annotations, summaries, or insights that could meaningfully support business processes.

The rapid development of artificial intelligence models – particularly large language models and neural speech-recognition systems [1] – opens new possibilities for automatic extraction of knowledge from voice communication. Technologies such as OpenAI Whisper [2] enable high-accuracy speech-to-text conversion even in noisy environments, while multimodal models like GPT-4o [3] offer advanced reasoning capabilities, including speaker diarization, dialogue interpretation, and structured information extraction. When integrated into CRM workflows, these models can serve as a powerful analytical layer, transforming ordinary phone calls into a constant flow of business intelligence.

Based on this context, the author's research focuses on designing, developing, and evaluating an intelligent middleware solution that bridges telephony services and CRM systems. The proposed system provides automatic transcription, diarization, summarization, identification of tasks and deadlines, and seamless synchronization with CRM records. Unlike traditional third-party services, it operates as an embedded analytical pipeline, minimizing delays and enabling organizations to convert every customer interaction into structured, searchable, and actionable digital knowledge.

Although CRM platforms have achieved a high degree of functional maturity, their built-in capabilities for audio processing remain severely limited. Systems such as KommoCRM [4], Bitrix24, Zoho CRM, and others rely primarily on manual note-taking performed by managers after each call. This introduces variability, reduces accuracy, and creates inconsistencies across teams. Moreover, traditional telephony systems provide only basic metadata – duration, phone number, call direction – while the semantic content of a conversation remains inaccessible without external processing.

A number of commercial tools attempt to address this issue, but each existing solution demonstrates structural limitations. For example, Binotel's transcription feature offers moderate accuracy and lacks full semantic extraction or diarization. Ringostat Insights achieves better transcription quality but still does not provide structured summaries, agreement extraction, or automated task generation. Third-party SaaS transcription tools provide high accuracy and can perform deeper analysis, yet their integration with CRM systems is almost always indirect, requiring manual upload of audio files or copying of generated text back into a CRM card. This disrupts workflow continuity and increases operational friction.

Academic research on speech recognition and natural-language understanding highlights continuous improvements in deep learning techniques. Whisper-based models outperform classical ASR pipelines, enabling more robust transcription in low-quality recordings. Meanwhile, transformer-based LLM architectures are capable of identifying semantic roles, extracting intents, classifying sentiments, and generating structured JSON outputs that match predefined schemas. However, most academic studies focus on model performance rather than end-to-end CRM integration, leaving a gap between theoretical advances and real business application.

The problem, therefore, consists of three key components: Lack of complete automation capable of converting raw audio into structured CRM data without human intervention; Fragmentation of tools, where transcription, analysis, and CRM updates operate as separate systems; Absence of unified architectures that combine ASR, diarization, semantic extraction, and workflow synchronization in a single real-time pipeline.

The proposed system addresses these gaps by implementing a fully integrated analytical layer that transparently connects telephony events, AI processing modules, and CRM data structures. It performs transcription via Whisper, applies GPT-4o diarization logic to distinguish speakers, extracts tasks and agreements using function-calling-based semantic analysis, and finally updates CRM records automatically. By merging these processes, the system provides a unified end-to-end solution that surpasses existing commercial tools and fills an important niche in the automation of customer-communication workflows.

To highlight these constraints, Table 1 compares the capabilities of common commercial solutions with the system developed in this research (based on the analysis in the qualification thesis).

Table 1 – Comparison of Existing Solutions and the Proposed System

Solution	Transcription Accuracy	Diarization	Semantic Analysis	CRM Integration	Automation Level
Binotel AI	Medium	No	No	Partial	Low
Ringostat Insights	High	No	Limited	Partial	Medium
External AI	Variable	Yes	Yes	No direct sync	Medium
Proposed System	High (Whisper)	Yes (GPT-4o)	Yes, structured JSON	Full KommoCRM	High

The development of the intelligent middleware system is grounded in a multi-layered methodological approach that integrates modern web technologies, scalable backend architecture, and advanced language-processing models into a unified analytical pipeline. The primary objective is to transform raw telephony audio data into structured CRM insights with minimal latency and full automation. The methodology is based on modularity, event-driven processing, and strict separation of system responsibilities, ensuring stable performance even under dynamic operational loads.

The core architectural foundation is built on Next.js [6], selected for its hybrid rendering model, efficient server-side capabilities, and native support for API routes. These routes act as dedicated endpoints for processing incoming telephony and CRM webhooks. The backend logic is implemented in Node.js with TypeScript, providing strong type safety, predictable error handling, and improved maintainability in long-term deployment scenarios. Asynchronous task handling is achieved through native promises, enabling the system to process multiple audio events concurrently without blocking the event loop.

To ensure reliable data persistence, the system uses SQLite [7] as its primary database. Although lightweight, SQLite ensures transactional integrity, high-speed read/write operations, and requires no external database server, which is ideal for webhook-driven microservices. The database stores structured logs, intermediate transcription states, diarization metadata, semantic extraction results, timestamps, and CRM synchronization history. This approach eliminates the dependency on external infrastructure, allowing the middleware to remain portable and easy to deploy across different server environments [8].

The audio-processing pipeline consists of several sequential stages. First, the system receives a webhook from the telephony provider [9] containing the URL of the recorded call. The middleware downloads the audio file, validates its format, and prepares it for processing. The transcription stage relies on a neural speech-recognition model [10] capable of maintaining high accuracy in noisy conditions, handling diverse accents, and distinguishing overlapping speech. This ensures that even low-quality recordings retain semantic clarity during analysis.

The diarization module applies speaker-role separation, identifying sections spoken by the manager and the client. This step is essential for correct interpretation of business interactions, as responsibilities, agreements, and objections typically depend on speaker identity. After diarization, the cleaned transcript undergoes semantic processing, where dialogue segments are transformed into structured entities such as tasks, deadlines, objections, agreements, context summaries, and action items. The system uses a function-oriented schema that enforces strict JSON output, ensuring complete consistency of extracted fields.

The final step of the architectural workflow is the CRM integration layer, which synchronizes results with KommoCRM. This includes pushing summaries into notes, generating tasks automatically, updating deal properties, and storing metadata for audit and analytics purposes. The event-driven architecture allows each stage to operate independently, enabling parallel execution, enhanced resilience, and simplified debugging. Together, these elements form a cohesive intelligence layer capable of capturing, interpreting, and operationalizing spoken communication within enterprise ecosystems.

The practical implementation of the system follows a production-oriented design philosophy, combining reliability, scalability, and transparency of internal processes. The backend operates as a webhook consumer, continuously listening for events triggered by telephony or CRM activities. When a call is completed, a webhook is dispatched to the middleware, initiating a processing sequence. The system retrieves the audio file, checks its availability, verifies integrity, and temporarily stores it for processing.

Transcription is performed through a high-accuracy speech model, after which the system executes stringent text-cleaning routines: removal of filler words, correction of timestamp artifacts, segmentation into logical phrases, and normalization of punctuation. These corrections significantly enhance the readability and interpretability of the output. Once the cleaned transcript is ready, the diarization logic assigns speaker labels based on linguistic cues, conversational context, and timing patterns. This ensures that dialogue structure mirrors the real dynamic of communication, which is crucial for downstream analysis.

The semantic extraction stage represents the central operational component of the system. It transforms unstructured language into structured entities, generating summaries, next steps, client expectations, risk notes, and explicit commitments made during the call. This is accomplished using schema-based prompts that enforce strict JSON formatting. The middleware validates the extracted fields, ensuring that all required elements are present. If certain fields are missing or ambiguous, the system performs an automatic secondary analysis pass to improve completeness.

After semantic extraction, the system initiates synchronization with KommoCRM. The integration logic selects the correct deal or contact card, attaches the transcript, creates tasks with deadlines, and adds a structured summary. Synchronization also includes status updates and history logging, ensuring that every analytical output is visible to managers in real time.

Error handling is implemented at several layers. The audio download stage includes fallback retries, the transcription stage has timeout limits, and the CRM synchronization layer logs failed deliveries for reprocessing. The database keeps a full audit trail, enabling administrators to trace any interaction from webhook creation to CRM injection.

In the frontend, an administrative interface built with React and Tailwind CSS allows supervisors to monitor processed calls. Users can view transcripts, inspect extracted insights, validate diarization accuracy, and track system performance. The interface also provides search functions, filtering by date, employee, or deal ID, and includes diagnostic markers that highlight calls where the system detected anomalies or incomplete segments.

Real-world testing demonstrated the robustness of the entire workflow. The time from receiving a webhook to full CRM update remains below one minute for standard call lengths. Transcription accuracy remains consistently high, and semantic fields align with managerial expectations for clarity and reliability. The practical deployment showed that the system significantly reduces manual workload, eliminates transcription delays, and improves operational transparency across the sales department.

The implementation and deployment of the intelligent middleware system produced a wide range of measurable technical, operational, and organizational outcomes, demonstrating its effectiveness and practical value within CRM-oriented environments. One of the most significant results is the system's ability to fully automate the extraction of critical business information from voice calls, eliminating the need for manual transcription or post-call documentation. This shift has dramatically reduced the routine workload previously handled by sales managers and call operators, allowing them to focus on customer engagement rather than administrative tasks.

During performance testing, transcription accuracy consistently remained at a high level, even in recordings with background noise, interruptions, or overlapping speech. This reliability is attributed to the robust architecture of the speech model and its capacity to preserve semantic clarity across diverse acoustic conditions. As a result, the produced transcripts not only reflect the spoken content but also maintain structural coherence, enabling meaningful downstream analysis. Moreover, the diarization module performed speaker-role differentiation with a high degree of precision, correctly identifying which parts of the conversation were spoken by the client and which by the manager. This is especially important for extracting agreement structures, commitments, objections, and follow-up actions.

Another core result concerns the semantic extraction capabilities of the system. The structured JSON outputs consistently contained summaries, tasks, deadlines, key decisions, client expectations, risk indicators, and contextual interpretations. Compared to manual reporting, which typically ranges from brief notes to incomplete summaries, the automated outputs were significantly more detailed and consistent. This uniformity supports analytical processes, improves record accuracy across the CRM environment, and decreases the likelihood of miscommunication within sales teams.

From an operational perspective, the system significantly increases workflow speed. The time required to process a single call – from the moment the webhook is triggered to the completion of CRM updates – remained under one minute. This reduction from traditional multi-minute manual workflows to automated processing greatly enhances team throughput and allows employees to handle a larger number of interactions within the same time frame. In addition, automated updates ensure that every call contributes to transparent and traceable communication histories, which is crucial for management oversight, dispute resolution, and auditing of customer interactions.

Managers and analysts also observed qualitative improvements. The system makes key insights immediately accessible, allowing supervisors to monitor communication patterns, evaluate employee performance, and detect behavioral changes. For example, repeated objections, missed opportunities, or recurring customer concerns become visible as aggregated patterns over time. In turn, this supports training, customer-experience optimization, and strategic decision-making.

Furthermore, the deployment demonstrated that an integrated AI layer can serve as a form of organizational memory, preserving the nuances of spoken communication and ensuring that knowledge is not lost when employees rotate or when responsibilities shift. This characteristic alone places the system at an advantage over traditional CRM workflows, where the continuity of knowledge is often dependent on individual workers.

Finally, the system's modularity allows it to adapt to various business contexts. Whether dealing with sales, support, consultation services, or internal communication monitoring, the architecture supports flexible expansion. It can integrate additional analytic modules, perform sentiment classification, detect emotional tone, measure conversational efficiency, or evaluate the complexity of client requests. As a result, the system not only addresses current organizational needs but also establishes a foundation for advanced AI-driven enterprise analytics that can evolve in parallel with business requirements.

The research conducted in this project demonstrates that an integrated AI-powered middleware system can serve as a transformative component within modern CRM environments, reshaping the way organizations process, interpret, and utilize spoken communication. The developed solution successfully bridges the technological gap between telephony services and CRM platforms, converting raw audio recordings into structured insights that support strategic, managerial, and operational decision-making.

The system's architecture – built upon Next.js, Node.js, TypeScript, high-accuracy speech-recognition models, and structured semantic extraction workflows – provides a robust technological foundation capable of meeting the challenging requirements of real-world operational environments. The combination of Whisper-based transcription, GPT-4o diarization, and large-scale language analysis enables highly precise and context-aware interpretation of customer calls. This proves that modern AI models are not limited to experimental scenarios but can be effectively embedded into business processes with significant practical benefits.

A key conclusion of the study is that full automation of call analysis eliminates long-standing inefficiencies traditionally associated with CRM documentation practices. By reducing manual workload, enhancing data accuracy, and ensuring consistency across hundreds of interactions, the system dramatically increases productivity. The transformation of audio into a searchable and structured digital format strengthens internal communication, improves accountability, and supports customer-relationship management through more informed and data-driven decision-making.

Moreover, the project shows that integrating AI modules into CRM ecosystems is not only feasible but highly scalable. The system's modular design allows for rapid adaptation, enabling organizations to expand their analytical capabilities by adding new features such as sentiment evaluation, objection tracking, customer-behavior forecasting, automated quality scoring, or real-time conversational assistance. These possibilities illustrate how the architecture can evolve into a full-fledged enterprise intelligence platform capable of supporting long-term organizational goals.

In summary, the development and evaluation of the middleware system confirm its ability to enhance operational transparency, accelerate business processes, and improve the quality of customer interactions. The work establishes a strong foundation for future research, including the implementation of real-time streaming transcription, deployment of local offline models for privacy-sensitive environments, advanced multi-modal analytics, and cross-system synchronization with enterprise-level software solutions. The results demonstrate that incorporating artificial intelligence into CRM workflows is a strategically valuable step that opens new opportunities for innovation, efficiency, and long-term competitive advantage.

1) Jurafsky, D., & Martin, J. H. (2024). Speech and Language Processing. Stanford University. https://web.stanford.edu/~jurafsky/slp3/. (date of application 09.11.2025)
2) Radford, A., Kim, J. W., Xu, T., et al. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. OpenAI. https://cdn.openai.com/papers/whisper.pdf. (date of application 10.11.2025)
3) OpenAI. (2023). GPT-4 Technical Report. OpenAI. https://arxiv.org/abs/2303.08774 (date of application 15.11.2025)
4) Kommo Developers. (2025). API Documentation. Kommo. https://developers.kommo.com/ (date of application 17.11.2025)
5) Vercel. (2025). The React Framework for the Web. Next.js Documentation. https://nextjs.org/docs (date of application 20.11.2025)
6) SQLite Development Team. (2025). SQLite Documentation. SQLite. https://www.sqlite.org/docs.html (date of application 23.11.2025)
7) Docker Inc. (2025). Docker Documentation. Docker. https://docs.docker.com/ (date of application 24.11.2025)
8) Phonet. (2025). API Documentation. Phonet. https://phonet.ua/api/ (date of application 28.11.2025)
9) OpenAI. (2025). Speech-to-text (Whisper). OpenAI API Documentation. https://platform.openai.com/docs/guides/speech-to-text (date of application 01.12.2025)

Додаток Б

Схема бази даних (prisma schema)

```prisma
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
model Call {
  id             String    @id @default(cuid())
  externalId     String    @unique
  phoneNumber    String
  managerPhone   String
  managerId      String?
  duration       Int
  audioUrl       String
  transcript     String?
  summary        String?
  sentimentScore Int?
  topics         String[]
  status         String    @default("processing")
  createdAt      DateTime  @default(now())
  processedAt    DateTime?
  updatedAt      DateTime  @updatedAt
  user           User?     @relation(fields: [managerId], references:
[id])
  tasks          Task[]
  crmDealId      String?
  @@map("calls")
}
model Task {
  id             String    @id @default(cuid())
  title          String
  description    String?
  status         String    @default("todo")
  priority       String    @default("medium")
  sourceCallId   String
  managerId      String?
  dueDate        DateTime?
  completedAt    DateTime?
  createdAt      DateTime  @default(now())
  updatedAt      DateTime  @updatedAt
  call           Call?     @relation(fields: [sourceCallId], references:
[id])
  user           User?     @relation(fields: [managerId], references:
[id])
  @@map("tasks")
}```
```

Додаток В

Компонент детальної аналітики

```typescript
// src/app/[locale]/(app)/calls/[id]/page.tsx
import { ContentLayout } from "@/components/admin-panel/content-layout";
import { PageHeader } from "@/components/admin-panel/page-header";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth";
import { redirect } from "next/navigation";
import { getTranslations } from "next-intl/server";
import { getCallById, calls } from "@/lib/mock-calls-data";
import { CallTranscript } from "./components/CallTranscript";
import { CallAnalysis } from "./components/CallAnalysis";
import { CallActions } from "./components/CallActions";
import { notFound } from "next/navigation";
import { Card, CardContent } from "@/components/ui/card";
export default async function CallDetailsPage({
  params,
  searchParams
}: {
  params: { locale?: string; id?: string };
  searchParams?: { returnUrl?: string };
}) {
  const locale = params?.locale || 'en';
  const callId = params?.id;
  const session = await getServerSession(authOptions);
  if (!session || !session.user?.id) {
    redirect(`/${locale}/auth/signin`);
  }
  if (!callId) {
    notFound();
  }
  const callDetails = getCallById(callId);
  if (!callDetails) {
    notFound();
  }
  const t = await getTranslations('CallDetails');
  const call = calls.find(c => c.id === callId);
  const callDate = call ? new Date(call.date).toLocaleDateString(locale
=== 'uk' ? 'uk-UA' : 'en-US') : new Date().toLocaleDateString(locale ===
'uk' ? 'uk-UA' : 'en-US');
  return (
    <ContentLayout title={`${t('title')} - ${callId}`}>
      <PageHeader
        title={`${callId} - ${callDate}`}
        crumbs={[]}/>
      <div className="space-y-6">
        <div className="mt-6">
          <Card className="rounded-lg border-none">
            <CardContent className="p-12">
```

```
              <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
                <CallTranscript transcript={callDetails.transcript} />
                <CallAnalysis
                  summary={callDetails.summary}
                  sentimentScore={callDetails.sentimentScore}
                  topics={callDetails.topics}
                  audioUrl={callDetails.audioUrl}/>
                <CallActions
                  dealInfo={callDetails.dealInfo}
                  tasks={callDetails.tasks}/>
              </div>
            </CardContent>
          </Card>
        </div>
      </div>
    </ContentLayout>
  );
}
```

```typescript
// src/app/[locale]/(app)/calls/[id]/components/CallAnalysis.tsx
"use client"
import { useState, useRef, useEffect } from "react"
import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card"
import { Badge } from "@/components/ui/badge"
import { Button } from "@/components/ui/button"
import { Progress } from "@/components/ui/progress"
import { Separator } from "@/components/ui/separator"
import { Play, Pause } from "lucide-react"
import { useTranslations } from "next-intl"
import { cn } from "@/lib/utils"
interface CallAnalysisProps {
  summary: string
  sentimentScore: number
  topics: string[]
  audioUrl: string
}
export function CallAnalysis({
  summary,
  sentimentScore,
  topics,
  audioUrl,
}: CallAnalysisProps) {
  const t = useTranslations('CallDetails')
  const [isPlaying, setIsPlaying] = useState(false)
  const [currentTime, setCurrentTime] = useState(0)
  const [duration, setDuration] = useState(100)
  const audioRef = useRef<HTMLAudioElement>(null)
  useEffect(() => {
    const audio = audioRef.current
```

```
    if (!audio) return
    const updateTime = () => setCurrentTime(audio.currentTime)
    const updateDuration = () => setDuration(audio.duration || 100)
    const handleEnded = () => setIsPlaying(false)
    audio.addEventListener('timeupdate', updateTime)
    audio.addEventListener('loadedmetadata', updateDuration)
    audio.addEventListener('ended', handleEnded)
    return () => {
      audio.removeEventListener('timeupdate', updateTime)
      audio.removeEventListener('loadedmetadata', updateDuration)
      audio.removeEventListener('ended', handleEnded)
    }
  }, [])
  const handlePlayPause = () => {
    const audio = audioRef.current
    if (!audio) return

    if (isPlaying) {
      audio.pause()
    } else {
      audio.play().catch(() => setIsPlaying(false))
    }
    setIsPlaying(!isPlaying)
  }
  const getSentimentColor = (score: number) => {
    if (score >= 70) return "text-green-600 dark:text-green-400"
    if (score >= 40) return "text-yellow-600 dark:text-yellow-400"
    return "text-red-600 dark:text-red-400"
  }
  return (
    <Card className="h-full flex flex-col">
      <CardHeader>
        <CardTitle>{t('analysis.title')}</CardTitle>
      </CardHeader>
      <CardContent className="space-y-6">
        <div className="space-y-3">
          <h3 className="text-sm font-
semibold">{t('analysis.audioPlayer')}</h3>
          <div className="flex items-center gap-3">
            <Button
              variant="outline"
              size="icon"
              onClick={handlePlayPause}
              className="h-10 w-10">
              {isPlaying ? (
                <Pause className="h-4 w-4" />
              ) : (
                <Play className="h-4 w-4" />
              )}
            </Button>
            <div className="flex-1 space-y-1">
```

```
              <Progress value={(currentTime / duration) * 100}
className="h-2" />
              <div className="flex justify-between text-xs text-muted-
foreground">
                <span>{formatTime(currentTime)}</span>
                <span>{formatTime(duration)}</span>
              </div>
            </div>
          </div>
          <audio
            ref={audioRef}
            src={audioUrl}
            className="hidden"/>
        </div>
        <Separator />
        <div className="space-y-3">
          <h3 className="text-sm font-
semibold">{t('analysis.sentimentScore')}</h3>
          <div className="space-y-2">
            <div className="flex items-center justify-between">
              <span className="text-sm text-muted-
foreground">{t('analysis.score')}</span>
              <span className={cn("text-2xl font-bold",
getSentimentColor(sentimentScore))}>
                {sentimentScore}
              </span>
            </div>
            <Progress
              value={sentimentScore}
              className="h-3"
            />
            <div className="flex justify-between text-xs text-muted-
foreground">
              <span>0</span>
              <span>100</span>
            </div>
          </div>
        </div>
        <Separator />
        <div className="space-y-3">
          <h3 className="text-sm font-
semibold">{t('analysis.summary')}</h3>
          <div className="rounded-md border bg-muted/50 p-4">
            <p className="text-sm leading-relaxed whitespace-pre-
wrap">{summary}</p>
          </div>
        </div>
        <Separator />
        <div className="space-y-3">
          <h3 className="text-sm font-
semibold">{t('analysis.topics')}</h3>
```

```
          <div className="flex flex-wrap gap-2">
            {topics.map((topic, index) => (
              <Badge key={index} variant="secondary">
                {topic}
              </Badge>
            ))}
          </div>
        </div>
      </CardContent>
    </Card>
  )
}
function formatTime(seconds: number): string {
  const mins = Math.floor(seconds / 60)
  const secs = Math.floor(seconds % 60)
  return `${mins}:${secs.toString().padStart(2, '0')}`
}
```

```typescript
// src/app/[locale]/(app)/calls/[id]/components/CallTranscript.tsx
"use client"
import { ScrollArea } from "@/components/ui/scroll-area"
import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card"
import { TranscriptMessage } from "@/lib/mock-calls-data"
import { useTranslations } from "next-intl"
import { cn } from "@/lib/utils"
interface CallTranscriptProps {
  transcript: TranscriptMessage[]
}
export function CallTranscript({ transcript }: CallTranscriptProps) {
  const t = useTranslations('CallDetails')
  return (
    <Card className="h-full flex flex-col">
      <CardHeader>
        <CardTitle>{t('transcript.title')}</CardTitle>
      </CardHeader>
      <CardContent className="flex-1 p-0">
        <ScrollArea className="h-[600px] px-6">
          <div className="space-y-4 py-4">
            {transcript.map((message, index) => (
              <div
                key={index}
                className={cn(
                  "flex flex-col gap-1 max-w-[80%]",
                  message.role === 'manager' ? "items-start" : "items-end
ml-auto"
                )}>
                <div
                  className={cn(
                    "rounded-lg px-4 py-2 text-sm",
```

```
                    message.role === 'manager'
                      ? "bg-blue-500 text-white"
                      : "bg-muted text-foreground"
                  )}>
                  <p className="whitespace-pre-wrap">{message.text}</p>
                </div>
                <span className="text-xs text-muted-foreground px-2">
                  {message.time}
                </span>
              </div>
            ))}
          </div>
        </ScrollArea>
      </CardContent>
    </Card>
  )
}
```

```typescript
// src/app/[locale]/(app)/calls/[id]/components/CallActions.tsx
"use client"
import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Separator } from "@/components/ui/separator"
import { ExternalLink, CheckSquare } from "lucide-react"
import { useTranslations } from "next-intl"
import { DealInfo, CallTask } from "@/lib/mock-calls-data"
import { Checkbox } from "@/components/ui/checkbox"
import { Label } from "@/components/ui/label"
import { useState } from "react"
import { cn } from "@/lib/utils"
interface CallActionsProps {
  dealInfo: DealInfo
  tasks: CallTask[]
}
export function CallActions({ dealInfo, tasks }: CallActionsProps) {
  const t = useTranslations('CallDetails')
  const [taskStates, setTaskStates] = useState<Record<string, boolean>>(
    tasks.reduce((acc, task) => ({ ...acc, [task.id]: task.completed }),
{})
  )
  const handleTaskToggle = (taskId: string) => {
    setTaskStates(prev => ({
      ...prev,
      [taskId]: !prev[taskId]
    }))
  }
  return (
    <Card className="h-full flex flex-col">
      <CardHeader>
```

```
        <CardTitle>{t('actions.title')}</CardTitle>
      </CardHeader>
      <CardContent className="space-y-6">
        <div className="space-y-3">
          <h3 className="text-sm font-
semibold">{t('actions.dealInfo')}</h3>
          <div className="rounded-md border bg-muted/50 p-4 space-y-3">
            <div className="space-y-2">
              <div className="flex justify-between items-center">
                <span className="text-sm text-muted-
foreground">{t('actions.clientName')}</span>
                <span className="text-sm font-
medium">{dealInfo.clientName}</span>
              </div>
              <div className="flex justify-between items-center">
                <span className="text-sm text-muted-
foreground">{t('actions.dealId')}</span>
                <span className="text-sm font-medium font-
mono">{dealInfo.id}</span>
              </div>
              <div className="flex justify-between items-center">
                <span className="text-sm text-muted-
foreground">{t('actions.amount')}</span>
                <span className="text-sm font-
medium">{dealInfo.amount}</span>
              </div>
              <div className="flex justify-between items-center">
                <span className="text-sm text-muted-
foreground">{t('actions.stage')}</span>
                <span className="text-sm font-
medium">{dealInfo.stage}</span>
              </div>
            </div>
            <Separator />
            <Button
              variant="outline"
              className="w-full"
              onClick={() =>
window.open(`https://kommo.com/deal/${dealInfo.id}`, '_blank')}>
              <ExternalLink className="mr-2 h-4 w-4" />
              {t('actions.openInCrm')}
            </Button>
          </div>
        </div>
        <Separator />
        <div className="space-y-3">
          <div className="flex items-center gap-2">
            <CheckSquare className="h-4 w-4" />
            <h3 className="text-sm font-
semibold">{t('actions.actionItems')}</h3>
          </div>
```

```
          <div className="space-y-3">
            {tasks.map((task) => (
              <div
                key={task.id}
                className="flex items-start space-x-3 rounded-md border
p-3">
                <Checkbox
                  id={task.id}
                  checked={taskStates[task.id] || false}
                  onCheckedChange={() => handleTaskToggle(task.id)}
                  className="mt-0.5"/>
                <Label
                  htmlFor={task.id}
                  className={cn(
                    "text-sm leading-relaxed cursor-pointer flex-1",
                    taskStates[task.id] && "line-through text-muted-
foreground"
                  )}>
                  {task.text}
                </Label>
              </div>
            ))}
          </div>
        </div>
      </CardContent>
    </Card>
  )
}
```

Додаток Г

Модуль обробки вебхуків

```typescript
// src/app/api/webhooks/phonet/route.ts
import { NextRequest, NextResponse } from 'next/server'
import { prisma } from '@/lib/prisma'
import OpenAI from 'openai'
const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
})
interface PhonetWebhookPayload {
  callId: string
  phoneNumber: string
  managerPhone: string
  duration: number
  audioUrl: string
  timestamp: string
  status: 'completed' | 'failed' | 'missed'
}
export async function POST(request: NextRequest) {
  try {
    const body: PhonetWebhookPayload = await request.json()

    // Валідація payload
    if (!body.callId || !body.audioUrl) {
      return NextResponse.json(
        { error: 'Invalid payload: callId and audioUrl are required' },
        { status: 400 }
      )
    }
    // Перевірка чи дзвінок вже оброблений
    const existingCall = await prisma.call.findUnique({
      where: { externalId: body.callId }
    })
    if (existingCall) {
      return NextResponse.json(
        { message: 'Call already processed', callId: existingCall.id },
        { status: 200 }
      )
    }
    // Створення запису дзвінка в БД
    const call = await prisma.call.create({
      data: {
        externalId: body.callId,
        phoneNumber: body.phoneNumber,
        managerPhone: body.managerPhone,
        duration: body.duration,
        audioUrl: body.audioUrl,
        status: 'processing',
        createdAt: new Date(body.timestamp),
```

```
      }
    })
    // Завантаження аудіо файлу
    const audioResponse = await fetch(body.audioUrl)
    const audioBuffer = await audioResponse.arrayBuffer()
    // Транскрипція через OpenAI Whisper
    const transcription = await openai.audio.transcriptions.create({
      file: new File([audioBuffer], 'audio.mp3', { type: 'audio/mpeg' }),
      model: 'whisper-1',
      language: 'uk',
    })
    const transcriptText = transcription.text
    // Аналіз через GPT-4o
    const analysisPrompt = `Проаналізуй цей телефонний дзвінок між
менеджером та клієнтом.
    Надай:
    1. Короткий резюме розмови
    2. Оцінку настрою (0-100, де 100 - дуже позитивний)
    3. Список тем обговорення
    4. Список завдань/дій, які потрібно виконати
    Транскрипція:
    ${transcriptText}`
    const analysisResponse = await openai.chat.completions.create({
      model: 'gpt-4o',
      messages: [
        {
          role: 'system',
          content: 'Ти експерт з аналізу телефонних дзвінків. Аналізуй
розмови та витягуй структуровані дані.'
        },
        {
          role: 'user',
          content: analysisPrompt
        }
      ],
      temperature: 0.3,
      response_format: { type: 'json_object' }
    })
    const analysis =
JSON.parse(analysisResponse.choices[0].message.content || '{}')
    // Оновлення запису дзвінка з результатами аналізу
    await prisma.call.update({
      where: { id: call.id },
      data: {
        status: 'processed',
        transcript: transcriptText,
        summary: analysis.summary || '',
        sentimentScore: analysis.sentimentScore || 50,
        topics: analysis.topics || [],
        processedAt: new Date(),
      }
```

```
    })
    // Створення завдань з аналізу
    if (analysis.tasks && Array.isArray(analysis.tasks)) {
      for (const taskText of analysis.tasks) {
        await prisma.task.create({
          data: {
            title: taskText,
            status: 'todo',
            priority: 'medium',
            sourceCallId: call.id,
            managerId: call.managerId,
            dueDate: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000), //
+7 днів}})}}
    // Синхронізація з КоммоCRM (якщо потрібно)
    if (process.env.KOMMO_API_KEY && analysis.dealInfo) {
      await syncWithKommoCRM(call.id, analysis.dealInfo)
    }
    return NextResponse.json(
      {
        message: 'Call processed successfully',
        callId: call.id,
        analysis: {
          sentimentScore: analysis.sentimentScore,
          topicsCount: analysis.topics?.length || 0,
          tasksCount: analysis.tasks?.length || 0
        }
      },
      { status: 200 }
    )
  } catch (error) {
    console.error('Webhook processing error:', error)
    // Логування помилки
    await prisma.systemLog.create({
      data: {
        level: 'error',
        message: `Webhook processing failed: ${error instanceof Error ?
error.message : 'Unknown error'}`,
        metadata: { error: String(error) }
      }
    })
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    )
  }
}
async function syncWithKommoCRM(callId: string, dealInfo: any) {
  // Реалізація синхронізації з КоммоCRM
  // ...
}
```