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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка голосового агента для автоматизації бізнес-процесів з обслуговування клієнтських запитів на базі RAG з використанням мікросервісної архітектури»

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

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

_____ Андрій АРТЕМЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Андрій АРТЕМЕНКО

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Артеменку Андрію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка голосового агента для автоматизації бізнес-процесів з обслуговування клієнтських запитів на базі RAG з використанням мікросервісної архітектури»
керівник кваліфікаційної роботи доктор філософії (PhD) Богдан ХУДІК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про бізнес-процеси в сфері гостинності та типові запити клієнтів; огляд та принципи роботи технології RAG та LLM; основи функціонування голосових агентів, включаючи технології STT та TTS; концепції мікросервісної архітектури, BFF та DDD; аналіз існуючих комерційних та відкритих рішень для створення голосових агентів, CRM-систем та платформ для розгортання комунікаційних сервісів; технічна документація для використаних бібліотек та фреймворків.

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

1. Дослідити характерні риси сфери гостинності, виділити сегмент бізнесу за родом діяльності, виокремити основні процеси та канали спілкування характерні для цього сегменту.
2. Проаналізувати існуючі аналоги програмних засобів для обслуговування вербальних запитів в сфері гостинності.
3. Провести аналіз функціональних та нефункціональних вимог до програмного забезпечення, беручи до уваги особливості людського вербального спілкування.
4. Обрати програмні засоби, сервіси та платформи для реалізації програмного продукту.
5. Спроєктувати архітектуру програмного забезпечення з фокусом на зменшення ризиків під час розробки та масштабованість системи.
6. Розробити програмне забезпечення. Протестувати отримане в результаті програмне забезпечення на відповідність поставленим вимогам, використовуючи автоматизоване та ручне тестування.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Типи доменів.
6. Діаграма класів shared модулю сервісів CRM Manager та Post-processing.
7. Інфологічна модель бази даних CRM.
8. Даталогічна модель бази даних CRM.
9. Діаграма діяльності.
10. Екранні форми.
11. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.02.2025	
2	Аналіз та дослідження існуючих аналогів	05.02-12.02.2025	
3	Дослідження характерних Рис. сфери гостинності, виділення сегменту бізнесу за родом діяльності, виокремлення основних процесів та каналів спілкування характерні для цього сегменту.	13.02-05.03.2025	
4	Аналіз існуючих аналогів програмних засобів для обслуговування вербальних запитів в сфері гостинності.	06.03-14.03.2025	
5	Аналіз функціональних та нефункціональних вимог до програмного забезпечення, беручи до уваги особливості людського вербального спілкування.	15.03-25.03.2025	
6	Вибір програмних засобів, сервісів та платформ для реалізації програмного продукту.	26.03-03.04.2025	
7	Проєктування архітектури програмного забезпечення з фокусом на зменшення ризиків під час розробки та масштабованість системи.	04.04-15.04.2025	
8	Розробка мікросервісу CRM Manager	16.04-20.04.2025	
9	Розробка мікросервісу Voice agent	21.04-25.04.2025	
10	Тестування програмного забезпечення на відповідність поставленим вимогам, використовуючи автоматизоване та ручне тестування.	25.02-28.04.2025	
10	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
11	Розробка демонстраційних матеріалів	12.05-18.05.2025	
12	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Андрій АРТЕМЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 116 стор., 6 табл., 66 рис., 35 джерел.

Мета роботи – автоматизувати обслуговування вербальних клієнтських запитів та оцінити потенціал заміщення людини шляхом розробки програмного забезпечення.

Об'єкт дослідження – процеси надання послуг з оренди житла на подобовій основі та супроводу споживача впродовж періоду проживання в орендованій оселі, використовуючи засоби голосового зв'язку.

Предмет дослідження – програмне забезпечення для обробки вербальних запитів клієнтів природною мовою стосовно подобової оренди житла в реальному часі з використанням підходу RAG на базі мікросервісної архітектури.

Короткий зміст роботи: в роботі проаналізовано предметну область гостинності, визначено функціональні та нефункціональні вимоги, а також проведено огляд існуючих платформ для створення голосових агентів та аналогічних програмних засобів, що обґрунтовує вибір технологічного стеку. Спроектовано архітектуру системи та виділено ключові мікросервіси: CRM manager, Voice agent та Post-processing. Розроблено голосового агента для автоматизації бізнес-процесів з обслуговування клієнтських запитів у сфері подобової оренди житла. Система базується на підході RAG та мікросервісній архітектурі. В роботі використано наступні технології: для реалізації серверної логіки та взаємодії з AI-сервісами – фреймворк Pipecat Flows; для розробки CRM-менеджера – NestJS та для розробки BFF – фреймворк Next.js.

Сферою використання застосунку є обробка вербальних користувацьких запитів в галузі гостинності в рамках надання послуг з подобової оренди житла.

КЛЮЧОВІ СЛОВА: ГОЛОСОВИЙ АГЕНТ, RAG, МІКРОСЕРВІСНА АРХІТЕКТУРА, DDD, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, LLM, STT, TTS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RAG – Retrieval-Augmented Generation (Генерація, доповнена пошуком)
STT – Speech-to-Text (Мова в текст)
TTS – Text-to-Speech (Текст в мову)
LLM – Large Language Model (Велика мовна модель)
CRM – Customer Relationship Management (Управління відносинами з клієнтами)
BFF – Backend for Frontend (Бекенд для фронтенду)
DDD – Domain-Driven Design (Предметно-орієнтоване проектування)
API – Application Programming Interface (Програмний інтерфейс застосунків)
VoIP – Voice over Internet Protocol (Голос по інтернет-протоколу)
VAD – Voice Activity Detector (Детектор голосової активності)
WER – Word Error Rate (Показник помилок у словах)
ASR – Automatic Speech Recognition (Автоматичне розпізнавання мови)
MCP – Model Context Protocol (Протокол контексту моделі)
JSON-RPC – JavaScript Object Notation Remote Procedure Call (Виклик віддалених процедур на основі нотації об'єктів JavaScript)
VUI – Voice User Interface (Голосовий користувацький інтерфейс)
HTML – HyperText Markup Language (Мова гіпертекстової розмітки)
SSR – Server-Side Rendering (Серверний рендеринг)
UI – User Interface (Користувацький інтерфейс)
CSS – Cascading Style Sheets (Каскадні таблиці стилів)
AWS – Amazon Web Services (Веб-сервіси Амазон)
S3 – Simple Storage Service (Сервіс простого зберігання)
QSR – Quick Service Restaurant (Ресторан швидкого обслуговування)
IoT – Internet of Things (Інтернет речей)
AVA – Aiello Voice Assistant (Голосовий помічник Aiello)
NLP – Natural Language Processing (Обробка природної мови)
CRUD – Create, Read, Update, Delete (Створити, Зчитати, Оновити, Видалити)
ORM – Object-Relational Mapping (Об'єктно-реляційне відображення)
HTTP – Hypertext Transfer Protocol (Протокол передачі гіпертексту)
DRY – Don't Repeat Yourself (Не повторюй себе)

ВСТУП

Для сьогодення є характерним те, що сфера гостинності впевнено закріпилася серед інших галузей підприємницької діяльності. Цей вид діяльності притаманний для малих та середніх бізнесів, що функціонують на певній географічній місцевості та конкурують між собою. В свою чергу, серед споживачів та власників вона, перш за все, асоціюється з високим рівнем якості послуг, що надаються – і це є запорукою стабільності та зростання бізнесу.

З-поміж великої кількості налагоджених бізнес-процесів, ми можемо виділити групу, що стосується комунікації з клієнтами. До такої групи існують чіткі вимоги, націлені на дотримання якісного обслуговування. У сучасному світі, з розвитком інформаційних технологій, значний обсяг спілкування з клієнтами перейшов у формат з використанням телекомунікаційних засобів, що, в тому числі, включає в себе телефонні розмови.

Зважаючи на наявність декількох моделей надання послуг у сфері гостинності, було прийнято рішення зосередитися на подобовій оренді житла, з ціллю забезпечити якісне та точкове покриття потреби.

Мета роботи – автоматизувати обслуговування вербальних клієнтських запитів та оцінити потенціал заміщення людини шляхом розробки програмного забезпечення.

Об’єкт дослідження – процеси надання послуг з оренди житла на подобовій основі та супроводу споживача впродовж періоду проживання в орендованій оселі, використовуючи засоби голосового зв’язку.

Предмет дослідження – програмне забезпечення, націлене на обробку запитів клієнтів в контексті подобової оренди житла в реальному часі.

Для досягнення заданої цілі було виокремлено наступні задачі:

1. дослідити характерні риси сфери гостинності, виділити сегмент бізнесу за родом діяльності, виокремити основні процеси та канали спілкування характерні для цього сегменту;
2. проаналізувати існуючі аналоги програмних засобів для обслуговування вербальних запитів в сфері гостинності;
3. провести аналіз функціональних та нефункціональних вимог до програмного забезпечення, беручи до уваги особливості людського вербального спілкування;
4. обрати програмні засоби, сервіси та платформи для реалізації програмного продукту;
5. спроєктувати архітектуру програмного забезпечення з фокусом на стабільність роботи, зменшення ризиків під час розробки та масштабованість системи;
6. розробити та протестувати отримане в кінцевому підсумку програмне забезпечення відповідно до поставлених вимог

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИОКРЕМЛЕННЯ ПРОЦЕСІВ ДЛЯ АВТОМАТИЗАЦІЇ

1.1 Загальний огляд характерних Рис. сфери гостинності

Індустрія гостинності асоціюється з особливою увагою та ставленням до споживачів, та функціонує на засадах стандартизованих правил, що забезпечують злагоджену роботу процесів, безпеку гостей та чіткість очікувань між сторонами. Ці правила можуть варіюватися в межах організацій, але, як правило, вони побудовані на загальних принципах менеджменту в сфері обслуговування гостей.

Час виселення та заселення варіюється в загальноприйнятих рамках, що дозволяє краще планувати час як гостям, так і обслуговуючому персоналу. Дуже поширеним часом для заселення є період з 13:00 до 14:00, а виселення, як правило, відбувається до 12:00. Такий проміжок часу дозволяє обслуговуючому персоналу підготувати житло для наступних гостей. З іншого боку, це створює сплески кількості гостей, які потребують одночасного обслуговування, що створює велике навантаження на персонал. Мати масштабовану систему автоматизації може буде набагато вигіднішим, ніж тримати постійний персонал.

Також, до переліку характерних правил для цієї сфери можна віднести політики бронювання та його скасування. Загальні практики містять положення для гарантованих та негарантованих бронювань, що різняться умовами щодо внесення та відшкодування оплати. Більша частина закладів та підприємців можуть просити клієнта про оплату певної суми коштів для бронювання житла, а також класифікувати скасування броні як «своєчасне» або «пізнє», що може призвести до штрафних санкцій. В той час, коли сума, необхідна для бронювання, та умови скасування можуть варіюватися в рамках бізнес-моделі, сам факт наявності цих правил є спільним для великої кількості закладів. Працюючи з часовими проміжками, співробітники можуть допускатися

помилок, в той час як програмне забезпечення значно зменшує вірогідність похибки, що захищає бізнес від непередбачуваних витрат.

Варто зазначити, що під перелік загальних правил можуть також підпадати політики щодо: куріння, допустимого рівня шуму, проведення вечірок та подій, кількості гостей та домашніх улюбленців. Шляхом забезпечення обізнаності клієнтів щодо цих правил, заклади можуть створити комфортну атмосферу для інших гостей чи мешканців прилеглих територій. Ознайомлення нових співробітників з цими правилами займатиме час, так само як і зміна цих правил потребуватиме часу на перенавчання усього персоналу. У випадку автоматизованої системи, актуалізація бази знань відбувається миттєво, без додаткових витрат часових ресурсів.

Окрім внутрішніх правил, на бізнеси у сфері гостинності також мають значний вплив зовнішні чинники, про які йтиметься далі.

Як і для інших бізнесів, цій галузі притаманна сезонність, що зумовлена сприятливими періодами для відпусток, святами, локальними подіями, або ж навіть тижневим циклом. Це зумовлює стрімкий попит серед потенційних клієнтів, що створює потребу для бізнесу у збільшенні власних ресурсів для закриття пропозиції на ринку.

Важливо також взяти до уваги той факт, що нерідко ключовий сегмент споживачів становлять іноземці. Це зумовлює додаткові вимоги до обслуговуючого персоналу, які базуються на володінні однією чи декількома іноземними мовами.

Беручи до уваги всі вищенаведені фактори, можна підсумувати, що процес масштабування, особливо для малого бізнесу, може стати досить складною задачею. Це може стати бар'єром до масштабування, особливо у періоди сезонних сплесків, через наступні фактори:

1. відносна складність предметної області, зумовлена наявністю правил та політик, створює додаткові перешкоди та часові витрати під час розширення штату співробітників. Це викликає труднощі в плануванні кадрових ресурсів, оскільки кожного нового працівника потрібно не лише ознайомити з широким пакетом внутрішніх норм і політик, але й адаптувати до коливань сезонного попиту та специфіки роботи в різні зміни, що значно подовжує період «виходу на продуктивність» і ускладнює точне прогнозування чисельності та складання графіків персоналу;
2. у сфері гостинності очікується не лише базове задоволення потреб клієнтів, але й створення приємного, персоналізованого досвіду. Це передбачає високий рівень емоційного інтелекту та комунікаційних навичок у співробітників, що є складнодосяжним при швидкому масштабуванні;
3. сфера гостинності тісно пов'язана з туризмом, погодними умовами, політичною стабільністю, санітарно-епідеміологічною ситуацією та іншими подіями чи явищами, які можуть різко змінити попит. Ці фактори можуть призводити як до різкого зростання, так і до зменшення кількості клієнтів, що ускладнює планування та інвестування в розширення бізнесу.

У цьому контексті вагому роль відіграє впровадження автоматизованих систем управління. Такі рішення можуть значно зменшити навантаження на адміністративний персонал. Для малого бізнесу автоматизація є не просто перевагою, а інструментом виживання і зростання в умовах високої конкуренції та нестабільного попиту. Правильно впроваджена автоматизація, шляхом застосування сучасних технологій, може компенсувати нестачу людських ресурсів, підвищити рівень сервісу та створити основу для сталого

масштабування, зберігаючи при цьому індивідуальний підхід до кожного клієнта.

1.2 Дослідження основних каналів та процесів комунікації з клієнтами в галузі гостинності та огляд способів покращення існуючих процесів за допомогою хмарних систем на базі штучного інтелекту

У сучасних умовах стрімкого розвитку цифрових технологій та зростаючих очікувань клієнтів, галузь гостинності змушена адаптувати свої канали комунікації для забезпечення максимальної зручності та ефективності обслуговування. Основними способами взаємодії між представниками бізнесу та клієнтами сьогодні виступають текстові повідомлення, телефонні дзвінки та суміжні новітні автоматизовані рішення на базі штучного інтелекту. Зміна пріоритетів у виборі каналу комунікації зумовлена потребою у швидкому, персоналізованому та доступному сервісі, який відповідає ритму життя сучасного споживача.

Для сьогодення характерним є те, що текстові повідомлення є основним каналом зв'язку між бізнесами та споживачами. Це спричинено стрімким розвитком та збільшенням способів миттєвої текстової комунікації.

Індустрія гостинності наслідує глобальну тенденцію бізнесу у своїх комунікаційних уподобаннях: текстові повідомлення демонструють переваги в обсязі та зручності для багатьох взаємодій, надаючи постійний доступ до інформації та фіксацію історії спілкування, забезпечуючи комунікацію у зручний час для обох сторін. Однак, голосовий зв'язок залишається ключовим у сценаріях, коли існує складність комунікації, що зумовлена ланцюжком пов’язаних запитів, та коли є потреба у забезпеченні довірливих відносин з клієнтом.

Дослідження показують, що 95% клієнтів вважають персоналізовані телефонні дзвінки важливими для забезпечення високого рівня загального клієнтського досвіду, що підкреслює постійну актуальність голосової

комунікації в сфері гостинності. Для малих підприємств з обмеженим штатом працівників визначення цих пріоритетних сценаріїв голосової комунікації допомагає ефективно розподіляти ресурси. Хоча текстові повідомлення домінують за загальним обсягом комунікації, голосове спілкування створює можливості для більш ефективного спілкування та, відповідно, швидшої обробки користувацьких запитів, що може диференціювати їхні послуги на конкурентному ринку.

Раніше було згадано основну перевагу текстових повідомлень над дзвінками в реальному часі – текстові повідомлення є менш нав'язливими і краще підходять для збереження історії комунікації, тоді як дзвінки часто розглядаються як такі, що заважають, якщо вони не були заздалегідь узгоджені. Використання автоматизованої системи, що буде направлена на обробку вхідних клієнтських дзвінків, дозволяє уникнути вищенаведених недоліків голосового способу спілкування, при цьому зберігши усі переваги цього виду комунікації.

В рамках надання послуг з подобової оренди житла можна виокремити наступні ключові процеси, що невід’ємно базуються на активній комунікації: бронювання нерухомості на певні дати, скасування бронювання нерухомості з попереднім інформуванням про суму відшкодування, дистанційне заселення в відповідний день, супровід гостей під час проживання (надання відповідей на запити інформаційного характеру стосовно вирішення побутових питань чи дій під час надзвичайних ситуацій).

Розвиток технологій у галузі генеративного штучного інтелекту дозволяє будувати системи, здатні обробляти запити користувача, сформовані у вигляді натуральної мови, та розуміти його наміри, при цьому виконуючи певні дії з метою зміни стану системи відповідно до бізнес-правил. Великі мовні моделі здатні приймати текст на вхід та генерувати вихідний контент у вигляді тексту чи викликів “інструментів”. При цьому, використання підходу RAG дозволяє моделям будувати результуючий вивід на основі актуальних даних з системи. У зв’язці з сучасними моделями TTS та STT, а також існуючими інструментами

VoIP ми можемо розробити програмне забезпечення, що здатне здійснювати вербальну обробку запитів без людського втручання.

1.3 Дослідження ключових характеристик вербального спілкування

Вербальне спілкування характеризується комплексом взаємопов'язаних метрик, які визначають його ефективність та природність. Розуміння цих характеристик є критичним для розробки систем, що прагнуть відтворити або взаємодіяти з людською мовою. Аналіз ключових метрик дозволяє створити базис для оцінки якості вербальної комунікації та слугувати підґрунтям для встановлення стандартів технологічних рішень.

Своєчасність відповіді є фундаментальною характеристикою ефективного діалогу. В людському спілкуванні оптимальний час реакції на репліку співрозмовника становить приблизно 200-700 мілісекунд [13,27]. Тривалі паузи можуть сприйматися як вагання або незацікавленість, причому спостереження показують, що затримки можуть тривати до 1 секунди, якщо відповідь співрозмовника відхиляється від загального контексту діалогу. Але, всупереч наявності наведених даних, адаптивна своєчасність, яка враховує контекст розмови, є більш природною, ніж фіксовані інтервали відповідей.

Важливим аспектом, що безпосередньо впливає на розуміння повідомлення, є чіткість мовлення. Вона охоплює точність вимови, граматичну правильність та плавність. Для оцінки цієї характеристики використовуються метрики на кшталт розбірливості мовлення, яка може значно покращуватися залежно від інструкцій щодо чіткості вимови.

Word Error Rate (WER) — ключовий показник у технологіях розпізнавання мови (STT), що вимірює частку помилково розпізнаних слів. Цей показник варіюється від 0 до 1, де 0 означає абсолютну ідентичність з оригінальним текстом, а 1 — повну відмінність [23]. При цьому, слід брати до уваги, що швидкість мовлення, оптимальна для сприйняття, варіюється в межах 140-200 слів за хвилину [11].

Іншим критичним фактором, який впливає на якість діалогу, є наявність переривань у розмові. Вони мають подвійну природу: колаборативні переривання є індикаторами координації та узгодженості в діалозі, тоді як конкурентні переривання мають інші просодичні та жестові характеристики.

Переходячи до інтерактивності розмови, можемо підкреслити той факт, що вона базується на механізмах зворотного зв'язку та чергуванні реплік. Ефективність взаємодії визначається такими параметрами, як швидкі та послідовні реакції, частота зміни ролей у розмові та тривалість мовлення – такі параметри позитивно впливають на соціальний зв'язок у розмові. Ця взаємодія є основою для формування спільного контексту, де учасники адаптують свої позиції через різноманітні механізми комунікації.

Також одним з відмінних аспектів виступає незворотність і динамізм комунікації. Слова, висловлені в реальному часі, не підлягають редагуванню, що змушує учасників постійно адаптуватися до нових умов. Ця динаміка відображається в когерентності діалогу — логічній послідовності реплік, що формують єдиний смисловий потік.

Вищезгадані метрики утворюють цілісну систему оцінки людського вербального спілкування, де кожен параметр впливає на загальну ефективність комунікації. Розуміння цих характеристик створює основу для розробки технологічних рішень, здатних взаємодіяти з людьми природним та інтуїтивним чином, зберігаючи ключові аспекти міжособистісного спілкування.

1.4 Законодавчі регулювання в сфері обслуговування та їхній вплив на вимоги до програмного забезпечення

Відповідно до статті 30 Закону України "Про забезпечення функціонування української мови як державної", усі суб’єкти господарювання зобов’язані надавати послуги та інформацію про товари державною мовою. Зокрема, закон визначає: "мовою обслуговування споживачів в Україні є

державна мова. На прохання клієнта його персональне обслуговування може здійснюватися також іншою мовою, прийнятною для сторін" [2] .

Ця норма безпосередньо впливає на архітектуру програмних рішень для галузі гостинності та спонукає до запровадження обов'язкової підтримки української мови на рівні інтерфейсу взаємодії з клієнтами (в контексті даного дипломного проєкту, це голосовий асистент). В той же час, інтерфейси, що доступні лише працівникам, можуть не мати української локалізації з метою заощадження часових ресурсів під час розробки.

Також, враховуючи, що певну частину гостей становлять іноземці, програмне забезпечення має бути багатомовним та додатково включати англійську мову задля можливості надання базового рівня послуг іноземним гостям.

1.5 Аналіз функціональних та нефункціональних вимог до програмного забезпечення

Визначення та аналіз функціональних і нефункціональних вимог до програмного забезпечення становить фундаментальну основу для успішної розробки системи голосового асистента в галузі гостинності. Якісно визначені вимоги є критично важливими для успіху проєктів розробки, оскільки вони мінімізують ризик поширення дефектів на пізніші стадії життєвого циклу розробки програмного забезпечення. Процес інженерії вимог включає три основні види діяльності: виявлення вимог через взаємодію із зацікавленими сторонами, перетворення цих вимог у стандартизований документ та перевірка того, що вимоги дійсно визначають програмне забезпечення, яке потребує замовник. У контексті голосового асистента для готельного бізнесу особливого значення набуває чітке розмежування між функціональними вимогами, що описують конкретні послуги системи, та нефункціональними вимогами, що визначають критерії якості та продуктивності.

1.5.1 Функціональні вимоги до системи

Функціональні вимоги визначають конкретні послуги, які система повинна надавати кінцевому користувачеві, описуючи, як програмне забезпечення має реагувати на специфічні вхідні дані та як воно має поводитися в певних обставинах. У контексті голосового асистента для сфери гостинності функціональні вимоги охоплюють широкий спектр взаємодій між гостями та системою, а також адміністративні функції для персоналу готелю. Основна поведінка голосового агента має включати обробку запитів, надання відповідей та управління контекстом розмови з дотриманням природності діалогу та збереженням релевантної інформації протягом сеансу взаємодії.

Система повинна забезпечувати обробку голосових запитів на оформлення бронювання нерухомості, що передбачає комплексну взаємодію з базами даних доступності та системами управління бронюваннями. Агент повинен надавати покрокові інструкції під час дистанційного заселення користувача на місце проживання, що особливо актуально в умовах автоматизації процесів гостинності та зменшення прямого контакту з персоналом.

Важливою функціональністю є надання оперативної інформації та інструкцій у разі надзвичайних ситуацій, що забезпечує безпеку гостей та відповідає вимогам законодавства про готельну діяльність. Агент має оновлювати записи у внутрішній CRM-системі під час проведення дзвінка, забезпечуючи актуальність даних та відслідковування всіх взаємодій з клієнтами. Адміністративний функціонал включає можливість для адміністратора редагувати список нерухомості та завантажувати текстові інструкції для обробки інформаційних та екстрених запитів до бази знань нерухомості, що забезпечує гнучкість системи та можливість швидкого оновлення інформації.

Система включає в себе голосовий агент, сервіс менеджменту CRM та модуль обробки завершених дзвінків. Детальний опис функціоналу цих сервісів та модулів зображено на рисунку 1.1.

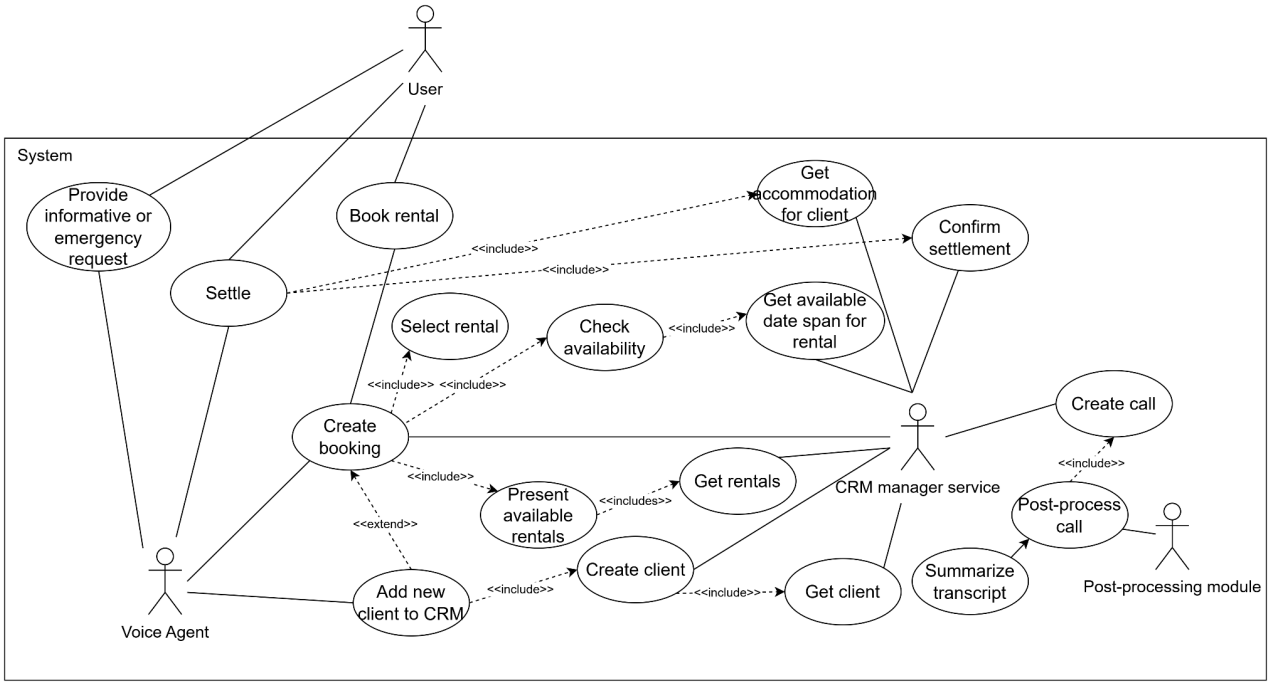


Рис. 1.1 Діаграма варіантів використання

Система реалізує декілька ключових діалогів, що покривають типові сценарії взаємодії клієнта з голосовим асистентом. Ці діалоги включають первинну ініціалізацію, обробку запитів на бронювання, заселення, а також надання інформації та допомоги в надзвичайних ситуаціях. Деталізація цих потоків представлена на діаграмах, зображених на рисунках 1.2-1.5.

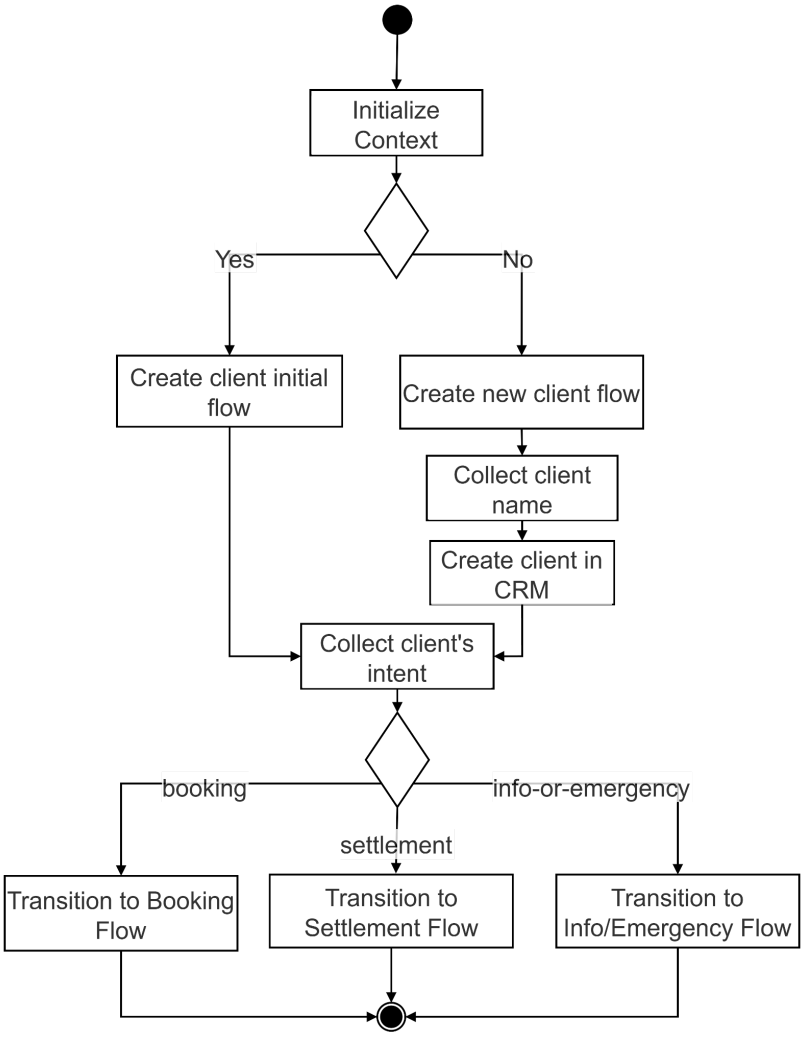


Рис. 1.2 Діаграма активності початку розмови з голосовим агентом

Початок розмови:

- кожен дзвінок розпочинається з ініціалізації контексту розмови (Initialize Context);
- система перевіряє, чи існує вже початковий потік для даного клієнта (Create client initial flow);
- якщо початковий потік не існує (новий клієнт або перша взаємодія в рамках даної сесії), ініціюється потік створення нового клієнта (Create new client flow). В рамках цього підпотоку система збирає ім'я клієнта (Collect client name) та створює відповідний запис у CRM-системі (Create client in CRM);

– після ініціалізації потоку (або його відновлення для існуючого клієнта), система переходить до збору основного наміру (інтенту) клієнта (Collect client's intent);

– на основі визначеного інтенту (наприклад, "booking", "settlement", "info-or-emergency") система перенаправляє розмову до відповідного спеціалізованого конверсаційного потоку (Transition to Settlement Flow, Transition to Info/Emergency Flow, перехід до потоку бронювання).

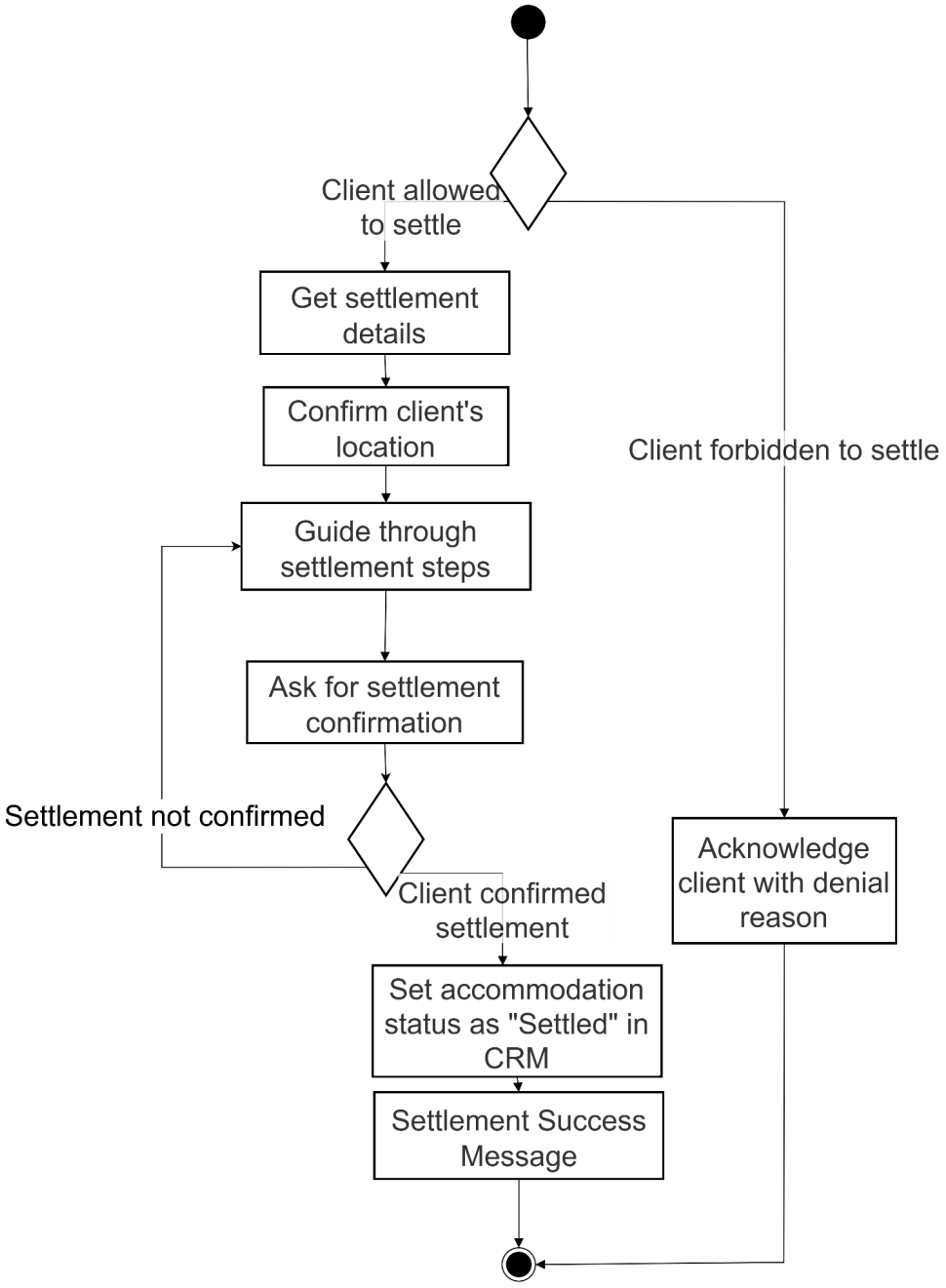


Рис. 1.3 Діаграма активності діалогу заселення

Діалог заселення:

- цей діалог активується, коли клієнт має намір, пов'язаний із заселенням;
- система насамперед перевіряє, чи дозволено клієнту заселення (Client allowed to settle). Ця перевірка включає аналіз внутрішніх правил, таких як час доби (наприклад, заселення дозволено лише після 13:00 у день заїзду), наявність активного бронювання та відповідність даних клієнта. Якщо заселення не дозволено, клієнт інформується про причину відмови (Client forbidden to settle) і отримує відповідне повідомлення (Acknowledge client with denial reason);
- якщо заселення дозволено, система збирає необхідні деталі для заселення (Get settlement details) та підтверджує місцезнаходження клієнта (Confirm client's location);
- далі система надає клієнту покрокові інструкції для проходження процедури дистанційного заселення (Guide through settlement steps) і запитує підтвердження виконання кроків (Ask for settlement confirmation);
- якщо клієнт підтверджує успішне заселення (Client confirmed settlement), статус відповідного житла оновлюється в CRM (Set accommodation status as "Settled" in CRM), і клієнт отримує повідомлення про успішне завершення процедури (Settlement Success Message);
- якщо клієнт не підтверджує заселення (Settlement not confirmed), система повідомляє про можливі проблеми або причину невиконання (Acknowledge client with denial reason).

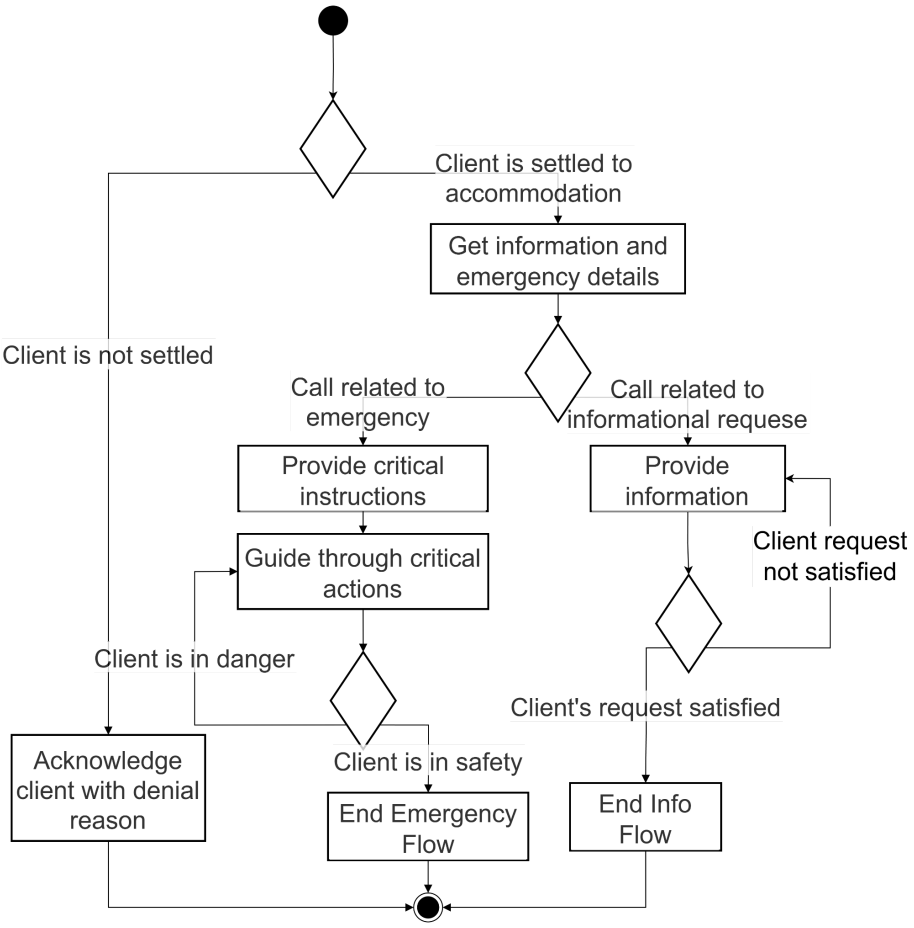


Рис. 1.4 Діаграма активності діалогу надання інформації та допомоги в надзвичайних ситуаціях

Діалог надання інформації та допомоги в надзвичайних ситуаціях:

Цей діалог ініціюється, якщо намір клієнта стосується отримання інформації або запитує допомогу в екстреній ситуації.

- система спочатку перевіряє, чи клієнт вже заселений у житло (Client is settled to accommodation), оскільки це може впливати на тип та зміст наданої інформації (наприклад, інструкції для мешканців житла);
- якщо клієнт заселений, система отримує відповідні деталі, пов'язані з інформаційним або екстреним запитом (Get information and emergency details);
- далі система визначає, чи запит стосується надзвичайної ситуації (Call related to emergency) або є суто інформаційним (Call related to informational request);

– у випадку надзвичайної ситуації: система надає критично важливі інструкції (Provide critical instructions) та направляє клієнта через необхідні дії (Guide through critical actions). Після цього система оцінює рівень безпеки клієнта (Client is in danger). Якщо клієнт у безпеці (Client is in safety), екстрений потік завершується (End Emergency Flow). Якщо клієнт залишається в небезпеці, система надає відповідне повідомлення або інструкції (Acknowledge client with denial reason).

– у випадку інформаційного запиту: система надає запитувану інформацію (Provide information) та перевіряє, чи запит клієнта задоволено (Client's request satisfied). Якщо так, інформаційний потік завершується (End Info Flow). Якщо ні, система повідомляє про можливу неможливість надати інформацію або пропонує альтернативні шляхи (Acknowledge client with denial reason).

Діалог бронювання житла (див. рисунок 1.5):

– цей діалог активується, коли намір клієнта пов'язаний з бронюванням житла;
– система представляє клієнту доступні варіанти житла (Show available rentals, Present new rental option);
– система перевіряє, чи клієнт не виявив інтересу до запропонованих варіантів (Client not interested). Якщо так, потік завершується без бронювання (End without booking);
– якщо клієнт зацікавлений, система запитує та збирає дати заїзду та виїзду (Collect check-in/out dates);
– система виконує перевірку коректності введених дат (Check dates correctness). Ця перевірка включає аналіз формату дат, логічну послідовність (дата виїзду пізніше дати заїзду) та відповідність іншим правилам системи (наприклад, мінімальний термін проживання). Якщо дати некоректні (Dates are invalid), система інформує про це клієнта;
– якщо дати коректні (Dates are valid), система перевіряє доступність обраного житла на вказані дати (Check rental availability);

- якщо житло недоступне (Not available), система інформує клієнта про це (Inform that property is not available);
- якщо житло доступне (Available), система запитує підтвердження від клієнта, чи бажає він забронювати саме це житло (Verify that Client would like to book this property);
- якщо клієнт підтверджує бронювання (Yes), система створює запис про бронювання у внутрішній системі (Create booking in system) та надає клієнту деталі успішного бронювання (Provide booking details to user);
- якщо клієнт відмовляється від бронювання на цьому етапі (No), потік завершується без створення бронювання (End without booking).

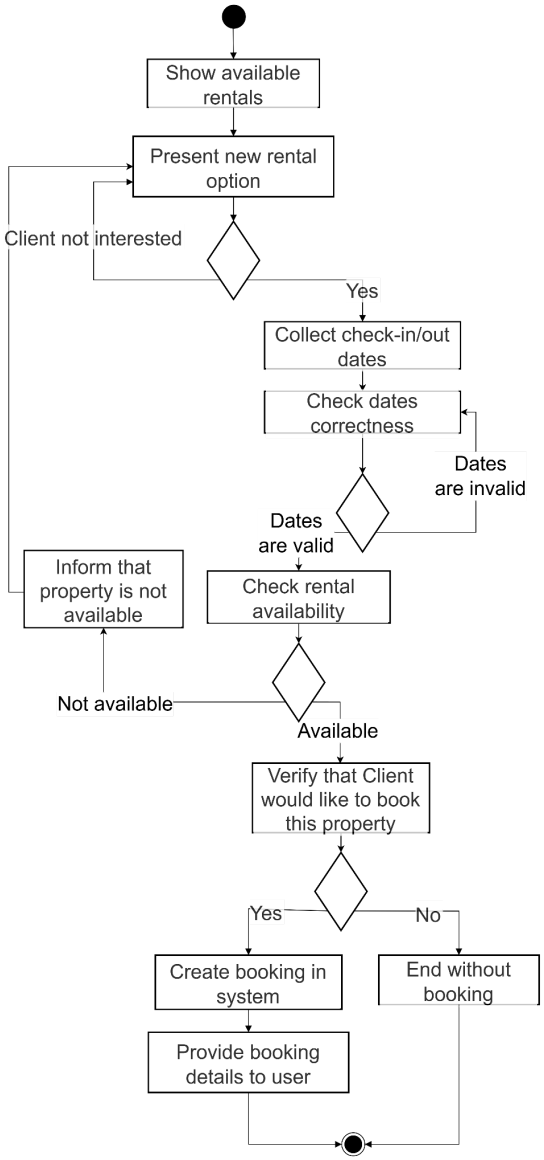


Рис. 1.5 Діаграма активності діалогу бронювання житла

Ці функціональні потоки забезпечують основу для взаємодії голосового асистента з користувачами, покриваючи ключові потреби у сфері гостинності, від початкового контакту до виконання специфічних запитів та надання допомоги.

1.5.2 Нефункціональні вимоги до системи

Нефункціональні вимоги визначають критерії якості системи та обмеження на її функціонування, що є одними з ключових критеріїв для порівняння різних програмних систем. Дослідження показують, що ідентифікація нефункціональних вимог є складним завданням. Хоч і існують добре розроблені техніки для виявлення функціональних вимог, бракує механізмів виявлення нефункціональних вимог та немає належного консенсусу щодо їх виявлення. У контексті голосового асистента, нефункціональні вимоги охоплюють продуктивність, надійність, доступність, вартісні метрики та якість голосу.

Продуктивність системи голосового асистента визначається кількома ключовими показниками, серед яких особливе значення має користувацька одночасність та затримка відповіді. Максимальна кількість одночасних користувачів, яких агент може обробляти, є визначальним параметром для планування потужностей системи. Benchmark тестування для 1000 одночасних користувачів має демонструвати стабільну роботу системи без деградації якості обслуговування, що вимагає ретельного планування архітектури та розподілу навантаження між компонентами системи.

Час відповіді від моменту завершення репліки користувачем до початку отримання відповіді від голосового асистента не має не перевищувати 1000 мс, що забезпечує природність діалогу та утримання уваги користувача. Дослідження показують, що вимірювання та аналіз продуктивності програмного забезпечення досягли високої складності через більш досконалі дизайни процесорів та складну взаємодію між користувацькими програмами, операційною системою та мікроархітектурою процесора. Складні розмови

можуть вимагати дещо довших пауз для повноцінної обробки, проте базовий ліміт у 1000 мс має залишатися цільовим показником для більшості взаємодій.

Угоди про рівень обслуговування для базових сервісів, включаючи розпізнавання мовлення, обробку природної мови та синтез мовлення, повинні гарантувати високу доступність системи та мінімізацію простоїв. Вплив на користувацький досвід при відмові або деградації одного з компонентів може бути критичним, тому система повинна включати механізми резервування та перемикання на резервні системи. Рівень помилок розпізнавання мовлення не вище 10% в середовищах без сторонніх голосів та рівнем шуму до 60 дБ забезпечує прийнятну якість взаємодії в типових умовах готельного середовища.

Вимоги до резервування системи та механізмів failover мають передбачати автоматичне переключення на резервні компоненти у випадку відмови основних систем, збереження сесій користувачів та мінімізацію втрати даних під час аварійних ситуацій. Архітектура системи повинна підтримувати горизонтальне масштабування для забезпечення стабільної роботи під час пікових навантажень, характерних для готельного бізнесу.

Характеристики голосу агента, включаючи тон, акцент, інтонацію та чіткість, мають відповідати культурним очікуванням цільової аудиторії та створювати позитивний користувацький досвід. Голосовий асистент повинен підтримувати українську та англійську мови на ввід та вивід мовлення відповідно до законодавчих вимог та потреб міжнародних гостей. Користувацький інтерфейс CRM та системи моніторингу дзвінків повинен підтримувати англійську мову для зменшення часу на розробку цього компоненту.

1.6 Аналіз існуючих програмних засобів для створення та розгортання голосового асистента. Огляд існуючих платформ та сервісів для створення голосових агентів

На сьогоднішній день, ринок платформ для створення голосових агентів представлений різноманітними рішеннями, що відрізняються функціональними можливостями, підходами до інтеграції та ціновими моделями. Розробка власного рішення потребує забезпечення безперебійної роботи асистента та безпечного процесу передачі даних. Для того, щоб зменшити ризики та акумулювати більше ресурсів на розробку, ми можемо використати існуючі платформи для розгортання голосового асистента.

Розглянемо детальніше провідні рішення, що дозволяють розробляти та впроваджувати голосових агентів для автоматизації вербальної взаємодії з клієнтами: Vapi.ai, Hume AI, Retell AI, Bland.ai, LiveKit та Daily.

1.6.1 Vapi.ai

Vapi.ai — це повноцінна платформа для розробки голосових агентів штучного інтелекту, що забезпечує розгортання голосових агентів для вхідних та вихідних дзвінків. Система поєднує потужні можливості проєктування діалогів із корпоративним рівнем безпеки. В основі платформи лежить середовище, що дозволяє менеджерам сфери гостинності візуально моделювати сценарії взаємодії з гостями без необхідності володіння технічними навичками.

Визначною характеристикою Vapi є можливість налаштовувати виклики інструментів без написання коду, що забезпечує можливість з легкістю налаштовувати виконання операцій у сторонніх системах, зокрема, оновлення даних у системах управління взаємовідносинами з клієнтами, обробку запитів на підвищення категорії номера чи бронювання.

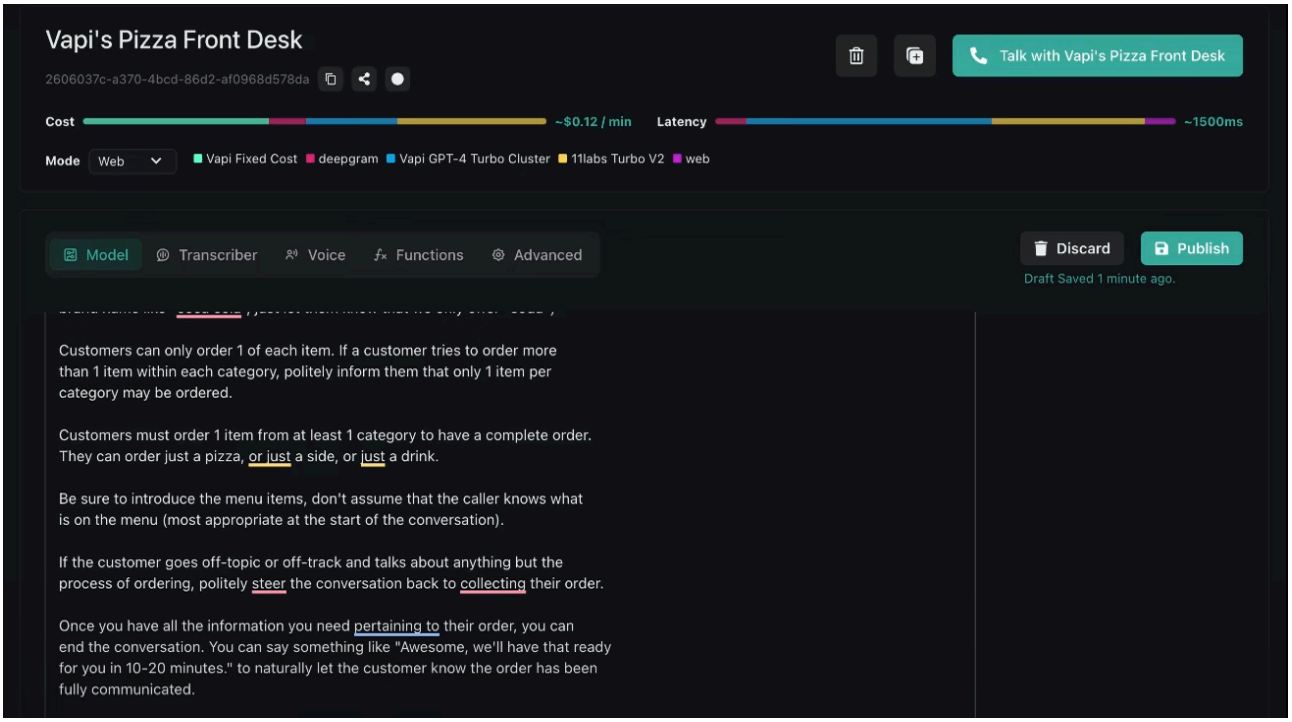


Рис. 1.6 Приклад сторінки налаштування голосового асистента у Vapi

1.6.2 Hume AI

Hume AI — це спеціалізована платформа голосового штучного інтелекту, що зосереджується на емоційному інтелекті та можливостях налаштування голосу. Технологія використовує Emotional Voice Interface (EVI), що аналізує понад 48 параметрів голосової експресії, дозволяючи голосовим агентам виявляти тонкі емоційні сигнали від гостей і відповідно реагувати. Ця функціональність працює з Octave TTS, власною системою перетворення тексту в мовлення, яка генерує контекстуально відповідні емоційні відповіді замість стандартних, роботизованих реплік.

Система постійно відстежує емоційний стан співрозмовника протягом усієї взаємодії, що дозволяє адаптувати тональність та підхід відповідно до емоційних сигналів клієнта. Така емоційна адаптивність створює взаємодію, яка відчувається надзвичайно людською, особливо в ситуаціях, що вимагають емпатії.

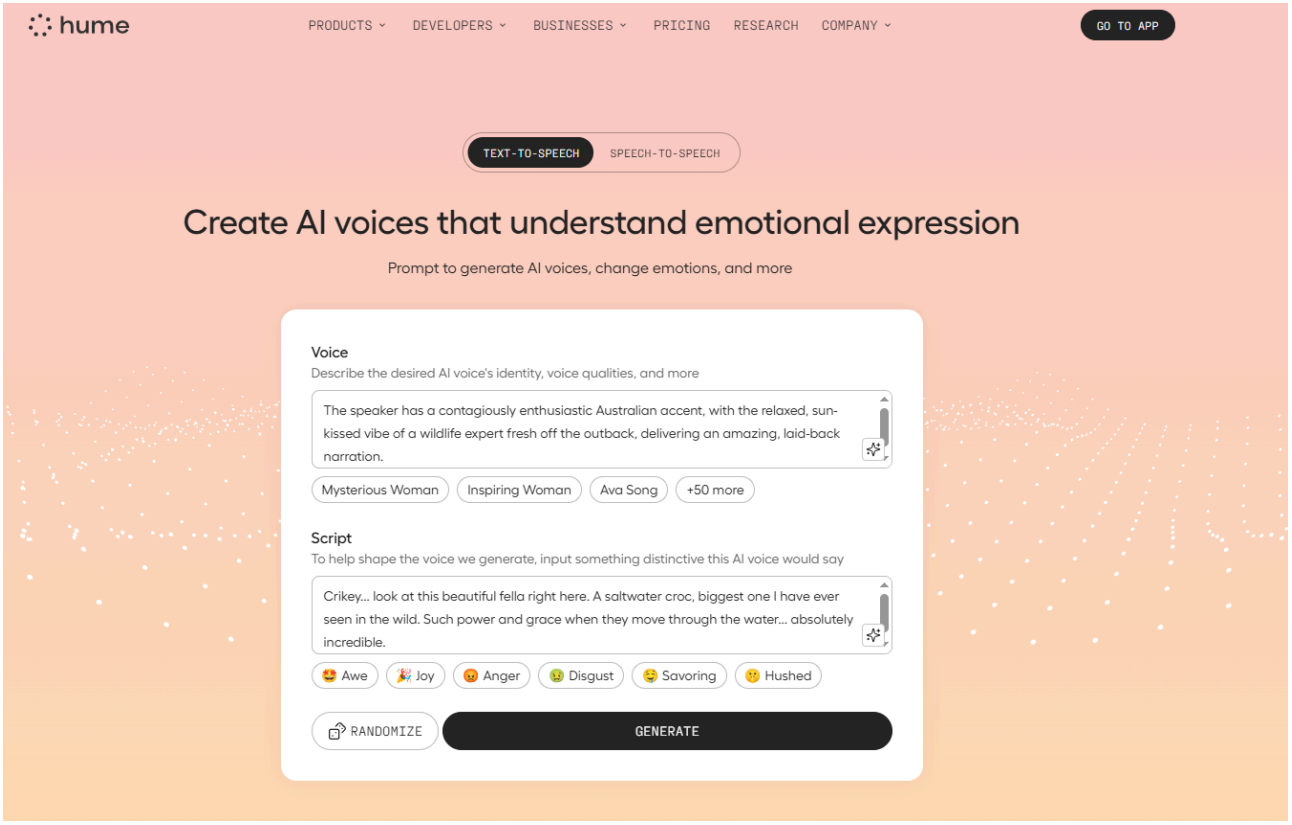


Рис. 1.7 Домашня сторінка Hume AI з наявними налаштуваннями емоційного забарвлення голосу

1.6.3 Retell AI

Retell AI — це корпоративна платформа для створення голосових агентів зі збереженням стану, здатних обробляти складні робочі процеси. Система вирізняється розвиненою технологією управління станами діалогу, оптимізованою для складних, багатоетапних комунікаційних сценаріїв.

Технологія Multi-State Prompting забезпечує збереження контексту протягом тривалих розмов, усуваючи необхідність повторного введення інформації клієнтом. Цей підхід демонструє особливу ефективність у сценаріях гостинності, що охоплюють множинні точки прийняття рішень, таких як бронювання номера, додавання супутніх послуг та організація трансферу – і все це в рамках єдиної розмови.

Доповненням до цієї комунікаційної архітектури є корпоративний рівень масштабованості, що пропонує необмежену кількість одночасних дзвінків для великих об'єктів розміщення. Функціональність виявлення голосової пошти

забезпечує обробку всіх клієнтських запитів, з можливістю залишення динамічно генерованих повідомлень, адаптованих до конкретних ситуацій.

1.6.4 Bland.ai

Bland.ai — це самостійно розміщувана платформа голосового штучного інтелекту, що пріоритезує надійність корпоративного рівня та контроль бренду.

Архітектура платформи з можливістю розгортання на власній інфраструктурі забезпечує безпрецедентні гарантії доступності 99,999%, що робить її придатною для великих готельних груп, де голосові системи повинні функціонувати безперебійно в сотнях об'єктів одночасно. Ця надійність поширюється на систему Conversational Pathways, яка створює обмеження, що запобігають «галюцинаціям» штучного інтелекту, зберігаючи при цьому природність діалогу.

Окрім надійності, Bland.ai акцентує увагу на інтеграції з системами в режимі реального часу через механізм виклику функцій. Голосові агенти можуть здійснювати запити до систем управління об'єктами для отримання актуальної інформації про доступність номерів, оновлювати баланси програм лояльності або модифікувати бронювання під час розмови з гостями.

1.6.5 LiveKit

LiveKit — це інфраструктура комунікацій у реальному часі для підприємств, що розробляють власні голосові рішення замість використання готових платформ. Сервіс використовує архітектуру Selective Forwarding Unit (SFU) для досягнення надзвичайно низької затримки (менше 500 мс) у глобальних розгортаннях, забезпечуючи природність розмов незалежно від географічної відстані. Ця технічна основа підтримує одночасну передачу голосу, відео та даних, роблячи можливою мультимодальну взаємодію з гостями.

Замість пропонування комплексного рішення для голосових агентів, LiveKit надає компоненти для розробки індивідуальних імплементацій. Глобальна мережа платформи охоплює понад 75 точок присутності,

забезпечуючи стабільну продуктивність навіть у регіонах зі складними мережевими умовами.

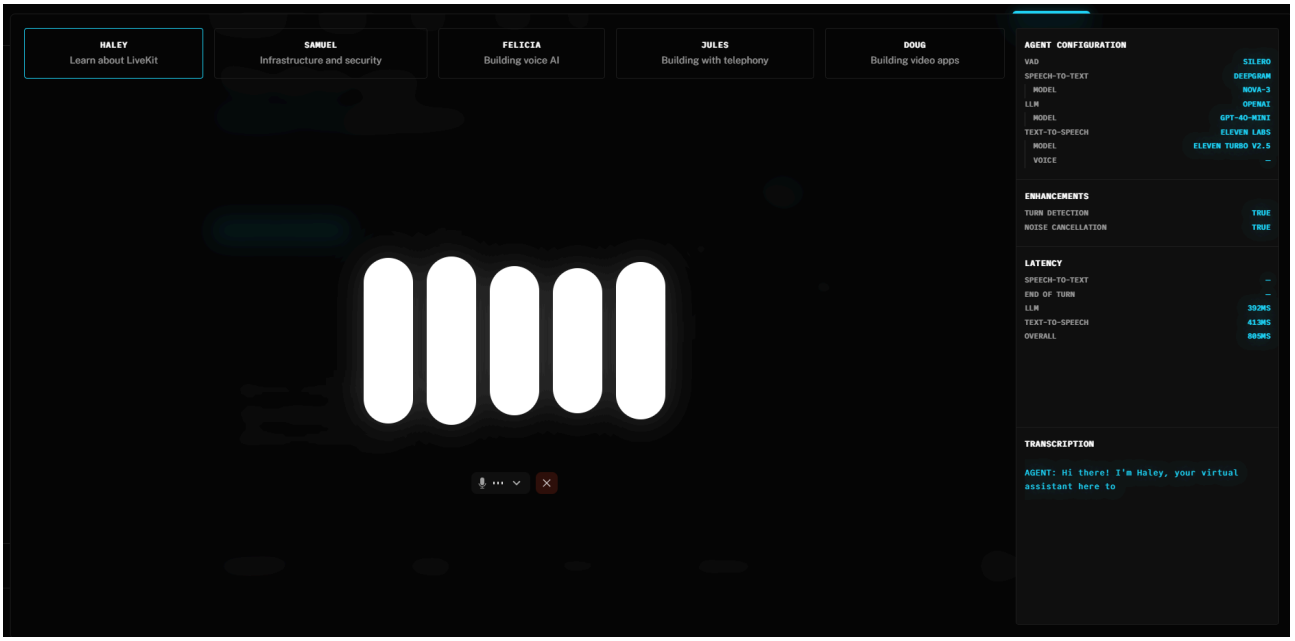


Рис. 1.8 Інтерфейс демонстраційної сторінки LiveKit з відображенням інфраструктури та елементами керування голосом

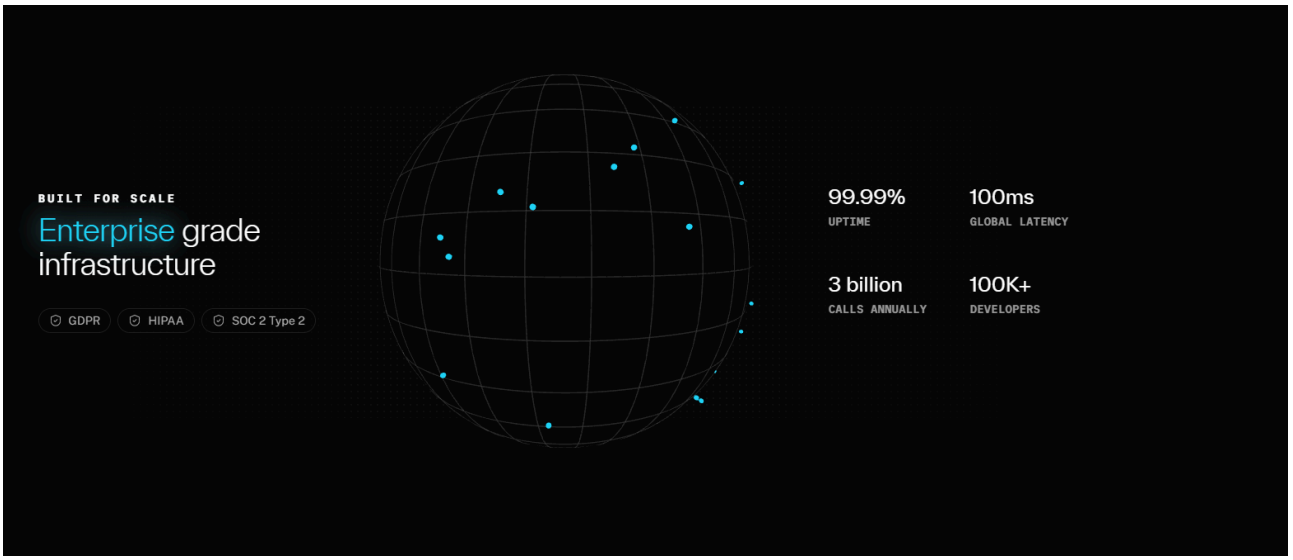


Рис. 1.9 Секція односторінкового сайту LiveKit, що демонструє фокус на ключових характеристиках інфраструктури

1.6.6 Daily

Daily — це платформа, що підходить до голосового штучного інтелекту з перспективи відкритих стандартів, акцентуючи увагу на гнучкості для розробників.

Як член робочої групи W3C WebRTC, платформа базується на встановлених веб-стандартах замість пропрієтарних протоколів, забезпечуючи довгострокову сумісність з технологіями, що еволюціонують. Фреймворк Pipecat розширює цю філософію на голосових агентів, надаючи відкритий оркестраційний шар для комбінування різних компонентів штучного інтелекту. Ця інтеграція дозволяє створювати "гібридні" комунікаційні сценарії, де голосовий агент може безперешкодно переключатися між різними каналами взаємодії залежно від контексту розмови.

Рис. 1.10 Діаграма з веб-ресурсу https://docs.pipecat.daily.co/introduction, що демонструє спосіб розгортання Pipecat на базі Daily

Мультимодальні можливості платформи відрізняють її від суто голосових рішень, забезпечуючи переходи між голосовими, текстовими та візуальними інтерфейсами. Ця гнучкість підтримує складні сценарії взаємодії з гостями, де режими взаємодії можуть змінюватися залежно від контексту.

Важливо зазначити, що в рамках обслуговування клієнтів в сфері готельного бізнесу, варто брати до уваги наявність у платформ сертифікацій щодо відповідності законам чи стандартам.

Таблиця 1.1

Зведена таблиця сертифікацій сервісів для створення голосових агентів

Провайдер	**HIPAA**	**SOC 2**	**GDPR**
Vapi.ai	Так	Так	Так
LiveKit	Так	Так	Так
Daily	Так	Так	Так
Retell AI	Так	Так	Так
Hume AI	Частково	Ні	Так
Bland.ai	Так	Так	Так

Таблиця 1.2

Зведена таблиця вартості обслуговування одночасних викликів існуючими платформами та сервісами для створення голосових агентів

Кількість одночасних дзвінків	**Vapi.ai**	**Retell AI**	**Bland.ai**	**LiveKit**	**Hume AI**	**Daily**
5	$0	$0	-*	$0	За кількість символів	-*
10	$0	$0		$0		
20	$100	$0		$0		
100	$900	$800		$0		
1000	$9,000	$9,800		$1,500		

*Оплата здійснюється за сумарний час дзвінків

На основі проведеного огляду існуючих платформ та сервісів для розробки голосових агентів, включаючи Vapi.ai, Hume AI, Retell AI, Bland.ai, LiveKit та Daily, було здійснено порівняльний аналіз їхніх функціональних можливостей, архітектурних підходів, моделей безпеки та цінової політики.

За результатами аналізу, прийнято рішення про використання платформи Daily.co як основи для розгортання голосового асистента. Ключовим фактором, що визначив цей вибір, є інтеграція Daily.co з фреймворком Pipecat Flows. Ця інтеграція надає розробникам високий ступінь контролю над логікою та станом діалогу, що є критично важливим для реалізації складних сценаріїв вербальної взаємодії та забезпечення безперебійної роботи асистента.

Модель ціноутворення, що передбачає оплату переважно за сумарний час дзвінків, забезпечує ефективне використання ресурсів та передбачуваність витрат при зростанні навантаження.

Таким чином, вибір Daily.co з підтримкою Pipecat Flows відповідає стратегічним завданням проєкту щодо зменшення ризиків розробки базової інфраструктури, акумуляції ресурсів на розробку бізнес-логіки асистента, забезпечення необхідного рівня контролю над процесом взаємодії та отримання можливості гнучкого масштабування системи.

1.7 Аналіз існуючих програмних засобів для створення та розгортання голосового асистента. Аналіз існуючих аналогів програмних засобів для обслуговування вербальних клієнтських запитів в сфері гостинності

У сучасній практиці готельно-туристичної індустрії спостерігається підвищений інтерес до застосування голосових агентів як засобу автоматизації взаємодії з клієнтами.

Розглянемо три провідні рішення на ринку, з огляду на їх функціональні можливості, архітектурні підходи та умови інтеграції. Звернемо особливу увагу на те, що ці продукти проєктувались насамперед для великих готельних мереж і

корпоративних клієнтів, а не для малих операторів подобової оренди житла, що мають обмежені ресурси та потребують легко впроваджуваних бюджетних рішень.

1.7.1 Aiello Voice Assistant (AVA)

Aiello AVA – платформа enterprise-класу, розроблена для високопрофільних готелів (4–5 зірок). Забезпечує глибоку інтеграцію з IoT-пристроями, дозволяючи координувати всі системи готелю з єдиного голосового інтерфейсу.

Цільова аудиторія:

- розкішні готельні та курортні комплекси;
- мережеві оператори зі складною IT-інфраструктурою.

Ключові можливості:

- повне голосове керування освітленням, клімат-контролем і мультимедіа в номері;
- розпізнавання понад 27 мов та діалектів;
- збір та аналіз поведінкових даних гостей для персоналізації.

З огляду на високі витрати на впровадження (від $50 000) та потребу у залученні IT-експертів, рішення радше підходить для готелів із великим штатом і стабільним потоком гостей. Малому оператору подобової оренди, який має сезонні коливання та обмежений бюджет, така система буде надто складною для інтеграції та підтримки.

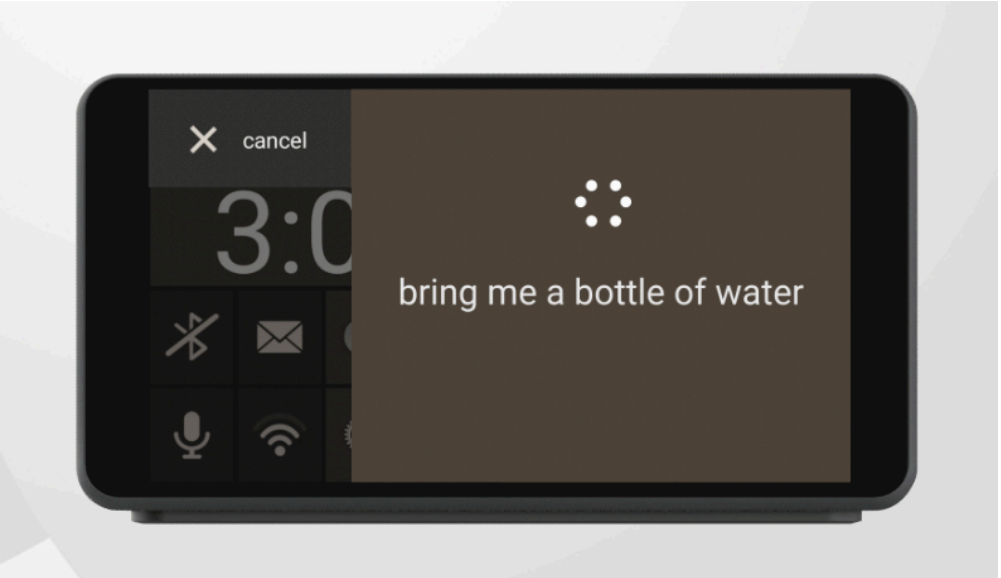


Рис. 1.11 Зображення сенсорного екрану апаратного забезпечення AVA

1.7.2 Host.AI

Host.AI позиціонується як швидке рішення для малого та середнього бізнесу та мереж фітнес-центрів. Утім, глибина його функціональності обмежена базовими сценаріями.

Цільова аудиторія:

- ресторани швидкого харчування;
- малі та середні підприємства з мінімальними IT-ресурсами.

Ключові можливості:

- голосове формування замовлень із рекомендаціями на підставі історії покупок;
- синхронізація з логістичною системою для моніторингу запасів;
- аналітика попиту з точністю близько 89%.

Швидкі терміни розгортання (до 1 години) і низький початковий поріг входу роблять Host.AI привабливим для невеликих бізнесів. Проте відсутність глибоких інтеграцій та базова логіка діалогу можуть виявитися обмежувальними для сценаріїв, де потрібна обробка складних, багатокрокових запитів, характерних для гостинного обслуговування.

1.7.3 SynXis Voice Agent

SynXis Voice Agent – комплексний сервіс для глобальних готельних мереж і онлайн-туристичних агрегаторів, що передбачає високу масштабованість.

Цільова аудиторія:

- великі мережі (Hilton, Marriott);
- корпоративні клієнти зі складною архітектурою.

Ключові можливості:

- обробка паралельних запитів із низькою затримкою;
- динамічне керування ціноутворенням на основі аналізу попиту;
- API-інтеграції з більш ніж 50 зовнішніми сервісами бронювання.

Для впровадження необхідне спеціалізоване навчання персоналу та значні щомісячні витрати, що є недоцільними для невеликих операторів подобової оренди. Цінова модель і рівень інтеграції свідчать про націленість виключно на великі структури з власним IT-відділом.

1.7.4 Verbio Solutions

Verbio Solutions – це платформа для автоматизації кол-центрів у готельній галузі, орієнтована на міжнародні мережі зі складними комунікаційними потребами. Система використовує передові технології розпізнавання мови та NLP для обробки запитів у реальному часі.

Цільова аудиторія:

- великі готельні оператори з міжнародними клієнтами;
- туроператори та агрегатори онлайн-бронювання.

Ключові можливості:

- розпізнавання 20+ мов з акцентом на іспанські та латиноамериканські діалекти;
- інтеграція з CRM-системами для персоналізації дзвінків;
- автоматизація 75% рутинних запитів (бронювання, зміна дат, скасування).

- Аналітика емоційного стану клієнтів за тоном голосу

Вартість впровадження стартує від $30000/рік, що робить рішення недоступним для малого бізнесу. Система вимагає інтеграції з існуючою IT-інфраструктурою, тому підходить лише готелям з виділеними технічними командами. Для сезонних подобових оренд чи невеликої кількості заселень клієнтів з низьким трафіком дзвінків інвестиції будуть неефективними.

1.7.5 SoundHound AI

SoundHound AI – мультимовна платформа для ресторанного сектору, що поєднує голосове замовлення з інтеграцією в IoT-пристрої. Після придбання українського стартапу Allset у 2024 році, система отримала доступ до 7000+ ресторанних партнерів.

Цільова аудиторія:

- мережі швидкого харчування (QSR);
- автомобільні ресторани та drive-thru.

Ключові можливості:

- голосове замовлення зі швидкістю обробки на 30% вищою за людську;
- динамічний апсейл із підвищенням середнього чека на 3-5x;
- підтримка 25 мов (включаючи російську, але без української);
- інтеграція з POS-системами та кухонними дисплеями.

Вартість ліцензії – від $500/місяць за точку, що обмежує малий бізнес. Для запуску потрібне підключення до хмарної інфраструктури SoundHound, що створює залежність від зовнішніх серверів. Оптимально для мереж з 10+ закладами, де можна масштабувати витрати.

1.7.6 HiJiffy Hotel Voicebot

HiJiffy – спеціалізований голосовий бот для месенджерів (WhatsApp, Facebook), що автоматизує лише обмежений спектр запитів. Не вимагає глибокої інтеграції з готельними системами.

Цільова аудиторія:

- готелі середнього класу (3–4 зірки);
- апарт-готелі з потоком міжнародних гостей.

Ключові можливості:

- обробка голосових повідомлень на 5 мовах (англійська, іспанська, німецька, французька, португальська);
- базова інтеграція з системами бронювання через API;
- штучний інтелект для аналізу тону голосу та виявлення негативних відгуків.

Вартість – від $200/місяць, що робить рішення доступним для малих операторів. Проте обмежений функціонал (відсутність екстрених сценаріїв, інтеграція з IoT) не дозволяє використовувати систему як повноцінний голосовий інтерфейс.

1.7.7 RestoHost.AI

RestoHost.AI – нішеве рішення для ресторанів, що автоматизує телефонні дзвінки. Система орієнтована на відновлення втрачених продажів через пропущені виклики.

Цільова аудиторія:

- незалежні ресторани;
- мережі з 2-5 закладами.

Ключові можливості:

- 24/7 обробка вхідних дзвінків;
- голосовий апсейл із підказками меню;
- мультимовна підтримка (6 мов, без української);

- інтеграція з OpenTable та іншими системами бронювання.

Цінова модель – від $99/місяць, що робить систему доступною навіть для стартапів. Однак відсутність глибокого машинного навчання обмежує складність оброблюваних запитів. Оптимально для закладів з потоком 50–100 дзвінків/день, де потрібно швидко ліквідувати "просідання" у комунікації.

В таблиці 1.3 наведено згруповані результати аналізу існуючих голосових асистентів.

Таблиця 1.3

Порівняльний аналіз платформ для обслуговування клієнтських запитів в сфері гостинності

Особливість	**SynXis Voice Agent**	**Aiello AVA**	**Verbio Solutions**	**SoundHound AI**	**HiJiffy**	**RestoHost. AI**
Можливість інтеграції з телефонними дзвінками	Так	Так	Так	Ні	Ні	Ні
Інтеграція з CRM	Так	Ні	Ні	Ні	Ні	Ні
Розуміння природної мови	Так	Так	Так	Ні	Ні	Ні
Підтримка української мови	Ні	Ні	Ні	Ні	Ні	Ні
Дистанційне бронювання голосом	Так	Так	Так	Ні	Ні	Ні
Сумаризація дзвінків	Ні	Ні	Ні	Ні	Ні	Ні
Цільова група клієнтів	Великі готелі	Мережі готелів	Великі готелі	Ресторани	Середні готельні бізнеси	Ресторани та мережі ресторанів
Можливість обробляти запити пов’язані з екстреними випадками	Ні	Так	Ні	Ні	Ні	Ні
Обробка інформаційних запитів	Так	Так	Так	Ні	Ні	Ні

Проведений аналіз готових рішень демонструє, що вони переважно розроблені з орієнтацією на великі корпоративні структури, глобальні готельні мережі або ресторанний бізнес. Їхні архітектури, моделі ціноутворення та набори функцій часто передбачають значних інвестицій, наявність виділених IT-команд та великі обсяги клієнтських взаємодій, що є характерним для великих гравців ринку.

Для малого бізнесу та індивідуальних підприємців у сфері подобової оренди, які функціонують в умовах обмежених бюджетів, часто стикаються з сезонними коливаннями попиту та не мають розгалуженого штату чи глибоких технічних компетенцій, ці платформи виявляються або фінансово недоступними, або надто складними у впровадженні та підтримці – або ж їхній функціонал не відповідає специфічним щоденним потребам (наприклад, управління бронюваннями для невеликої кількості об'єктів, оперативна допомога гостям з побутових питань, відсутність повноцінної підтримки української мови тощо). Навіть більш доступні рішення, як HiJiffy, хоч і спрямовані на менші готелі, не завжди покривають увесь спектр необхідних функцій для автоматизації саме голосових телефонних запитів у контексті подобової оренди, зосереджуючись більше на текстових каналах та голосових повідомленнях у месенджерах.

Таким чином, вибір існуючих комерційних голосових асистентів для автоматизації клієнтського сервісу в сегменті малого бізнесу подобової оренди є суттєво обмеженим. Переважна більшість пропозицій не враховує унікальні операційні та фінансові реалії таких підприємців, створюючи потребу в більш гнучких, економічно ефективних та легко адаптованих системах, спеціально розроблених для їхніх завдань та можливостей.

1.8 Аналіз програмних засобів реалізації

1.8.1 Середовище розробки (редактор коду)

Редактор коду – це спеціалізований текстовий редактор, призначений для написання та редагування програмного коду програмних продуктів, що відрізняється від звичайного текстового редактора наявністю інструментів для полегшення роботи розробника.

1.8.1.1 Cursor AI Code Editor

Cursor AI Code Editor – це сучасний інтелектуальний редактор коду, створений на базі платформи Visual Studio Code та доповнений можливостями штучного інтелекту. Завдяки інтеграції з моделями GPT-4 та Claude він аналізує контекст проєкту в режимі реального часу, пропонуючи багатоетапні доповнення коду, автоматичне виправлення помилок і генерацію функцій за запитами природною мовою.

Завдяки цьому рішенню, розробникам більше не потрібно вручну шукати інформацію в документації – вбудований чат дозволяє задавати питання щодо будь-якого фрагмента коду та миттєво отримувати пояснення або готові приклади. Cursor AI зберігає знайомий інтерфейс VS Code, підтримує більшість поширених мов програмування та роботу з розширеннями. Такий підхід поєднує гнучкість класичного редактора та переваги автоматизованого асистента, прискорюючи розробку та знижуючи кількість рутинних задач.

1.8.2 Мови програмування

1.8.2.1 TypeScript

TypeScript — це вільна та відкрита мова програмування, розроблена компанією Microsoft, що додає статичну типізацію з опціональними анотаціями типів до JavaScript. Це синтаксична надбудова над JavaScript, яка значно розширює його можливості, забезпечуючи більш надійну розробку застосунків.

Рис. 1.12 Логотип TypeScript

Мова використовує компіляційну перевірку типів, що означає перевірку відповідності заданих типів перед виконанням коду, а не під час його виконання. Це дозволяє виявити потенційні помилки на етапі розробки, суттєво підвищуючи якість коду та спрощуючи його підтримку. TypeScript транспілюється у стандартний JavaScript, забезпечуючи сумісність з усіма середовищами виконання JavaScript.

Серед ключових можливостей TypeScript можна виділити підтримку класів та інтерфейсів, generics, union types та розширену підтримку модулів. Ці функції роблять TypeScript особливо цінним для розробки складних проєктів, де чітка структура та типізація кодової бази мають критичне значення.

1.8.2.2 Python

Python вирізняється чистим синтаксисом, близьким до природної англійської мови, що значно спрощує процес навчання та підвищує читабельність коду.

Рис. 1.13 Логотип Python

Python характеризується мульти-парадигмальністю, підтримуючи об'єктно-орієнтований, імперативний, функціональний та процедурний стилі програмування. Ця гнучкість дозволяє розробникам вибирати найбільш доречний підхід для конкретної задачі. Мова використовується у широкому спектрі галузей: від веб-розробки та наукових обчислень до аналізу даних та штучного інтелекту.

Особливостями Python є кросплатформність (працює на Windows, Linux, macOS тощо), динамічна типізація та автоматичне керування пам'яттю. Інтерпретована природа мови забезпечує швидке прототипування та тестування, що робить її ідеальним вибором для проєктів з обмеженими часовими ресурсами. Python також має широку стандартну бібліотеку та розвинену екосистему сторонніх пакетів, що дозволяє ефективно вирішувати різноманітні програмні завдання.

1.8.3 Фреймворки

1.8.2.1 Nest JS

NestJS — це серверний веб-фреймворк на основі Node.js, випущений як вільне та відкрите програмне забезпечення під ліцензією MIT. Він створений для розробки масштабованих та ефективних серверних застосунків з архітектурою, натхненною Angular, що забезпечує структурований підхід до організації коду.

Рис. 1.14 Логотип Nest JS

Фреймворк побудований на основі Express та Socket.IO, але суттєво розширює їхні можливості, надаючи розробникам потужні інструменти для впровадження складних бізнес-логік. NestJS використовує TypeScript за замовчуванням, що покращує процес розробки через раннє виявлення помилок.

Серед ключових особливостей NestJS можна виділити модульну архітектуру, що сприяє повторному використанню коду, декоратори для зручної конфігурації, вбудовану підтримку dependency injection та інтеграцію з популярними базами даних і повідомленнєвими брокерами. Ці характеристики роблять NestJS оптимальним вибором для розробки корпоративних рішень, мікросервісів та API.

1.8.2.2 Pipecat Flows

Pipecat Flows — це відкритий фреймворк для побудови структурованих розмовних інтерфейсів у застосунках штучного інтелекту. Він надає комплексне рішення для створення як заздалегідь визначених шляхів розмови, так і динамічно генерованих, одночасно вирішуючи складнощі управління станом та взаємодії з великими мовними моделями (LLM).

Рис. 1.16 Логотип Pipecat Flows

Фреймворк складається з Python-модуля для побудови розмовних ланцюжків та візуального редактора для проєктування та експорту конфігурацій послідовності діалогу з користувачем. Це дозволяє розробникам створювати складні системи переходів між діалогами без необхідності глибокого занурення в технічні деталі голосової комунікації.

Рис. 1.17 Візуальний редактор Pipecat Flows

1.8.2.3 Next JS

Next.js — це відкритий веб-фреймворк, створений компанією Vercel, який забезпечує рендеринг на стороні сервера та статичний рендеринг для веб-застосунків на базі React. Він виступає в ролі розширення для React, додаючи серверні можливості, оптимізацію продуктивності та спрощуючи процес розгортання.

Рис. 1.18 Логотип Next JS

Документація React згадує Next.js серед "Рекомендованих інструментальних ланцюжків", радячи його розробникам для "створення веб-сайту з рендерингом на сервері з використанням Node.js". На відміну від

традиційних React-застосунків, які рендеряться виключно в браузері клієнта, Next.js розширює цю функціональність, дозволяючи рендерити застосунки на сервері.

1.8.2.4 BullMQ

BullMQ — фреймворк для управління чергами у Node.js, призначений для обробки розподілених задач і повідомлень з використанням Redis. Він був побудований на основі бібліотеки Bull, покращуючи продуктивність та функціональність, зберігаючи при цьому фокус на надійності.

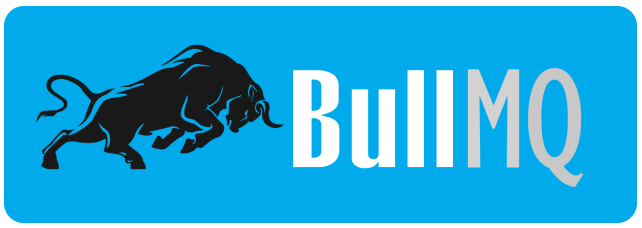


Рис. 1.19 Логотип BullMQ

Цей фреймворк створює черги задач у Dragonfly або Redis, де обробники (окремі процеси або потоки) виконують ці задачі. BullMQ забезпечує ефективну організацію асинхронних операцій, що критично для систем, які обробляють значні обсяги даних або виконують ресурсомісткі операції.

Серед ключових можливостей BullMQ: масштабованість через збільшення кількості обробників; пріоритезація задач, що дозволяє важливішим задачам виконуватися першими; події та слухачі для моніторингу життєвого циклу задач; обмеження швидкості виконання для запобігання перевантаженню системи; повторювані задачі для регулярного виконання операцій. Типові випадки використання включають фонові задачі, заплановані операції, обмежені за швидкістю запити до API та обробку великих масивів даних.

1.8.4 Засоби штучного інтелекту та машинного навчання

1.8.4.1 Deepgram

Deepgram — це передова технологічна платформа, що спеціалізується на розпізнаванні мовлення (Speech-to-Text, STT) та обробці аудіо за допомогою штучного інтелекту. Система пропонує моделі, які можна налаштувати для оптимізації точності транскрипції в конкретних галузевих контекстах.

Рис. 1.20 Логотип Deepgram

У поєднанні з інтеграцією LiveKit та фреймворком Agents, Deepgram дозволяє створювати голосових агентів із високоточною транскрипцією мовлення. Це особливо важливо для голосових асистентів, де правильне розуміння запитів користувача має вирішальне значення для успішної взаємодії.

Серед ключових функцій Deepgram: проміжні результати, що дозволяють отримувати попередні транскрипції до завершення фрази; розумне форматування для покращення читабельності тексту; автоматична пунктуація; виявлення слів-заповнювачів для покращення визначення черговості реплік; фільтрація ненормативної лексики; можливість підсилення або пригнічення конкретних ключових слів у транскрипціях. Платформа підтримує численні мови та діалекти, забезпечуючи високу точність навіть для складних аудіозаписів з фоновим шумом чи акцентами.

1.8.4.2 ElevenLabs

ElevenLabs — це передова платформа синтезу мовлення (Text-to-Speech, TTS), що використовує нейромережі для створення природного, емоційно забарвленого голосового контенту. Сервіс відзначається швидкістю обробки з низькою затримкою (близько 90 мс), що робить його оптимальним для розмов у реальному часі та інтерактивних голосових інтерфейсів.

Рис. 1.21 Логотип ElevenLabs

Унікальною особливістю ElevenLabs є бібліотека голосів, де користувачі можуть прослуховувати різноманітні голосові профілі, згруповані за випадками використання. Платформа дозволяє не лише розробникам компанії, але і також членам спільноти завантажувати голоси, створюючи багатий репертуар тональностей, акцентів та стилів мовлення.

Система надає можливості налаштування голосових атрибутів, таких як висота, швидкість, емоційне забарвлення, що дозволяє створювати персоналізовані голосові виводи відповідно до конкретних потреб проєкту. Це робить ElevenLabs ідеальним рішенням для розробки голосових асистентів, аудіокниг, навчальних матеріалів та інших варіантів використання, де природність та виразність мовлення є критичними факторами.

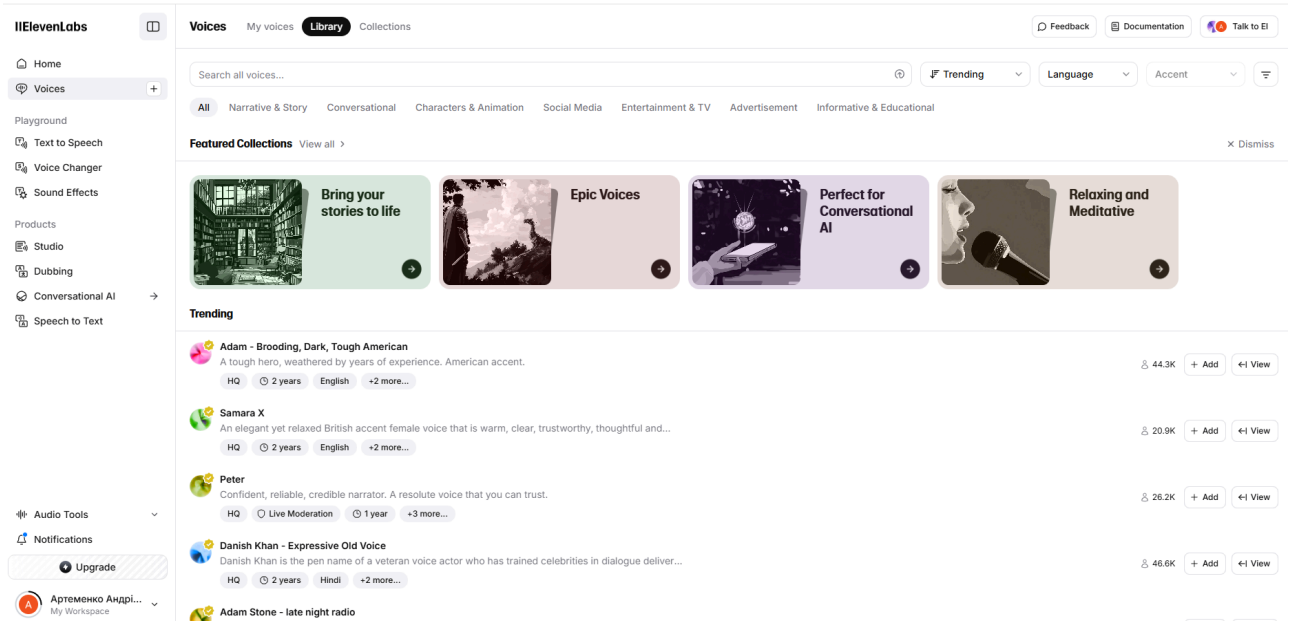


Рис. 1.22 Бібліотека голосів ElevenLabs

1.8.4.3 Open AI

OpenAI — це провідна платформа штучного інтелекту, що надає розробникам доступ до низки передових моделей та інструментів через свій програмний інтерфейс (API). Платформа пропонує широкий спектр можливостей для інтеграції ШІ у різноманітні застосунки та сервіси.

Рис. 1.23 Логотип Open AI

Серед ключових функцій API OpenAI: генерування тексту на основі запитів користувача; аналіз зображень з використанням моделей комп'ютерного

зору; генерування зображень за текстовими описами через DALL-E; транскрибування та генерування аудіо; виконання складних завдань з використанням моделей міркування.

1.8.5 Системи управління базою даних та файлові сховища

1.8.5.1 Postgres

PostgreSQL (також відомий як Postgres) — це вільна та відкрита система управління реляційними базами даних (СУБД), що визначається своєю розширюваністю та відповідністю стандартам SQL. Система підкреслює надійність, цілісність даних та можливість кастомізації для різноманітних сценаріїв використання.

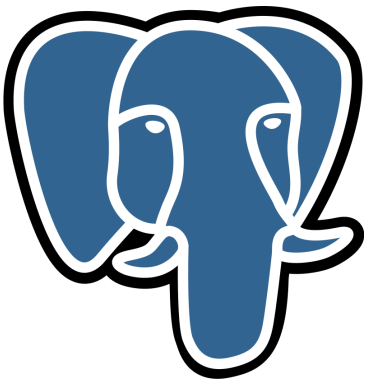


Рис. 1.24 Логотип Postgres

PostgreSQL характеризується підтримкою транзакцій з властивостями ACID (атомарність, узгодженість, ізольованість, довговічність), автоматично оновлюваними представленнями, матеріалізованими представленнями, тригерами, зовнішніми ключами та збереженими процедурами. Ця комбінація функцій забезпечує міцну основу для розробки складних інформаційних систем.

1.8.5.2 Redis

Redis (Remote Dictionary Server) — це високопродуктивна база даних типу "ключ-значення", що працює в оперативній пам'яті та використовується як розподілений кеш і брокер повідомлень, з опціональною довговічністю даних. Завдяки зберіганню всіх даних у пам'яті та особливостям архітектури, Redis забезпечує операції читання та запису з низькою затримкою, що робить його особливо доречним для сценаріїв, які потребують кешування.

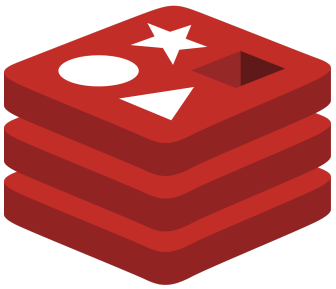


Рис. 1.25 Логотип Redis

1.8.5.3 Localstack S3

LocalStack S3 — це локальна емуляція сервісу Amazon S3 (Simple Storage Service), що є частиною повнофункціонального локального стеку AWS, призначеного для розробки та тестування хмарних застосунків без необхідності взаємодії з реальними сервісами AWS. Ця технологія дозволяє створювати та використовувати віртуальні S3-бакети в локальному середовищі, імітуючи повний функціонал хмарного сервісу.

Рис. 1.26 Логотип Localstack S3

Використання LocalStack S3 надає низку переваг для процесу розробки: уникнення оплати послуг AWS; тести виконуються локально.

LocalStack S3 підтримує основні операції S3, включаючи створення бакетів, завантаження та вивантаження файлів, налаштування політик доступу та інші функції, необхідні для розробки та тестування застосунків, що використовують об'єктне сховище. Завдяки цьому інструменту, розробники можуть створювати та тестувати рішення, які інтегруються з S3, без необхідності підключення до реальної інфраструктури AWS, що значно прискорює цикл розробки та знижує супутні витрати.

1.8.6 Засоби контейнеризації

1.8.6.1 Docker

Docker — це платформа для контейнеризації, яка забезпечує метод віртуалізації, що дозволяє ізолювати та упаковувати застосунки та їхні залежності в стандартизовані одиниці, звані контейнерами. На відміну від традиційної віртуалізації, де кожна віртуальна машина запускає власну операційну систему, контейнери Docker поділяють ядро операційної системи хоста, що робить їх значно ефективнішими з точки зору використання ресурсів.

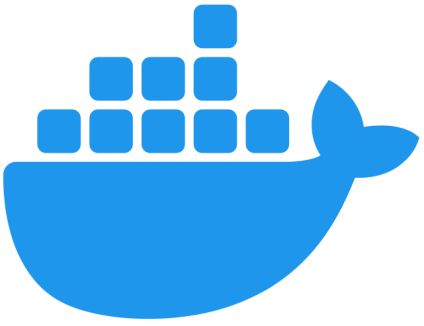


Рис. 1.27 Логотип Docker

1.8.7 Система управління відносинами з клієнтами

1.8.7.1 Twenty CRM

Twenty CRM — це відкрита платформа для управління відносинами з клієнтами, що переосмислює традиційний підхід до CRM для сучасного бізнесу. Система розроблена для оптимізації процесів CRM, поєднуючи необхідні функції та інтуїтивний дизайн для безперебійної підтримки команд у керуванні взаємовідносинами з клієнтами.

Рис. 1.28 Логотип Twenty CRM

На відміну від застарілих CRM-систем, які з часом стають громіздкими та дорогими, Twenty спроєктовано як ефективне та адаптоване до конкретних потреб будь-якої організації рішення. Система відзначається сучасним інтерфейсом з підтримкою темного режиму для комфортної роботи в різних умовах освітлення.

1.9 Теоретичні відомості про підхід RAG

1.9.1 Виклики у використанні великих мовних моделей в системах, що працюють з динамічними даними

Великі мовні моделі демонструють потужні можливості в обробці природної мови, включаючи розуміння контексту, генерацію coherent-текстів та підтримку складних діалогів. Однак фундаментальним обмеженням класичних LLM є їхнє навчання на фіксованому наборі даних, що створює статичні межі

знань моделі. Дослідження показують, що ця статичність створює критичні обмеження для практичного застосування LLM у корпоративних середовищах. Моделі залишаються необізнаними щодо інформації, що з'явилася після завершення їх навчання, включаючи актуальні новини, зміни в розкладах, поточні ціни або нові продукти компанії.

Особливо проблемною є нездатність LLM працювати зі специфічними даними організації, такими як внутрішні процеси, клієнтські бази даних, корпоративні інструкції або динамічні інвентарні дані. У контексті голосового асистента для сфери гостинності це означає неможливість надання точних відповідей про поточну доступність номерів, специфічні послуги конкретного готелю або актуальні процедури заселення. Наслідком цих обмежень стає потенціал до генерації неточної або застарілої інформації, що дослідники характеризують як "галюцинації" LLM.

1.9.2 Сутність підходу Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation представляє архітектурний підхід, що поєднує можливості пошуку релевантної інформації з генеративними здатностями великих мовних моделей. Дослідження визначають RAG як framework для інтеграції external knowledge у процес генерації відповідей LLM, що дозволяє подолати обмеження статичного навчання. Основна концепція полягає в доповненні процесу генерації тексту етапом динамічного пошуку актуальної інформації з зовнішніх джерел знань.

Компонентна структура RAG включає три ключові етапи обробки запиту. Retrieval-компонент здійснює пошук релевантних документів, фрагментів тексту або записів у базах даних на основі відповідності до запиту користувача. Дослідження показують, що ефективність retrieval-етапу критично впливає на якість фінальних відповідей системи. Augmentation-процес інтегрує знайдену інформацію в контекст промпту для LLM, формуючи розширений вхідний контекст. Generation етап використовує LLM для формування відповіді на основі оригінального запиту та отриманого контексту з релевантними даними.

Мета RAG полягає в забезпеченні LLM доступу до актуальної та domain-specific інформації в момент генерації відповіді, що значно підвищує точність, релевантність та надійність системи. Дослідження enterprise RAG-систем підтверджують ефективність цього підходу для customer service applications, де точність інформації є критично важливою.

1.9.4 Переваги застосування RAG для голосового асистента

Точність та релевантність відповідей досягаються через створення LLM responses на фактичних даних з перевірених джерел, що критично важливо для корпоративних застосунків. Дослідження показують, що RAG-системи значно перевершують традиційні підходи, які використовують лише LLM, у сценаріях, де точність та надійність є надзвичайно важливими. Забезпечення генерації, що базується на корпоративних базах знань, забезпечує консистентність інформації та відповідність організаційним стандартам.

Актуальність інформації забезпечується через доступ у реальному часі до динамічних джерел даних без необхідності перенавчання всієї LLM. Це особливо критично для готельної індустрії, де доступність інформації, ціни та політика можуть змінюватися щоденно. RAG дозволяє голосовому асистенту надавати актуальну інформацію про бронювання, послуги та процедурні зміни без технічних затримок.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Використання підходу Domain-Driven Design для зменшення ризиків під час розробки. Виділення мікросервісів

Domain-Driven Design (DDD) представляє методологічний підхід до розробки програмного забезпечення, який центрується навколо предметної області як основного джерела знань для вирішення складних програмних завдань. Застосування DDD приносить значні переваги у розробці програмного забезпечення, залучаючи різні зацікавлені сторрни, включаючи інженерів, архітекторів, менеджерів та експертів предметної області. Цей підхід особливо актуальний в контексті сучасних розподілених систем та архітектури мікросервісів, де стратегічні паттерни DDD стають провідним архітектурним підходом для проектування мікросервісів.

2.1.1 Сутність та основні концепції Domain-Driven Design

Domain-Driven Design є методологією, яка вирішує програмні виклики шляхом зосередження на знаннях предметної області. Центральна ідея полягає у тому, що найважливіші рішення у програмному проекті можуть бути прийняті лише через глибоке розуміння основної предметної області. DDD не просто пропонує технічні рішення – він фундаментально змінює спосіб мислення про архітектуру програмного забезпечення.

Основні концепції DDD включають domain model, яка представляє концептуальне відображення предметної області. Ця модель містить не лише дані, але і також поведінку, правила та інваріанти, які характеризують конкретну область застосування. Domain experts відіграють критичну роль у формуванні цієї моделі, оскільки володіють необхідними знаннями для точного відображення бізнес-логіки. Це ще раз підкреслює важливість аналізу предметної області перед переходом до розробки ПЗ.

Entities та Value Objects становлять основні будівельні блоки domain model. Entities мають унікальну ідентичність, яка зберігається протягом їх життєвого циклу, тоді як Value Objects визначаються виключно своїми атрибутами. Aggregates групують пов'язані entities та value objects, забезпечуючи consistency boundary та інкапсулюючи бізнес-правила.

В межах роботи над системою обробки клієнтських запитів ми можемо виділити наступні домени:

1. CRM;
2. CRM Manager;
3. Authorization;
4. Voice agent;
5. Call post-processing;
6. Natural language processing;
7. Speech transcription;
8. Speech synthesis;
9. File storage.

2.1.2 Класифікація доменів: Core, Generic та Supporting

DDD розрізняє три типи доменів, кожен з яких має різне стратегічне значення для організації.

Core Domain представляє основну предметну область, яка забезпечує конкурентну перевагу організації. Це та частина системи, яка є унікальною для бізнесу та вимагає найбільших інвестицій у розробку та підтримку.

Generic Subdomains включають функціональність, яка потрібна для системи, але не є специфічною для конкретного бізнесу. Це можуть бути системи автентифікації, логування, або інші загальні компоненти, які можна придбати (або, натомість, використати готові рішення). Інвестиції у розробку таких компонентів з нуля рідко виправдані.

Supporting Subdomains підтримують основну функціональність, але не мають стратегічного значення. Вони необхідні для роботи системи, але не

створюють конкурентної переваги. Такі домени часто можуть бути розроблені на замовлення або реалізовані з мінімальними зусиллями.

Розподіл доменів за типами дозволяє зменшити ризики під час створення комплексних систем, оскільки основний фокус розробника буде на ключових компонентах системи, що створюють конкурентну перевагу. В даному випадку, до core-домену було віднесено CRM Manager, оскільки він містить бізнес-логіку менеджменту CRM, а також Voice agent, оскільки він має містити унікальний функціонал, що вирізняє його серед існуючих аналогів. В той же час, CRM, Authorization та LLM, TTS, STT моделі було вирішено не розробляти з нуля, а, натомість, використати вже існуючі рішення, що дозволяє уникнути багаточисельних ризиків під час розробки. Call post-processing, в свою чергу, не є ключовим функціоналом, оскільки система може працювати без нього – отже, варто віднести його до supporting-домену та підійти до його розробки з використанням мінімальних ресурсів.

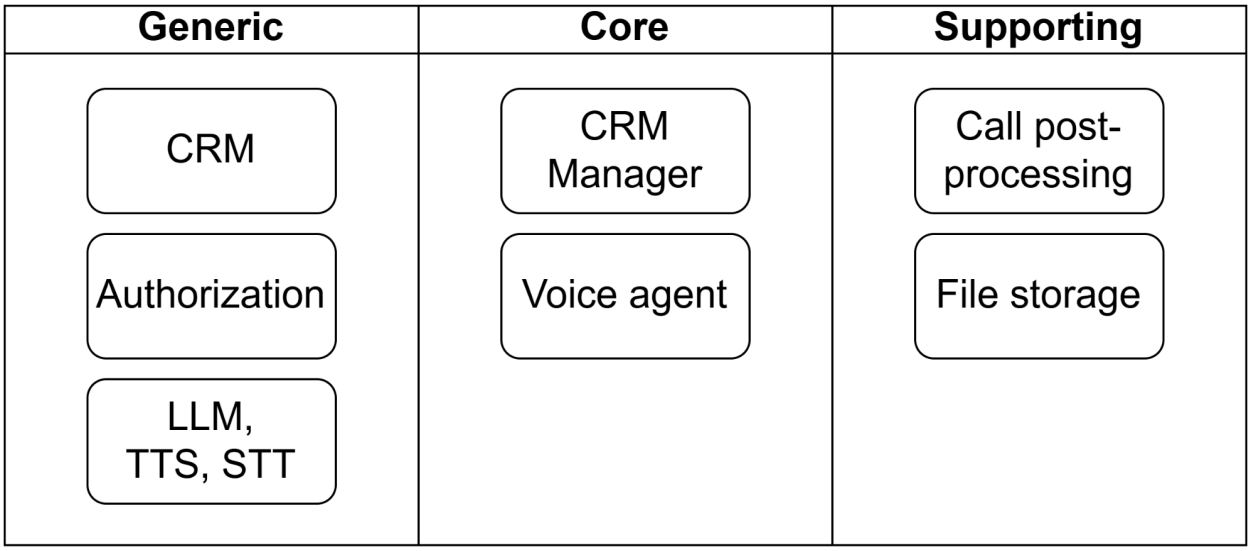


Рис. 2.1 Діаграма типів доменів системи обробки вербальних запитів в сфері гостинності

2.1.3 Bounded Context та зв'язок з мікросервісами

Bounded Context представляє явну межу, в рамках якої domain model має конкретне значення. В межах одного bounded context модель залишається консистентною та цілісною. Різні bounded contexts можуть мати різні інтерпретації тих самих концептів, що є нормальним та очікуваним.

Мікросервісна архітектура набула популярності як один з провідних архітектурних підходів для розробки масштабних систем, замінюючи монолітну архітектуру. Стратегічний DDD здобув визнання як провідний архітектурний підхід для розробки мікросервісів. Bounded contexts переходять у мікросервіси, забезпечуючи логічну основу для декомпозиції.

В той же час, для пришвидшення темпів розробки та збереження значної кількості переваг мікросервісної архітекрури, можна використовуати модульний моноліт. Модульний моноліт зберігає всю бізнес-логіку в межах єдиного розгортання, водночас розділяючи її на автономні модулі. Кожен модуль відповідає за окремий набір функцій і має власний Bounded Context. Така організація коду дозволяє командам працювати ізольовано, зменшує залежності та полегшує тестування. Пізніше модулі за потреби можуть стати окремими мікросервісами без втрати консистентності та зрозумілості доменної моделі.

2.2 Проєктування сервісу “CRM manager” для інтеграції з Twenty CRM

CRM Manager служить центральним мікросервісом для управління клієнтськими даними в системі обробки голосових запитів. Його основна функція – забезпечити абстракцію над Twenty CRM та надати уніфікований API для інших компонентів системи.

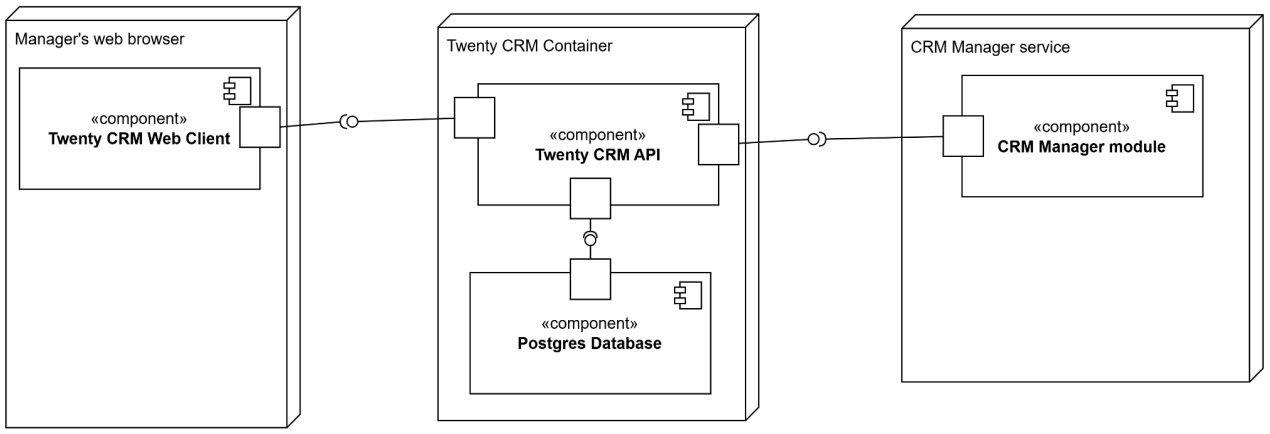


Рис. 2.2 Діаграма компонентів сервісів Twenty CRM та CRM Manager

2.2.1 Архітектурна роль та функції

CRM Manager працює як адаптер між внутрішньою доменною моделлю та зовнішньою платформою Twenty CRM. Сервіс приховує деталі Twenty CRM від решти системи. Якщо потрібно змінити CRM платформу, зміни торкнуться лише цього сервісу. Код організований навколо бізнес-сутностей (Client, Rental, Accommodation, Call), а не технічних деталей. Архітектура розділена на три шари – domain (бізнес-логіка), application (сценарії використання), infrastructure (технічна реалізація).

2.2.2 Доменна модель

Система оперує чотирма основними сутностями. Client містить дані клієнта: ім'я, прізвище, телефон, мовні налаштування. Включає валідацію номерів телефонів та підтримку двох мов (англійська, українська).

Rental описує об'єкт нерухомості: локація, ціна за день, опис, зручності. Містить логіку для пошуку вільних періодів та створення бронювань. Accommodation представляє конкретне бронювання з датами та статусом. Управляє станами від очікування підтвердження до завершення. Call зберігає дані про телефонні дзвінки для подальшої обробки.

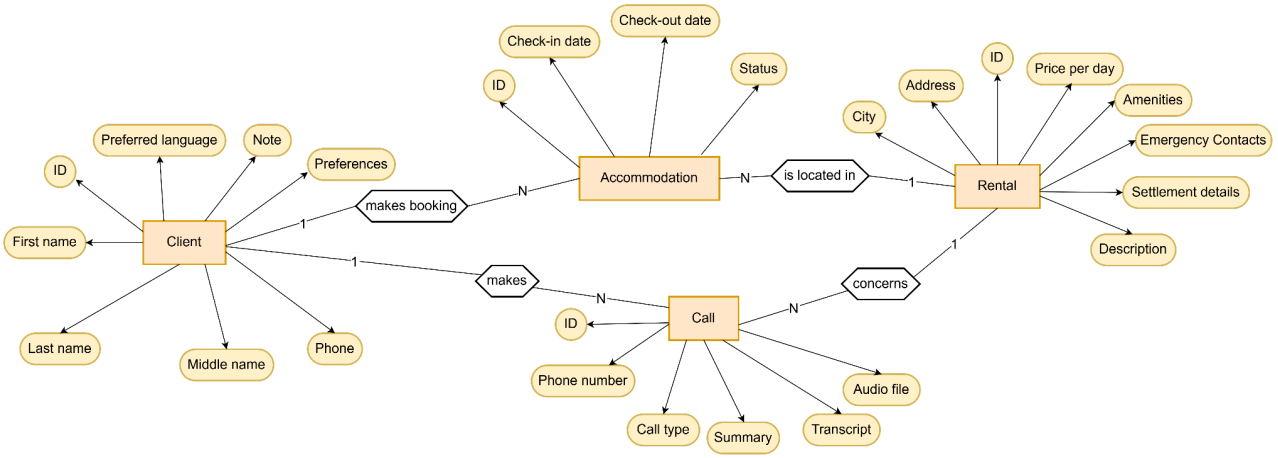


Рис. 2.3 Інфологічна модель домену сервісу CRM Manager

Структура зберігання даних доменної моделі реалізована згідно з датологічною моделлю, що відображає сутності та зв'язки між ними у вигляді реляційних таблиць. Цей рівень моделювання, на відміну від інфологічного, містить всі необхідні атрибути для забезпечення цілісності та зв'язності даних при зберіганні.

Дані кожної доменної сутності організовані у відповідних таблицях:

- інформація про клієнтів зберігається в таблиці clients. Ця таблиця містить атрибути, що описують ідентифікаційні дані кожного клієнта, його контактну інформацію та налаштування – зокрема, мовні преференції;
- відомості про об'єкти нерухомості (оренди) містяться в таблиці rentals. Тут фіксуються атрибути, що характеризують кожен об'єкт: його фізичне місцезнаходження, вартість оренди за період, детальний опис та перелік доступних зручностей;
- деталі кожного бронювання фіксуються в таблиці accommodations. Кожен запис у цій таблиці представляє конкретне бронювання певного об'єкта нерухомості певним клієнтом на визначений період часу та містить атрибути, що відображають статус бронювання;
- інформація про кожен телефонний дзвінок зберігається в таблиці calls. Ця таблиця містить атрибути, що описують сам факт дзвінка, його тип, основне резюме та посилання на повну транскрипцію та аудіозапис.

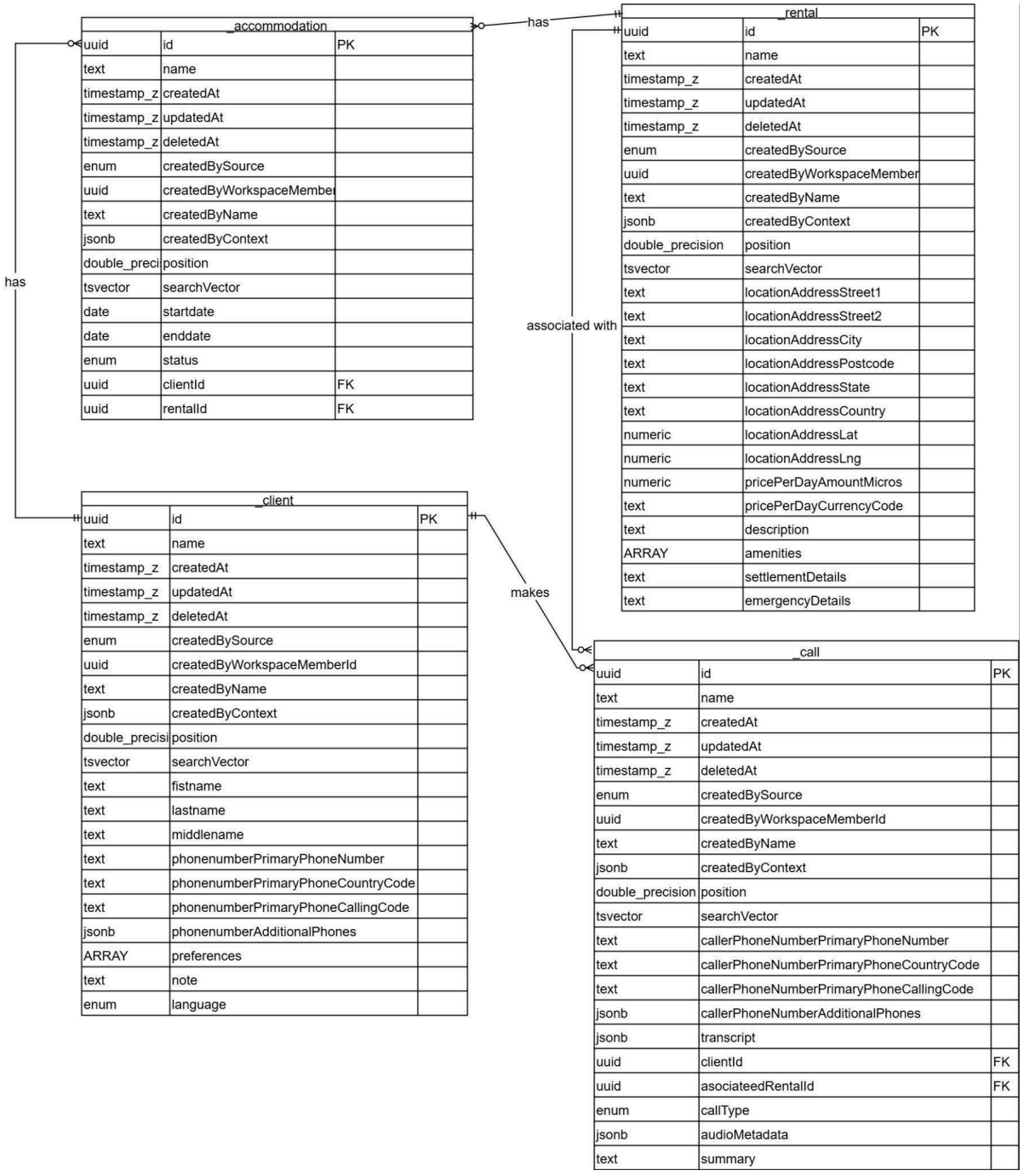


Рис. 2.4 Даталогічна модель бази даних сервісу CRM Manager

Зв'язки між сутностями, зображені на датологічній моделі, реалізуються на рівні схеми бази даних шляхом встановлення відношень між записами в цих таблицях. Зокрема:

– відношення "makes booking" між Client та Accommodation реалізується через атрибут в таблиці accommodations, який посилається на відповідний запис у таблиці clients, визначаючи, хто саме здійснив бронювання;

- відношення "makes" між Client та Call також реалізується через атрибут у таблиці calls, що вказує на клієнта з таблиці clients, який ініціював або був учасником дзвінка;
- зв'язок "is associated with" між Accommodation та Rental відображається атрибутом у таблиці accommodations, який пов'язує кожне бронювання з конкретним об'єктом нерухомості, описаним у таблиці rentals;
- відношення "concerns" між Call та Rental встановлюється через атрибут у таблиці calls, який посилається на запис у таблиці rentals, вказуючи, якого об'єкта нерухомості стосувалася розмова.

Така реляційна структура зберігання даних забезпечує інтегроване представлення доменних сутностей та їхніх взаємозв'язків, дозволяючи ефективно здійснювати операції читання та запису даних, необхідні для функціонування системи.

2.2.3 Інтеграція з Twenty CRM

Інтеграція реалізована через Repository pattern з наступними компонентами. Система використовує згенерований TypeScript клієнт twenty-crm-api-client для типобезпечного доступу до Twenty CRM API.

Кожна доменна сутність має відповідний репозиторій, який реалізує операції CRUD. Спеціальні mappers перетворюють дані між доменними об'єктами та форматом Twenty CRM. Кожен mapper відповідає за одну сутність.

2.3 Проєктування сервісу голосового асистента

Голосовий агент — це автономна програмна система, яка в реальному часі перетворює мовні запити користувача на текст, аналізує їх за допомогою великої мовної моделі та одразу генерує звукову відповідь природною мовою, мінімізуючи паузи між діалоговими ходами. Для розширення функціональності такі агенти використовують механізми виклику зовнішніх інструментів (function calling, Model Context Protocol тощо), що дозволяють оперативно збирати

необхідні дані з бекенд-сервісів, запускати бізнес-логіку чи оновлювати стан системи. Завдяки цьому голосовий агент може виконувати складні запити, інтегруватися з базами даних, CRM чи IoT-пристроями та віддавати результат у вигляді мови з мінімальною затримкою, створюючи враження наближеності до спілкування з людиною.

2.3.1 Структура та взаємодія компонентів

Всередині мікросервісу Voice agent працює кілька ключових компонентів, що є складовими фреймворку Pipecat Flows. Local Transport обробляє передачу аудіопотоку з периферійного обладнання комп'ютера. В той же час, існує можливість підключити голосовий асистент напряму до VoIP-провайдера Twilio чи Daily.co для інтеграції телефонних дзвінків. Компонент Silero VAD (Voice Activity Detector) визначає, коли в аудіо є активність голосу, відфільтровуючи тишу та забезпечуючи швидший відгук агента на «перебивання» користувача, ніж це могло бути з використанням лише моделі TTS. «Перебивання» існує для того, щоб забезпечити комфорт користувача, давши йому можливість припинити відтворення репліки асистента за допомогою своєї контр-фрази, таким чином, уточнивши чи давши відповідь ще до того, як агент встигне завершити свою репліку повністю.

Аудіо, що містить голосову активність, надходить у Pipeline. Це центральний оркестратор, який керує послідовністю обробки. Pipeline взаємодіє з кількома спеціалізованими сервісами:

- Elevenlabs STT Service використовує Elevenlabs для перетворення голосової інформації в текст (Speech-to-Text);
- Deepgram TTS Service використовує Deepgram для зворотного перетворення тексту в голос (Text-to-Speech);
- Open AI LLM Service підключається до OpenAI для розуміння запитів, обробки бізнес-логіки та генерації відповіді на основі тексту за допомогою LLM;

- "Voice Agent Application Layer" є рівнем бізнес-логіки, що координує взаємодію між Pipeline та іншими внутрішніми та зовнішніми сервісами. Він отримує результат від Pipeline (оброблений текст або команди) і керує ходом розмови через команди до Pipeline;
- компонент CRM Manager Service відповідає за взаємодію з CRM. Він отримує запити від Application Layer (наприклад, знайти клієнта) і використовує інтеграцію з мікросервісом CRM Manager для виконання цих операцій;
- File storage service працює зі сховищами, що використовують протокол S2. В даному випадку, застосовується контейнер Localstack S3, що є локальним S3 File storage. Він використовується для зберігання аудіозаписів розмов для подальшої передачі їх в інші сервіси;
- Bull MQ Client використовується для надсилання подій про завершення дзвінка в чергу. Події потім можуть бути асинхронно опрацьовані іншим сервісом. Redis тут виступає як message broker, що дозволяє обробляти задачі асинхронно, наприклад, ставити в чергу довгі операції або повідомлення між компонентами.

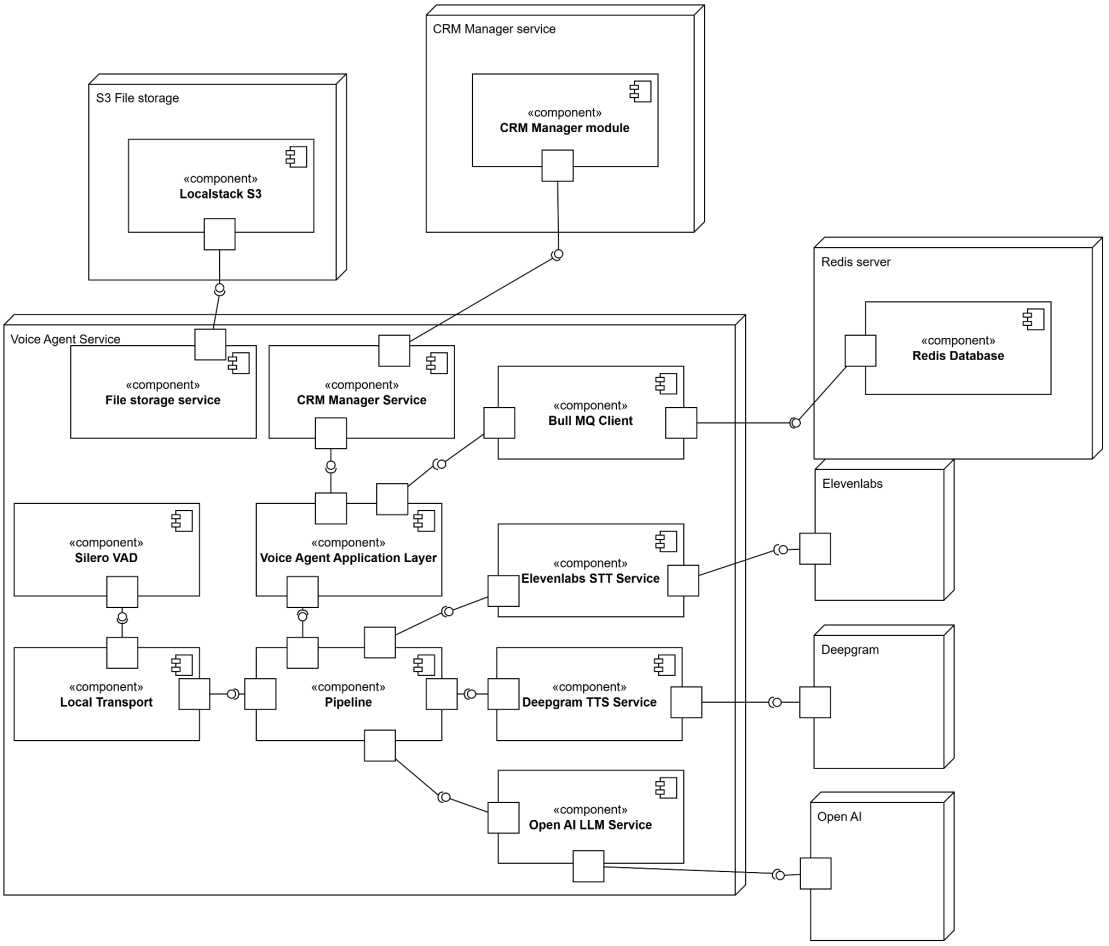


Рис. 2.5 Діаграма компонентів Voice agent та дотичних сервісів

Таким чином, голос проходить через ланцюжок з VAD, Transport, Pipeline (STT, LLM, TTS) під управлінням Application Layer. Application Layer взаємодіє з CRM Manager та іншими сервісами. Файли зберігаються в Localstack S3 через File storage service, а асинхронні задачі обробляються через Bull MQ та Redis. Це забезпечує послідовну та організовану обробку голосових запитів, інтеграцію з даними клієнтів та ефективне використання зовнішніх ШІ-сервісів.

2.3.2 Роль Application layer в забезпеченні стабільності та безпеки даних в сервісі голосового асистента

Швидке впровадження голосових агентів на базі великих мовних моделей у корпоративні системи створює нові виклики для архітектури програмного забезпечення. Традиційні підходи до розробки не враховують специфіку інтеграції LLM як компонента, який може бути скомпрометованим

зловмисниками через промпт-ін'єкції та інші атаки на рівні обробки природної мови. Це вимагає перегляду архітектурних рішень з метою мінімізації ризиків безпеки.

Забезпечення цілісності дотримання правил у корпоративному програмному забезпеченні набуває критичного значення при інтеграції агентських систем. Корпоративні системи оперують чутливими даними та виконують бізнес-критичні операції, тому будь-яка вразливість може призвести до значних фінансових втрат або порушення конфіденційності. Відокремлення основних бізнес-правил від логіки взаємодії з LLM дозволяє створити додатковий рівень захисту, де навіть у випадку компрометації агентської частини, критичні операції залишаються під контролем перевірених модулів системи. Такий підхід забезпечує принцип найменших привілеїв для LLM компонентів та дозволяє застосовувати традиційні методи валідації та авторизації незалежно від поведінки мовної моделі.

Агентські системи з голосовим інтерфейсом користувача демонструють значний потенціал для трансформації способів взаємодії з корпоративними додатками. VUI надає природний та інтуїтивний спосіб доступу до складних систем, знижуючи бар'єри для користувачів та підвищуючи продуктивність роботи. Голосові агенти можуть обробляти складні запити в реальному часі, інтегруватися з множиною backend-сервісів та надавати персоналізовані відповіді на основі контексту користувача. Однак, даний підхід вимагає ретельного проектування архітектури, де LLM частково виступає як контролер бізнес-логіки, але, в той же час, містить в собі відомості про домен.

Сучасні підходи до інтеграції LLM у агентських системах базуються на механізмах function calling – технології, яка дозволяє мовним моделям взаємодіяти із зовнішніми інструментами через структуровані запити. Цей процес передбачає генерацію мовними моделями JSON-об'єктів із параметрами, що обробляються на стороні сервісу, котрий здійснює виклик мовної моделі, після чого результат для необхідного інструменту надсилається з наступним викликом API-моделі.

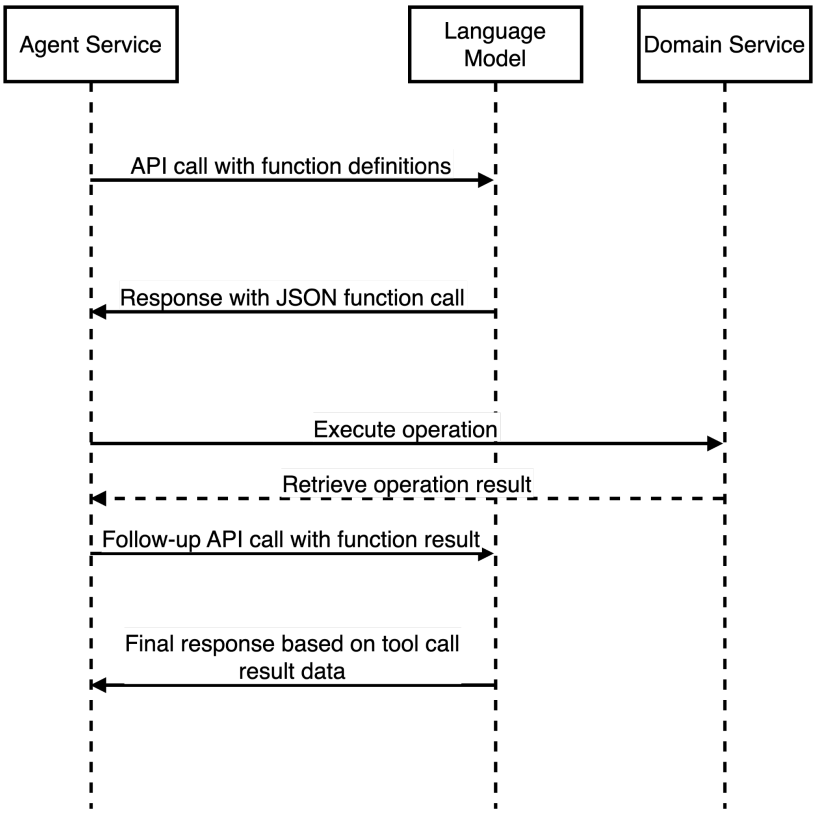


Рис. 2.6 Діаграма послідовності function calling

Ще одним способом надати агентському рішенню доступ до зміни стану системи через виклик структурованих команд є Model Context Protocol (MCP), що представляє собою відкритий стандарт, розроблений компанією Anthropic для забезпечення так званої «безшовної» інтеграції між LLM та зовнішніми джерелами даних і інструментами. Протокол використовує архітектуру «клієнт-хост-сервер»,, де хост-додаток створює та керує множинними клієнтськими екземплярами, кожен з яких підтримує ізольоване з'єднання з окремим сервером. MCP базується на JSON-RPC 2.0-повідомленнях та забезпечує стандартизований спосіб обміну контекстом між компонентами системи через три основні примітиви: ресурси, промпти та інструменти (функції для виконання LLM). Протокол реалізує систему узгодження можливостей, де клієнти та сервери явно декларують свої підтримувані функції під час ініціалізації, що забезпечує чітке розуміння доступного функціоналу та підтримує розширюваність протоколу. Ключовою особливістю MCP є принцип

ізоляції, згідно з яким сервери не мають доступу до повної історії розмови та не можуть мати доступ до конфігурації моделі, що забезпечує безпеку та дозволяє хост-процесу зберігати межі між компонентами.

Фундаментальний принцип безпечної інтеграції голосових агентів полягає у чіткому розділенні основних бізнес-правил від логіки взаємодії з мовними моделями. Цей підхід створює архітектурний бар'єр, що обмежує потенційний вплив атак на LLM компоненти та на критично важливі операції системи. Функціональна референтна архітектура для LLM-інтегрованих систем пропонує багатошарову структуру з виділеними компонентами для моніторингу та захисту.

Application Layer відіграє критичну роль у цьому розділенні. Хоча LLM використовується для інтерпретації намірів користувача та ведення діалогу, безпосереднє виконання бізнес-логіки та операцій з даними відбувається не в самій LLM, а в межах Application Layer, або через виклики до спеціалізованих зовнішніх сервісів, таких як CRM Manager.

LLM взаємодіє з бізнес-логікою через структуровані механізми, подібні до function calling. У цій моделі LLM, отримавши контекст діалогу, може запропонувати виконати певну дію, ідентифікуючи відповідну "функцію" та необхідні параметри (наприклад, витягнуті з мови користувача). Ця пропозиція передається до Application Layer у структурованому форматі.

Application Layer отримує команду від LLM і валідує її перед виконанням. Валідація може включати перевірку наявності необхідних параметрів, їх відповідності очікуваним типам та форматам, а також перевірку відповідності запиту поточним бізнес-правилам та стану системи.

Важливим елементом цього є використання компонента стану розмови ConversationContext, який зберігає інформацію про поточний сеанс. Цей компонент може містити локальні бізнес-правила, що визначають можливість виконання певних дій на даному етапі діалогу. Наприклад, перевірки про можливість заселення чи видачі доступу до інформаційних послуг конкретного житла, використовуючи дані про клієнта та бронювання, вже отримані з CRM

Manager. Ці локальні перевірки слугують першим рівнем захисту перед тим, як навіть буде зроблена спроба викликати відповідну функцію в CRM Manager.

Результат виконання бізнес-логіки (успіх операції, помилка валідації від CRM Manager або локальної перевірки) повертається назад до Application Layer, який передає цей структурований результат LLM, яка, в свою чергу, використовує цей результат, щоб сформувати відповідну текстову відповідь для користувача, коректно реагуючи на успіх чи невдачу операції, не маючи при цьому прямого доступу до виконання критичних дій.

Такий підхід забезпечує, що:

1. критична бізнес-логіка та правила цілісності даних залишаються поза контролем LLM;
2. LLM функціонує як інтелектуальний інтерфейс для інтерпретації намірів та управління діалогом;
3. Application Layer діє як посередник, валідатор та координатор між LLM та реальними бізнес-операціями;
4. навіть якщо LLM згенерує шкідливу пропозицію (наприклад, через промпт-ін'єкцію), Application Layer або бізнес-логіка в CRM Manager відхилить її на етапі валідації.

Для забезпечення ще більшої ізольованості та передбачуваності агентської поведінки пропонується використання архітектурного підходу з багатоагентними системами, де кожен етап обробки запиту виконується спеціалізованим агентом із чітко визначеним контекстом дій. Такий підхід дозволяє сегментувати складні завдання на послідовні ланцюги операцій, де кожен агент має доступ лише до обмеженого набору інструментів та доменних знань, що суттєво знижує поверхню атаки та ризики компрометації.

У сучасних дослідженнях все частіше наголошується на перевагах ієрархічної організації агентів, які виконують окремі функції у чітко визначеній послідовності. Це дозволяє не лише підвищити безпеку, але й забезпечити більшу передбачуваність роботи системи. Ієрархічна організація агентів із механізмами валідації проміжних станів забезпечує дотримання глобальних

обмежень навіть при обробці складних багатоетапних запитів. Кожен агент у такій системі виконує чітко визначену функцію:

5. ініціалізація контексту – аналіз вхідного запиту та визначення необхідних доменних сервісів для звернення до них через інструменти;

6. виконання доменних операцій – взаємодія з бекенд-сервісами через стандартизовані API;

7. генерація відповіді – формування текстового/голосового відгуку з урахуванням контексту.

Така послідовність етапів усуває необхідність надання LLM прямого доступу до критичних інструментів, оскільки кожен агент працює в рамках свого ізольованого середовища з обмеженим набором привілеїв.

Ключовим аспектом є динамічне формування набору доступних інструментів для кожного агента на основі:

- поточного етапу workflow;
- рівня авторизації користувача;
- історичного контексту взаємодії.

Введення передбачуваних моделей для оцінки ризиків кожного виклику інструменту дозволяє проактивно блокувати потенційно небезпечні операції. Наприклад, агент нормалізації запитів може мати доступ лише до семантичного парсера, тоді як агент виконання операцій – виключно до доменних API з жорсткою валідацією вхідних параметрів.

Ще однією важливою перевагою послідовної багатоагентної архітектури є зменшення залежності від складних процедур донавчання великих мовних моделей. Завдяки вузькій спеціалізації кожного агента та стандартизованим шаблонам промптів, система може ефективно працювати навіть із моделями без доменної адаптації.

Послідовна архітектура усуває необхідність складного налаштування моделей через спеціалізацію агентів, так як кожен компонент вирішує вузьке завдання, що не вимагає універсальності LLM, а також контекстну ізоляцію, що забезпечує обмеження доменних знань у межах одного агента.

Велика мовна модель у запропонованій архітектурі виступає як інтелектуальний медіатор, що перетворює природномовні запити користувача в структуровані виклики API, але залишається ізольованою від критичних даних домену. Цей підхід об'єднує всі попередні принципи, створюючи систему, де LLM є потужним інтерфейсом, а не носієм критичної бізнес-логіки.

2.3.3 Принцип забезпечення підтримки багатомовності в розрізі конфігурації TTS та STT моделей

Система реалізує багатомовну підтримку через комбінацію статичної конфігурації мовних параметрів на рівні STT/TTS та гібридного підходу до керування мовою на рівні LLM. Ця архітектура дозволяє уникнути надмірної складності при масштабуванні, зберігаючи високу точність транскрипції та синтезу.

Мова взаємодії визначається до ініціалізації сеансу на основі клієнтських метаданих або виділених телефонних номерів для кожної мови. На відміну від динамічного виявлення мови під час розмови, такий підхід дозволяє:

1. уникнути конфліктів між STT-моделями при спробі розпізнати кілька мов одночасно;
2. заборонити непередбачені зміни мови під час критичних операцій (наприклад, підтвердження платежів);
3. оптимізувати використання обчислювальних ресурсів через попереднє знання мовного контексту.

Для англійської та української мов повинні використовуються окремі точки входу, що активізує відповідний набір конфігураційних параметрів цього мікросервісу.

Система використовує STT модель Nova-2-general через її підтримку 36 мов – включно з українською. Незважаючи на появу Nova-3 з функцією real-time code-switching, Nova-2 залишається оптимальним вибором, оскільки новіша модель не даний момент не забезпечує підтримки української мови. Мовний параметр задається динамічно через language="uk" для української або

language="en" для англійської без необхідності створення окремих екземплярів сервісу. Також, важливо зазначити, що модель з коректними параметрами автоматично адаптується до акцентів та фонових шумів завдяки технології штучного розширення тренувальних даних та має допустимий показник WER на рівні <10%.

Варто вказати, що спроби використати модель з налаштуванням розпізнавати українську мову показали, що модель здатна розпізнавати та англійську мову, але цей спосіб спричиняє недопустиме значення WER.

Для синтезу мовлення використовується модель eleven_multilingual_v2, яка підтримує 32 мови та має затримку генерації 90 мс. На противагу STT, TTS вимагає жорсткої прив'язки до voice_id для мови, у випадку якщо голос не підтримує усі необхідні мови. Ця умова не є обов’язковою, але без її дотримання вимова може мати виражений акцент, що може справити негативне перше враження клієнта.

На рівні великої мовної моделі застосовується гібридний підхід. Основні промпти (role_messages, task_messages) залишаються англомовними з метою уніфікації розробки та підтримки, тоді як додаткове системне повідомлення українською мовою інструктує модель формувати всі вихідні тексти українською літературною мовою. Така стратегія дозволяє не дублювати англомовні шаблони, зберігаючи при цьому повну україномовну взаємодію кінцевого користувача. Після отримання відповіді LLM, згенерованої відповідно до системного повідомлення, текст переходить на синтез з уже встановленим кодом мови та voice_id TTS.

Таким чином, підтримка багатомовності реалізована через динамічну передачу мовних параметрів у запитах до STT та TTS сервісів, комбіновану з локалізацією мовних інструкцій на рівні LLM. Такий підхід забезпечує гнучкість конфігурації, передбачувану якість розпізнавання та синтезу голосу та спрощує масштабування системи на нові мови.

2.4 Проєктування модуля Post-Processing для опрацювання завершених дзвінків

Модуль Post-Processing призначений для комплексної обробки даних після завершення голосового дзвінка, включаючи аналіз транскриптів, створення резюме та збереження результатів у системі управління відносинами з клієнтами. Відповідно до принципів Domain-Driven Design, цей модуль класифіковано як Supporting Domain, що має фундаментальні наслідки для архітектурних рішень та стратегії розробки. Supporting Domain характеризується необхідністю мінімального використання ресурсів розробки при забезпеченні високої ефективності функціонування, оскільки він підтримує основні бізнес-процеси, але не є критичним для конкурентних переваг організації. Тому з метою зменшення зусиль на реалізацію, було прийнято рішення розробити його в рамках модульного моноліту з сервісом CRM Manager. Таке архітектурне рішення забезпечує модулю Post-Processing можливість безпосереднього використання use cases та сутностей модуля CRM без необхідності реалізації складної міжсервісної HTTP-комунікації. Це критично важливо для Supporting Domain, оскільки зменшує складність розробки та потенційні ризики, пов'язані з мережевою взаємодією.

2.4.1 Асинхронна комунікація із застосуванням брокера повідомлень

Обробка завершених дзвінків має бути асинхронною з декількох критично важливих причин, пов'язаних з архітектурою та продуктивністю системи. Синхронна обробка може призвести до блокування основного потоку обробки дзвінка, що неприпустимо для систем реального часу, де затримки напряму впливають на якість користувацького досвіду. Асинхронний підхід також забезпечує відмовостійкість системи, дозволяючи продовжувати обробку нових дзвінків навіть у випадку тимчасових проблем з компонентами постобробки.

Для реалізації асинхронної взаємодії обрано брокер повідомлень BullMQ з використанням Redis як базового сховища даних. Цей вибір обґрунтований

високою продуктивністю Redis для операцій в пам'яті та надійністю BullMQ для обробки черг повідомлень. Такий підхід особливо важливий у контексті систем, що обробляють дані з різних доменів, де необхідно забезпечити консистентність та надійність передачі інформації.

Схема взаємодії реалізована таким чином, що сервіс Agent після завершення дзвінка публікує подію call completed у чергу BullMQ, інкапсулюючи всю необхідну інформацію про завершений дзвінок, включаючи повний транскрипт розмови. Модуль Post-Processing у складі crm-manager виступає споживачем цих повідомлень, використовуючи CallCompletedProcessor для обробки вхідних подій. Такий підхід забезпечує розділення відповідальності між компонентами системи, що є ключовим принципом мікросервісної архітектури.

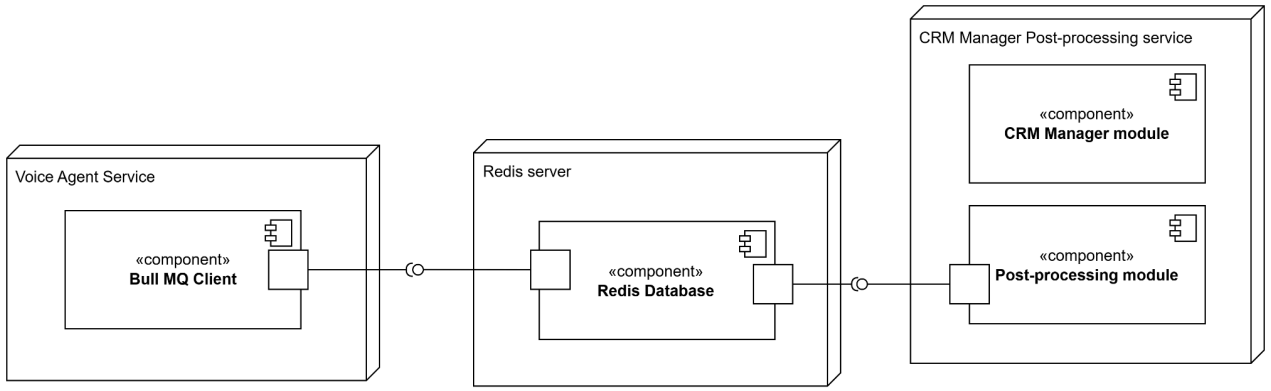


Рис. 2.7 Діаграма компонентів Voice agent та Post processing module

Використання брокера повідомлень у даному контексті надає значні переваги для забезпечення надійності та масштабованості системи. Гарантована доставка повідомлень забезпечує, що жодна інформація про завершені дзвінки не буде втрачена навіть у випадку тимчасових збоїв у роботі системи. Можливість масштабування досягається через здатність BullMQ обробляти великі обсяги повідомлень та підтримувати множинних споживачів для однієї черги. Дослідження в галузі адаптації доменів показують, що ефективне

вирівнювання розподілів між доменами є критично важливим для успішної роботи систем, що обробляють дані з різних джерел.

2.5 Проєктування BFF з авторизацією та мікрофронтенду для перегляду транскрипції дзвінків та прослуховування аудіозапису дзвінка

Backend for Frontend як архітектурний паттерн надає ефективне рішення для вирішення проблем безпеки та оптимізації взаємодії клієнта з множиною бекенд-сервісів. BFF розташовується між клієнтським застосунком у браузері менеджера та внутрішніми сервісами, включаючи CRM Manager та File Storage, виступаючи єдиною точкою входу для клієнтських запитів. Такий підхід забезпечує централізований контроль безпеки та дозволяє оптимізувати протоколи взаємодії відповідно до специфічних потреб клієнтського застосунку, що є ключовою перевагою BFF-патерну в мікросервісних архітектурах.

2.5.1 Застосування Next.js як Full-stack Framework для побудови BFF

Вибір Next.js як технологічної основи для реалізації BFF обґрунтований його універсальністю як full-stack фреймворку, що дозволяє ефективно реалізувати як клієнтську частину з використанням React, так і серверну частину через API routes. Next.js розширює можливості React, додаючи серверний рендеринг, автоматичну оптимізацію та вбудовані механізми для створення API endpoints, що робить його ідеальним вибором для реалізації BFF-паттерну. Переваги використання full-stack фреймворку в контексті даного проєкту включають значне спрощення процесу розробки через уніфікацію технологічного стеку, підтримку єдиної кодової бази для фронтенду та бекенду, а також легку інтеграцію між клієнтською та серверною частинами застосунку.

React як основа для клієнтської частини забезпечує створення сучасного, реактивного користувацького інтерфейсу з високою продуктивністю та відмінним користувацьким досвідом. Компонентна архітектура React дозволяє створювати модульні та повторно використовувані елементи інтерфейсу, що

особливо важливо для реалізації мікрофронтенд-підходу в системі перегляду деталей дзвінків. API routes Next.js надають потужний механізм для створення серверних ендпоінтів, що можуть служити як BFF-шар, обробляючи автентифікацію, авторизацію та проксування запитів до внутрішніх сервісів системи.

2.5.2 Реалізація авторизації за допомогою Clerk

Впровадження надійної системи авторизації є критично важливою вимогою для забезпечення безпечного доступу до конфіденційних даних дзвінків, оскільки такі дані можуть містити персональну інформацію клієнтів та бізнес-критичні відомості. Доступ до системи повинен бути обмежений виключно авторизованими користувачами з відповідними правами доступу, що вимагає реалізації комплексних механізмів автентифікації та авторизації. Вибір рішення для управління користувачами та авторизацією має враховувати не лише технічні аспекти безпеки, але й зручність інтеграції з обраним технологічним стеком та можливості масштабування системи.

Clerk представляє собою сучасний сервіс для управління користувачами та авторизацією, що надає готові рішення для інтеграції з Next.js та іншими популярними фреймворками. Переваги використання Clerk включають вбудовану підтримку різних методів автентифікації, готові компоненти користувацького інтерфейсу для реєстрації та входу, а також комплексні засоби управління сесіями користувачів. Інтеграція Clerk з Next.js забезпечується через спеціалізовані middleware-компоненти та React hooks, що дозволяють легко додавати перевірку авторизації як на клієнтській, так і на серверній стороні застосунку.

Схема авторизації в BFF реалізується через багаторівневий підхід, де Clerk middleware перевіряє наявність валідної сесії користувача перед обробкою будь-яких запитів до захищених ресурсів. На рівні API routes Next.js кожен endpoint, що надає доступ до даних дзвінків, захищений перевіркою автентифікації через Clerk SDK, що гарантує обробку лише авторизованих

запитів. Додавання нових користувачів здійснюватиметься через адміністративну панель Clerk Dashboard, що забезпечує централізоване управління користувачами без необхідності реалізації власного функціоналу реєстрації. Такий підхід відповідає принципам мінімізації ризиків під час розробки та гарантування безпеки системи.

2.5.3 Отримання та відображення даних дзвінка з використанням підходу Server-Side Rendering

Система відображення деталей дзвінка оперує з трьома основними типами даних: транскрипція дзвінка у текстовому форматі, автоматично згенероване summary розмови та аудіозапис у вигляді файлу. Кожен з цих типів даних має специфічні вимоги до обробки та доставки користувачеві, що вимагає диференційованого підходу до їх отримання та відображення.

Для статичних даних, включаючи транскрипцію та короткі підсумки (summary) дзвінка, обрано стратегію Server-Side Rendering в Next.js, що забезпечує оптимальну продуктивність та безпеку системи. При запиті сторінки з деталями конкретного дзвінка BFF на серверній стороні виконує автентифіковані запити до CRM Manager для отримання транскрипції та резюме, після чого рендерить HTML сторінку, що вже містить всі необхідні текстові дані. Дослідження підходів до рендерингу веб-застосунків показують, що SSR значно покращує швидкість початкового завантаження контенту та зменшує навантаження на клієнтську частину. Такий підхід також гарантує кращу безпеку, оскільки конфіденційні API-ключі та внутрішні ендпоінти залишаються недоступними для клієнтської сторони.

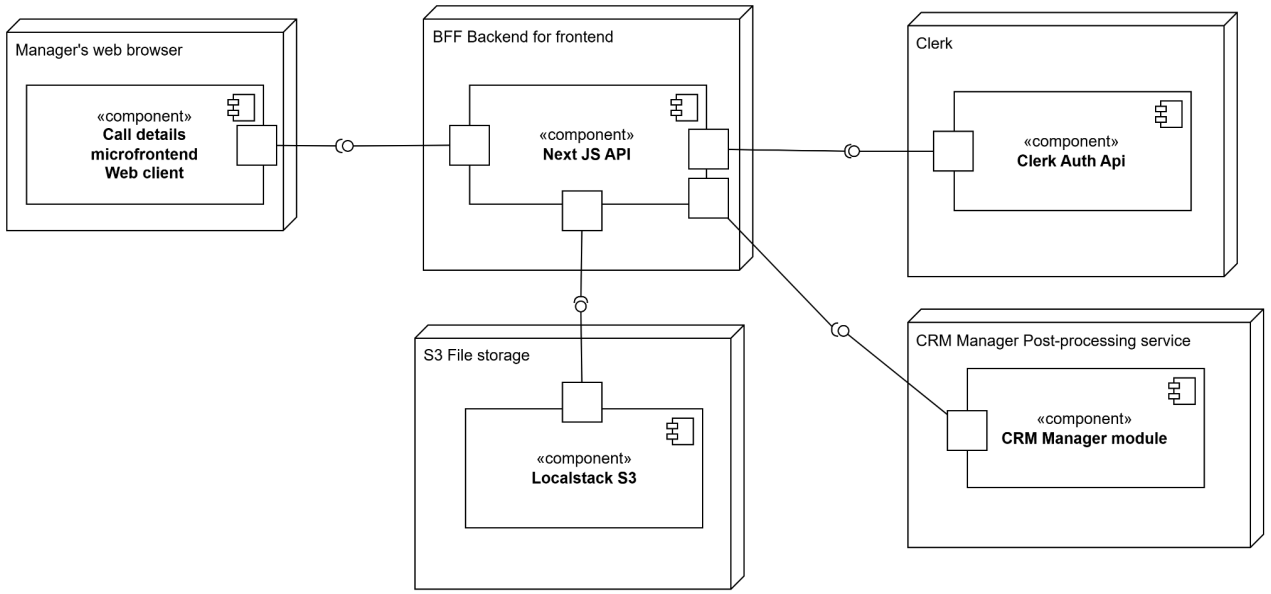


Рис. 2.8 Діаграма компонентів BFF, S3 File Storage, CRM manager та Clerk

Для динамічних даних, зокрема, аудіозаписів дзвінків, реалізовано захищений API маршрут у BFF, що служить проксі між клієнтом та File Storage сервісом. Цей endpoint перевіряє авторизацію користувача через Clerk, а потім стрімить аудіофайл з Localstack S3 безпосередньо до клієнта, забезпечуючи при цьому контроль доступу на кожному етапі запиту. Альтернативним підходом може бути генерація тимчасових, захищених URL для прямого доступу до S3 ресурсів, проте проксування через BFF забезпечує більший контроль та можливості логування доступу до конфіденційних даних.

2.5.4 Мікрофронтенд для перегляду деталей дзвінка

Концепція мікрофронтендів представляє архітектурний підхід до розбиття великих фронтенд-застосунків на менші, незалежно розгортувані та підтримувані компоненти, що дозволяє різним командам розробки працювати автономно над окремими частинами користувацького інтерфейсу. Цей підхід особливо ефективний у контексті великих корпоративних систем, де різні функціональні області можуть мати різні вимоги до технологій, циклів розгортання та команд розробки. Мікрофронтенди забезпечують технологічну

незалежність, дозволяючи використовувати найбільш релевантні інструменти для кожного конкретного випадку використання.

Компонент відображення деталей дзвінка, що включає інтерфейси для перегляду транскрипції, резюме та аудіоплеєр для прослуховування записів, реалізовано як незалежний мікрофронтенд у рамках Next.js застосунку, доступ до якого буде надаватися за посиланням, що розміщуватиметься в Twenty CRM Web Client.

2.6 Проєктування інтерфейсу системи

Для розробки Web calls dashboard було прийнято рішення застосувати модульний підхід та сучасні інструменти. Ідея полягає в тому, щоб збудувати єдину візуальну систему на базі Shadcn UI Kit — набору React-компонентів, що поєднує мінімалістичний дизайн із тонкою кастомізацією через Tailwind CSS. Завдяки використанню стилізованих компонентів передбачається отримати консистентний вигляд елементів керування, гнучко налаштовувати стилі відповідно до корпоративних стандартів.

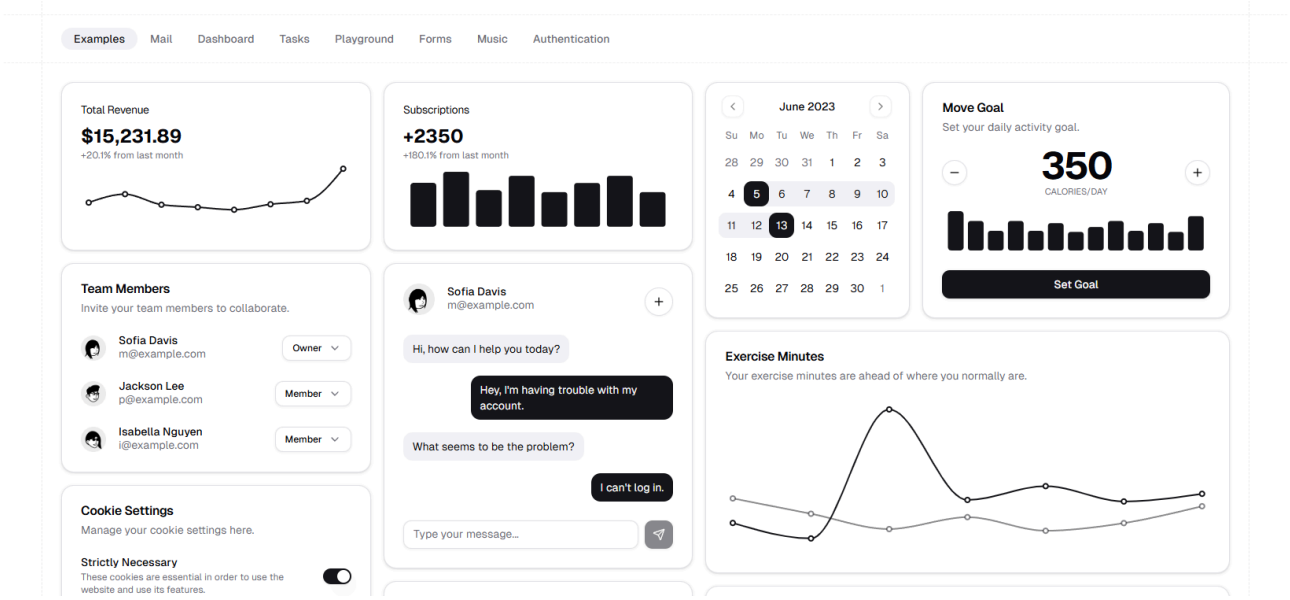


Рис. 2.9 Демонстрація компонентів Shadcn UI Kit

Для контролю доступу та автентифікації планується інтеграція Clerk, який забезпечить готові UI-компоненти та middleware для перевірки сесій на рівні маршрутів. Ця схема дозволить захищати приватні розділи панелі, спрямовуючи неавторизованих користувачів на сторінку входу та централізовано обробляючи токени доступу без додаткової реалізації власних механізмів.

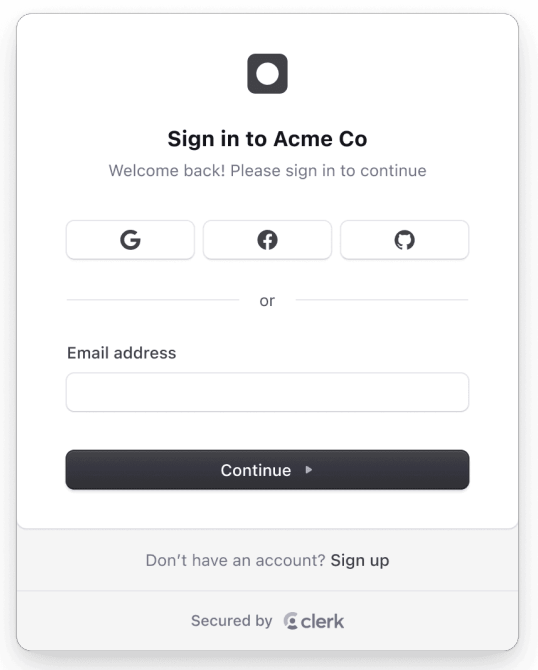


Рис. 2.10 Компонент входу в систему з Clerk UI Components

Візуальна ідентичність формується за допомогою Tailwind CSS, де глобальний файл стилів визначає палітру кольорів, типографіку, радіуси елементів і змінні для світлої та темної тем. Використання кастомних утиліт і змінних надасть можливість адаптувати інтерфейс під різні розміри екранів та системні налаштування, зберігаючи єдність стилю та високу контрастність.

vag-microservices - page.tsx

```tsx
<Card className="p-4">
  <h2 className="text-lg font-semibold mb-2">Summary</h2>
  {loading ? (
    <Skeleton className="h-4 w-2/3" />
  ) : (
    <div className="text-sm">{mockCall.summary}</div>
  )}
</Card>
```

Рис. 2.11 Приклад застосування стилізації з використанням класів Tailwind CSS

Структура головної сторінки передбачається у вигляді вертикального потоку блоків: панель фільтрів із Input-полями та Button-елементами зверху, а нижче — список Card-компонентів із ключовими метаданими дзвінків. Детальна сторінка дзвінка включатиме секції з розрізненням ролей у транскрипції, автоматично згенерованим резюме, вбудованим аудіоплеєром із візуалізацією хвилі та блоком метаданих.

3.7 Загальний огляд архітектури

Створена архітектура об’єднує набір взаємодіючих сервісів з чітким розмежуванням за доменними ролями, синхронними HTTP-запитами та асинхронною обробкою через BullMQ, що дозволяє знизити ризики під час розробки та забезпечити масштабованість.

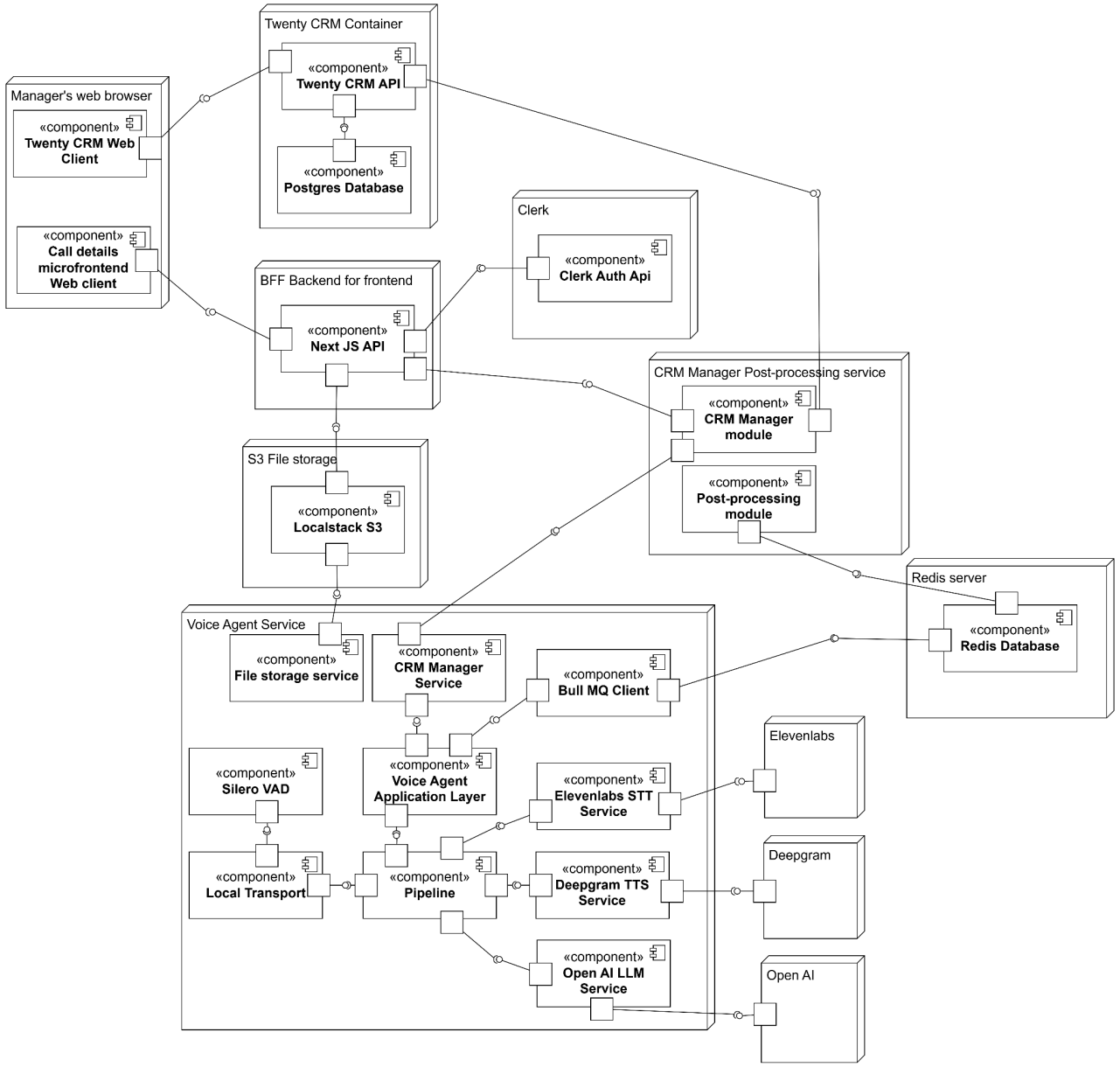


Рис. 2.12 Діаграма компонентів системи – повний вигляд

Починаючи з фронтенду, у браузері менеджера розгортаються два клієнти: Twenty CRM Web Client, який синхронно звертається до Twenty CRM API, і Call details microfrontend Web client, що працює через BFF – Next JS API. BFF виконує автентифікацію через Clerk Auth Api, передаючи HTTP-запити до цього сервісу і блокуючи доступ, якщо сесія відсутня. Далі Next JS API синхронно запитує Twenty CRM API для метаданих дзвінків і Localstack S3 для аудіофайлів, після чого формує готові відповіді для клієнта.

Voice Agent Service реалізовано як окремий мікросервіс із компонентами Silero VAD, Local Transport та центральним Pipeline. Для розпізнавання мовлення Pipeline звертається по HTTP до Elevenlabs STT Service, для обробки контексту — до Open AI LLM Service, а для синтезу мови — до Deepgram TTS Service. Паралельно з цим,бізнес-логіка Voice Agent Service синхронно викликає CRM Manager module у складі CRM Manager Post-processing service для запису основних даних про дзвінки. По завершенню дзвінка, Voice agent service публікує доменну подію call completed у чергу BullMQ, що працює на базі Redis, забезпечуючи асинхронний канал обміну.

CRM Manager Post-processing service побудовано за підходом модульного моноліту: Core Domain реалізовано в CRM Manager module, Supporting Domain – у Post-processing module. Перший синхронно обробляє запити від Voice Agent Service і BFF, другий виступає отримувачем повідомлень із BullMQ, аналізує транскрипти та генерує резюме. Завдяки модульному підходу та чіткому розподілу на піддомени DDD знижуються ризики: Core Domain ізольовано від допоміжної логіки, а Supporting Domain можна мінімізувати або виділити в окремий сервіс пізніше.

Таким чином, синхронні HTTP-канали забезпечують оперативну взаємодію з користувачем та критичними сервісами (автентифікація, STT, TTS, LLM, CRM), а асинхронний BullMQ-Redis об’єднує голосового агента та модуль пост-обробки без затримок основного потоку. Модульний моноліт із DDD-розмежуванням скорочує область складності та спрощує еволюцію системи.

3 ОПИС ФУНКЦІОНУВАННЯ, ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

3.1 Стратегія покриття тестами

У процесі розробки програмного забезпечення особлива увага приділяється забезпеченню його якості та надійності. Одним із ключових інструментів для досягнення цієї мети є тестування. У проекті використано комбінований підхід, що включає ручне тестування, інтеграційні та юніт-тести, кожен з яких виконує свою специфічну роль у загальній стратегії тестування.

Інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами системи. У контексті даного дипломного проєкту, інтеграційні тести застосовуються для перевірки коректності роботи репозиторіїв з базою даних або іншими зовнішніми сервісами. Такий підхід дозволяє переконатися, що шари доступу до даних функціонують належним чином в умовах, максимально наближених до реальних. Було прийняте свідоме рішення відмовитися від використання моків для репозиторіїв під час тестування сценаріїв використання, оскільки на меті було тестувати справжню інтеграцію компонентів, а не їхню ізольовану поведінку з імітованими залежностями. Це забезпечує вищу достовірність результатів тестування та допомагає виявляти проблеми на стиках модулів.

Юніт-тестування фокусується на перевірці окремих, найменших частин коду (юнітів), таких як функції, методи або класи, за умов ізоляції від решти системи. Під час тестування цей тип тестів застосовується переважно для перевірки складної бізнес-логіки, яка може бути інкапсульована в окремих модулях чи сервісах. Приклади таких тестів можна знайти у директорії crm-manager/src/modules/crm/__tests__/unit. Використання юніт-тестів для таких цілей дозволяє детально перевірити коректність алгоритмів та логічних конструкцій, забезпечуючи їхню надійність та передбачуваність.

Такий підхід до тестування (інтеграційні тести для перевірки взаємодії компонентів та взаємодії з реальними залежностями, а також юніт-тести для ізольованої перевірки складної логіки) дозволяє досягти комплексного покриття коду та забезпечити високу якість кінцевого продукту. Це допомагає виявляти помилки на ранніх етапах розробки, знижувати ризики та підвищувати стабільність системи в цілому.

3.2 Опис основних тестових кейсів та результатів тестування

Таблиця 3.1

Опис перевірених тестових кейсів компонентів системи з використання мануального підходу до тестування

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
Voice Agent	Обробка голосових запитів на оформлення бронювання нерухомості	Агент коректно розпізнає намір користувача забронювати нерухомість, збирає необхідну інформацію (дати, тип нерухомості тощо) та ініціює процес бронювання через CRM API
Voice Agent	Агент повинен надавати покрокові інструкції під час дистанційного заселення користувача на місце проживання	Агент надає чіткі та послідовні інструкції для самостійного заселення, використовуючи інформацію з бази знань нерухомості
Voice Agent	Агент має оновлювати записи у внутрішній CRM системі під час проведення дзвінка	Протягом розмови агент фіксує ключову інформацію та оновлює відповідні записи в CRM системі через API

Продовження таблиці 3.1

Опис перевірених тестових кейсів компонентів системи з використання мануального підходу до тестування

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
CRM Manager	Адміністратор повинен мати можливість редагувати список нерухомості	Інтерфейс адміністратора CRM надає функціонал для додавання, редагування та видалення об'єктів нерухомості, їхніх описів, фотографій, цін та іншої пов'язаної інформації
CRM Manager	Адміністратор повинен мати можливість завантажувати текстові інструкції для обробки інформаційних та екстрених запитів до бази знань нерухомості	Інтерфейс адміністратора CRM дозволяє завантажувати та керувати текстовими інструкціями для кожного об'єкта нерухомості

3.3 Автоматизовані тести

У модулі crm-manager реалізовано два основних типи автоматизованих тестів, які забезпечують перевірку коректності роботи системи на різних рівнях:

– Інтеграційні тести розташовані в директорії /src/modules/crm/__tests__/integration/. Ці тести призначені для перевірки взаємодії між компонентами системи, зокрема, коректності роботи репозиторіїв із Twenty CRM API. Основна мета цих тестів – переконатися, що операції збереження, пошуку, оновлення та видалення даних виконуються належним чином, а маппінг даних між сутностями домену та моделями представлення даних CRM працює коректно. Тести охоплюють функціональність репозиторіїв для управління проживаннями (TwentyCrmAccommodationsRepository), дзвінками (TwentyCrmCallsRepository), клієнтами (TwentyCrmClientsRepository) та об'єктами оренди (TwentyCrmRentalsRepository).

Кожен набір тестів перевіряє CRUD-операції (Create, Read, Update, Delete), а також специфічні для сутності методи пошуку та обробки граничних випадків, таких як запити до неіснуючих записів.

– Юніт-тести знаходяться в директорії /src/modules/crm/__tests__/unit/. Ці тести сфокусовані на перевірці ізольованих частин коду, зокрема, складної бізнес-логіки, реалізованої в доменних сутностях або сервісах. Наприклад, тестується логіка визначення вільних проміжків для бронювання та коректність роботи з діапазонами дат. Юніт-тести використовують моки для залежностей, що дозволяє точно перевірити поведінку конкретного модуля без впливу зовнішніх факторів.

Таблиця 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmAccommodationsRepository (Integration test)	Створення нового запису про проживання (save)	Запис успішно створюється в базі даних, повертається унікальний ідентифікатор створеного запису
TwentyCrmAccommodationsRepository (Integration test)	Пошук проживання за ідентифікатором (findById)	Повертається об'єкт проживання з усіма відповідними полями, якщо запис існує; в іншому випадку повертається null
TwentyCrmAccommodationsRepository (Integration test)	Пошук проживань за ідентифікатором оренди (findByRentalId)	Повертається список проживань, пов'язаних із зазначеним ідентифікатором оренди. Кожен запис у списку має ідентифікатор
TwentyCrmAccommodationsRepository (Integration test)	Отримання всіх записів про проживання (findAll)	Повертається масив усіх наявних записів про проживання. Кожен запис у списку має ідентифікатор
TwentyCrmAccommodationsRepository (Integration test)	Видалення запису про проживання (delete)	Запис успішно видаляється з бази даних. Повторний пошук за ідентифікатором видаленого запису повертає null

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmAccommodationsRepository (Integration test)	Обробка помилки 404 при видаленні неіснуючого проживання	Спроба видалення запису з неіснуючим ID не викликає помилку та завершується коректно
TwentyCrmAccommodationsRepository (Integration test)	Обробка помилок при створенні запису з невалідними даними	Спроба створити запис з неповними або некоректними даними призводить до виникнення помилки
TwentyCrmAccommodationsRepository (Integration test)	Обробка помилок під час операцій оновлення (зі збоєм на рівні API)	Спроба оновити запис, коли внутрішній виклик API для оновлення завершується помилкою, призводить до відповідної помилки оновлення
TwentyCrmAccommodationsRepository (Integration test)	Обробка помилки 404 при пошуку неіснуючого проживання за ID	Пошук за неіснуючим ID повертає null без виникнення помилок
TwentyCrmAccommodationsRepository (Integration test)	Пошук останнього проживання за ідентифікатором клієнта (findMostRecentByClientId)	Повертається найновіший (за датою початку бронювання) запис про проживання для вказаного клієнта

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmAccommodationsRepository (Integration test)	Повернення null при пошуку останнього проживання для клієнта без проживань	Якщо для вказаного клієнта немає жодного запису про проживання, пошук повертає null
TwentyCrmCallsRepository (Integration test)	Створення нового запису про дзвінок (save)	Запис про дзвінок успішно створюється, повертається унікальний ідентифікатор
TwentyCrmAccommodationsRepository (Integration test)	Повернення null при пошуку останнього проживання для клієнта без проживань	Якщо для вказаного клієнта немає жодного запису про проживання, пошук повертає null
TwentyCrmCallsRepository (Integration test)	Пошук дзвінка за ідентифікатором з усіма полями (findById)	Повертається об'єкт дзвінка з усіма полями (номер абонента, тип, транскрипт, ID клієнта, ID пов'язаної оренди, ID аудіофайлу, резюме), якщо запис існує. В іншому випадку повертається null
TwentyCrmCallsRepository (Integration test)	Отримання всіх записів про дзвінки (findAll)	Повертається масив усіх наявних записів про дзвінки. Кожен запис у списку має ідентифікатор

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmCallsRepository (Integration test)	Видалення запису про дзвінок (delete)	Запис успішно видаляється. Повторний пошук за ідентифікатором повертає null
TwentyCrmCallsRepository (Integration test)	Обробка помилки 404 при пошуку неіснуючого дзвінка	Пошук за неіснуючим ID повертає null
TwentyCrmCallsRepository (Integration test)	Обробка помилок при створенні запису дзвінка з невалідними даними	Спроба створити запис з неповними або некоректними даними призводить до виникнення помилки
TwentyCrmClientsRepository (Integration test)	Створення нового клієнта (save)	Запис про клієнта успішно створюється, повертається унікальний ідентифікатор
TwentyCrmClientsRepository (Integration test)	Пошук клієнта за ідентифікатором (findById)	Повертається об'єкт клієнта з усіма відповідними полями, якщо запис існує; в іншому випадку повертається null
TwentyCrmClientsRepository (Integration test)	Пошук клієнта за номером телефону (findByPhoneNumber)	Повертається об'єкт клієнта, пов'язаний із зазначеним номером телефону, якщо такий існує

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmClientsRepository (Integration test)	Отримання всіх клієнтів (findAll)	Повертається масив усіх наявних записів клієнтів. Кожен запис у списку має ідентифікатор
TwentyCrmClientsRepository (Integration test)	Видалення клієнта (delete)	Запис успішно видаляється. Повторний пошук за ідентифікатором повертає null
TwentyCrmClientsRepository (Integration test)	Обробка помилки 404 при пошуку неіснуючого клієнта	Пошук за неіснуючим ID повертає null
TwentyCrmClientsRepository (Integration test)	Обробка помилки 404 при видаленні неіснуючого клієнта	Спроба видалення запису з неіснуючим ID не викликає помилку
TwentyCrmRentalsRepository (Integration test)	Створення нового запису про оренду (save)	Запис про оренду успішно створюється, повертається унікальний ідентифікатор

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
TwentyCrmRentalsRepository (Integration test)	Пошук оренди за ідентифікатором (findById)	Повертається об'єкт оренди з усіма відповідними полями, якщо запис існує; в іншому випадку повертається null
TwentyCrmRentalsRepository (Integration test)	Отримання всіх записів про оренду (findAll)	Повертається масив усіх наявних записів про оренду. Кожен запис у списку має ідентифікатор
TwentyCrmClientsRepository (Integration test)	Обробка помилок при створенні запису клієнта з невалідними даними	Спроба створити запис з неповними або некоректними даними призводить до виникнення помилки
TwentyCrmRentalsRepository (Integration test)	Видалення запису про оренду (delete)	Запис успішно видаляється. Повторний пошук за ідентифікатором повертає null
TwentyCrmRentalsRepository (Integration test)	Обробка помилки 404 при пошуку неіснуючої оренди	Пошук за неіснуючим ID повертає null
TwentyCrmRentalsRepository (Integration test)	Обробка помилок при створенні запису оренди з невалідними даними	Спроба створити запис з неповними або некоректними даними призводить до виникнення помилки

Продовження таблиці 3.2

Опис перевірених тестових кейсів компонентів системи покритих автоматизованими тестами

Компонент, що тестується	**Тестовий кейс**	**Очікуваний результат**
Rental accommodations getFreeDaysSpansInRangeIncluding (Unit test)	Коректне визначення вільних проміжків дат між бронюваннями	Метод повертає правильні проміжки вільних днів між існуючими бронюваннями в межах заданого діапазону дат
Rental accommodations (Unit test)	Повернення порожнього списку вільних проміжків, якщо весь період зайнятий	Метод getFreeDaysSpansInRangeIncluding повертає порожній масив, якщо весь вказаний діапазон дат повністю покритий бронюваннями
DaysSpan (Unit test)	Створення валідного об'єкта DaysSpan	Статичний метод create успішно створює екземпляр DaysSpan з коректними датами початку та кінця
DaysSpan (Unit test)	Викидання помилки, якщо дата кінця передує даті початку	Спроба створити DaysSpan з датою кінця, що передує даті початку, призводить до виникнення помилки
DaysSpan (Unit test)	Викидання помилки, якщо дати початку та кінця однакові	Спроба створити DaysSpan з однаковими датами початку та кінця призводить до виникнення помилки

Для написання та виконання як інтеграційних, так і юніт-тестів у модулі crm-manager, використовується сучасний тестовий фреймворк vitest. Його перевагами є висока швидкість виконання тестів, нативна підтримка TypeScript, стандартизований API для створення моків (наприклад, vi.fn() для функцій та vi.mock() для модулів), а також сумісність із конфігурацією Vite, що спрощує налаштування тестового середовища. Використання vitest дозволяє ефективно організовувати тестові набори, описувати тестові випадки (it) та групи тестів (describe), а також використовувати перевірки (expect) для підтвердження відповідності фактичних результатів очікуваним.

3.4 Особливості мануального тестування голосового асистента

Голосові асистенти потребують оцінки якості через призму людського сприйняття, оскільки автоматизовані метрики не охоплюють усі аспекти взаємодії. Мануальне тестування має ключову роль при перевірці нефункціональних вимог, пов’язаних із якістю комунікації, де автоматизовані інструменти не здатні імітувати когнітивні процеси користувача.

Рис. 3.1 Інфографіка гучності звуку в співставленні з його чинником

Для верифікації вимоги WER ≤10% при рівні шуму ≤60 дБ, що приблизно відповідає гучності розмови двох людей чи рівень шуму в ресторані було використано відео з відкритих джерел з відповідними джерелами шуму (вуличний гомін, офісні діалоги). Тестувальники відтворюють фрази з контрольованою вимовою, а система має їх розпізнати. Ця методика дозволяє виявити деградацію якості розпізнавання при пороговому значенні 60 дБ, що відповідає типовому середовищу в сфері гостинності.

Вимірювання затримки відповіді в рамках мануального тестування передбачає аналіз системних журналів (логів). Тестувальник, виконуючи контрольні діалоги, фіксує час від завершення власної репліки до початку відтворення відповіді асистентом. Ці дані з логів дозволяють перевірити дотримання нефункціональної вимоги щодо максимальної затримки, що не повинна перевищувати 1000 мс, для окремих взаємодій. Такий підхід, на відміну від автоматизованих навантажувальних тестів, дозволяє оцінити затримку в контексті конкретного діалогового сценарію та умов тестування.

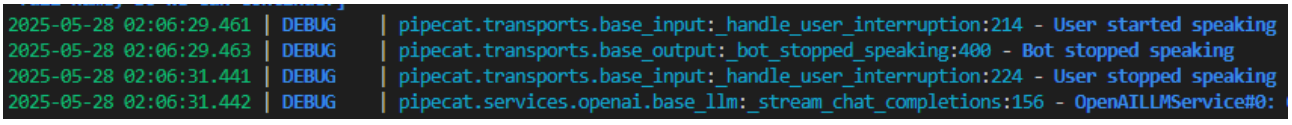


Рис. 3.2 Логи голосового асистента завершення репліки користувачем

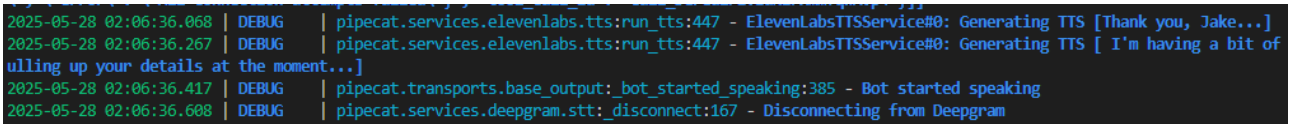


Рис. 3.3 Логи голосового асистента початку відповіді на репліку користувача

В аспекті лінгвістичної експертизи, команда тестувальників має включати фахівців, які глибоко розуміють специфіку підтримуваних мов – української та англійської. Це критично для адекватної оцінки якості розпізнавання та синтезу мовлення. Для об’єктивної верифікації вимоги щодо рівня помилок розпізнавання мовлення (WER<10%), тестувальники з різними

артикуляційними особливостями та темпом мовлення (наприклад, у діапазоні 120–180 слів за хвилину) проводять серії запитів. Це дозволяє сформувати уявлення про ефективність системи ASR для репрезентативної вибірки голосових профілів і співставити отримані показники WER із цільовим значенням.

Ключова роль мануального тестування полягає в оцінці тих параметрів якості взаємодії, які складно або неможливо повністю формалізувати технічними метриками. Зокрема, тестувальники оцінюють натуральність пауз у мовленні асистента, які мають сприяти відчуттю живого діалогу; адекватність емоційного забарвлення синтезованого голосу, що може проявлятися в інтонаційному виділенні ключових слів (наприклад, при обробці термінових запитів); а, також, здатність системи підтримувати контекст розмови – наприклад, відновлювати діалог після нетривалих перерв. Саме ці, значною мірою суб’єктивні, чинники визначають фінальне сприйняття користувачем комфорту та загальної якості взаємодії з голосовим асистентом, доповнюючи формальні показники продуктивності та точності.

3.5 Альтернативні способи тестування голосового асистента

Оцінка голосового асистента виходить за межі традиційних методів функціонального тестування: аналіз якості діалогів, продуктивність відповіді та стійкість моделі вимагають більш гнучких, контекстно-чутливих підходів. Симуляція взаємодії двох великих мовних моделей відкриває можливість автоматизованого скринінгу промптів із подальшим скорингом за допомогою reasoning-моделей та інструментів class checking calls. Разом з тим, паралельне впровадження системного моніторингу затримок через аналіз логів Voice agent service дозволяє виявити деградацію продуктивності у реальному часі. Інтеграція платформ на кшталт Langfuse для трасування LLM-запитів або Convex для тестування справності елементів голосового pipeline сприяє глибшому розумінню поведінки асистента та підвищенню рівня спостережності

системи. Такий комплексний підхід поєднує можливості мовних моделей із сучасними засобами моніторингу, розширюючи межі достовірності та оперативності тестування.

Однією з перспективних методик є симуляція діалогу між двома інстанціями великих мовних моделей. У ході такої імітації перша модель генерує запити голосового користувача, тоді як друга виступає в ролі асистента. Діалогові логи передаються до промислових моделей для reasoning та class checking calls, які автоматично аналізують кожен обмін репліками за заданими критеріями — семантичною коректністю, логічною зв’язністю або відповідністю тональності. Результатом стає скорингова функція, що повертає числову чи категоріальну оцінку якості відповіді асистента. Такий підхід дозволяє швидко виявити систематичні помилки в обробці запитів, дискримінаційні або неточні відповіді, а також оцінити здатність асистента підтримувати заданий стиль комунікації.

Окрім оцінки змісту, важливо контролювати час реакції системи. Для цього налаштовується автоматичний збір і аналіз логів Voice Agent Service із вилученням часових міток ключових подій — завершення промови користувача, після чого відбувається відтворення відповіді асистента. За допомогою скриптів або спеціалізованих бібліотек, ці мітки співставляються із нормативним інтервалом у 1000 мс. Результати попереднього аналізу автоматично агрегуються у звіти про середню затримку, перцентилі та максимальні значення. Такий моніторинг у реальному часі дозволяє виявити деградацію продуктивності на ранніх етапах і забезпечити своєчасне масштабування або переналаштування сервісів.

Для глибшого розуміння поведінки асистента доцільно інтегрувати спеціалізовані платформи спостережності. Зокрема, Langfuse може використовуватися для детального трасування LLM-запитів та відповідей, візуалізації метрик якості промптів і версійного контролю змін у логіці генерації.

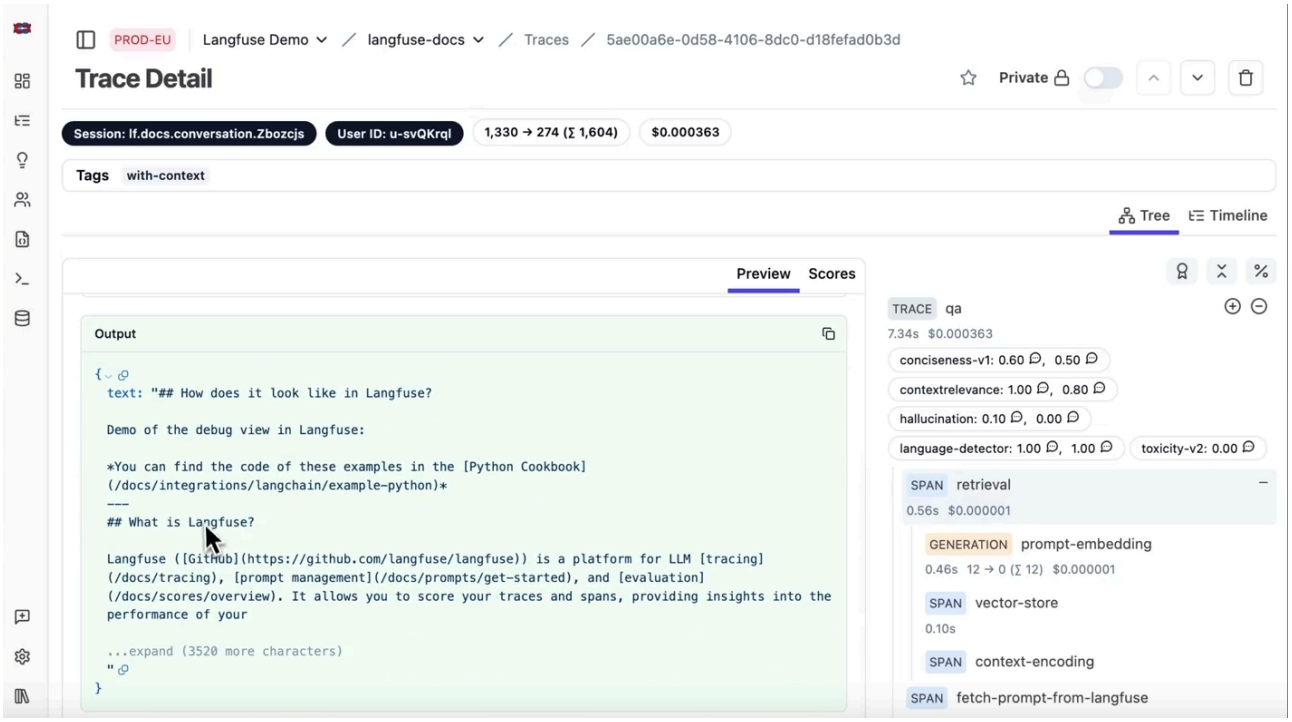


Рис. 3.4 Знімок екрану трасування в Langfuse

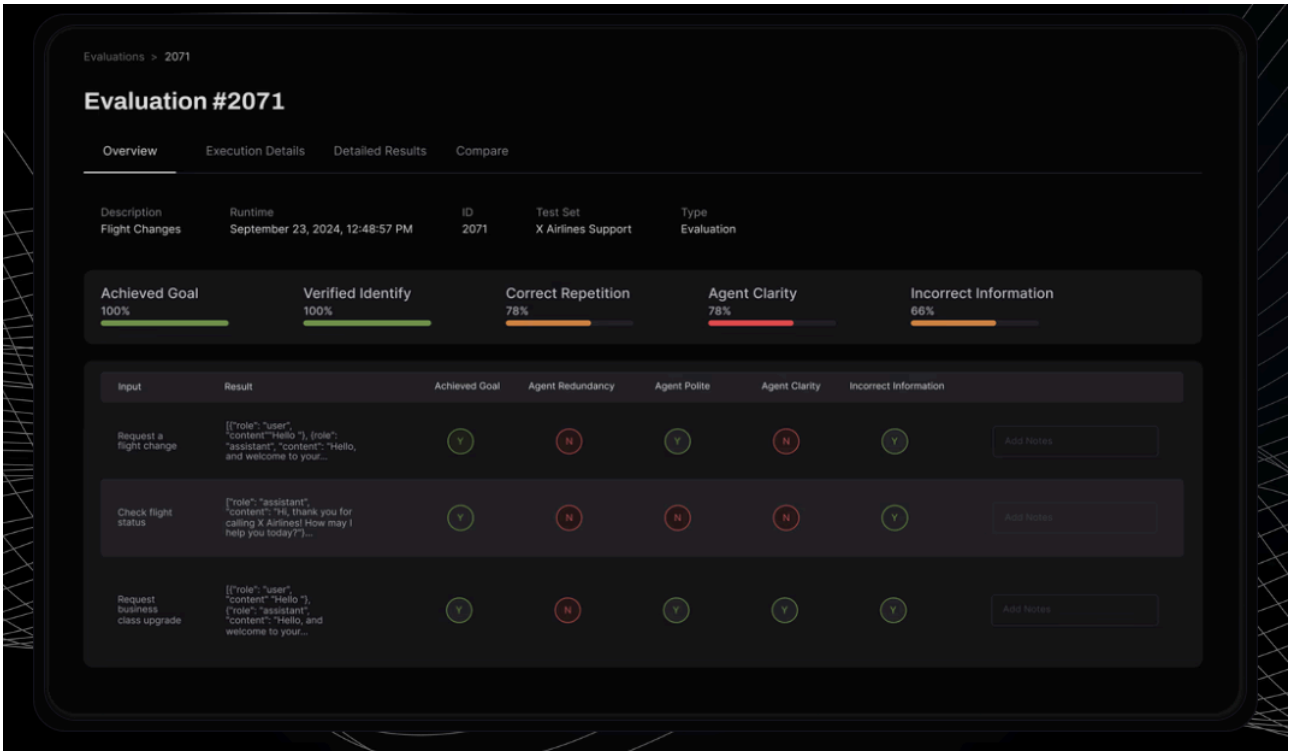


Рис. 3.5 Демонстраційний варіант результатів тестування через Coval

Аналогічно, Coval як сервіс тестування дає змогу проводити load-тести голосових потоків та перевіряти коректність обробки сценаріїв в голосовому

режимі. Завдяки такому поєднанню власного моніторингу та зовнішніх рішень досягається цілісний огляд роботи системи, що значно спрощує діагностику, розробку та тестування.

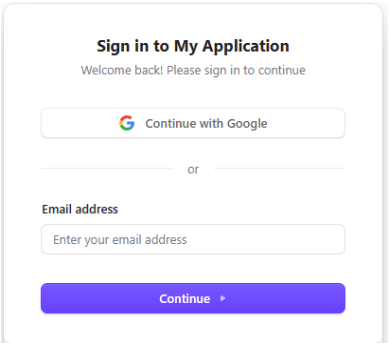
3.5 Опис реалізованого функціоналу та графічного інтерфейсу Calls dashboard

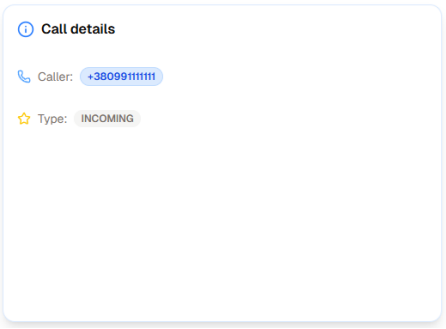
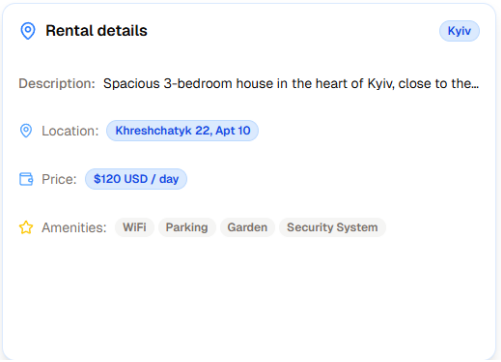
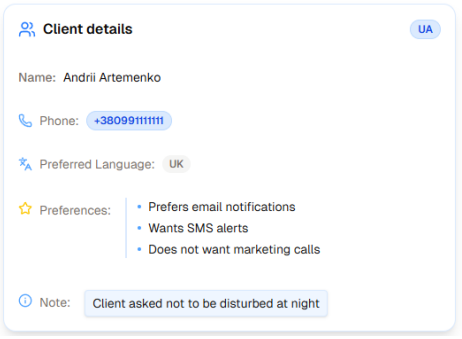
Calls dashboard є спеціалізованим інтерфейсом, призначеним для перегляду детальної інформації про завершені телефонні дзвінки, їх транскрипцій, резюме та прослуховування аудіозаписів. Доступ до цієї панелі здійснюється безпосередньо за посиланням із системи управління взаємовідносинами з клієнтами Twenty CRM, що забезпечує зручну навігацію між системами та швидкий доступ до контекстної інформації про дзвінки.

Візуальна структура та основні функціональні блоки Calls dashboard представлені у вигляді таблиці, що ілюструє ключові елементи інтерфейсу на відповідних знімках екрану.

Таблиця 3.3

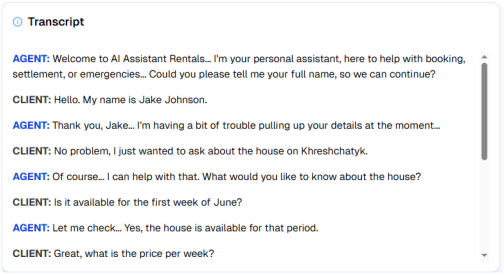
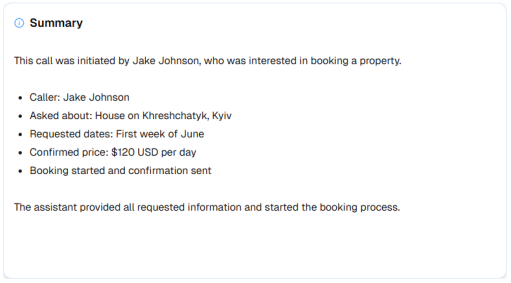
Опис функціоналу UI компонентів з графічним представленням

Назва UI компоненту	**Знімок екрану**	**Опис функціоналу**
Форма авторизації в систему	 Рис. 3.6 Форма авторизації в системі	Містить поля вводу для адреси електронної пошти та паролю

Назва UI компоненту	**Знімок екрану**	**Опис функціоналу**
Деталі дзвінка	 Рис. 3.6 Деталі дзвінка	Відображає основні метадані конкретного телефонного дзвінка, зокрема номер телефону ініціатора дзвінка та його тип (наприклад, вхідний)
Інформація про об'єкт оренди	 Рис. 3.8 Інформація про об'єкт оренди	Надає детальні відомості про об'єкт нерухомості, якого стосувався дзвінок або з яким пов'язаний клієнт.
Деталі клієнта	 Рис. 3.7 Деталі клієнта	Містить ключову інформацію про клієнта, пов'язаного з дзвінком. Включає такі дані, як ім'я та прізвище клієнта, контактний номер телефону та додаткові примітки щодо клієнта

Продовження таблиці 3.3

Опис функціоналу UI компонентів з графічним представленням

Назва UI компоненту	**Знімок екрану**	**Опис функціоналу**
Аудіо плеєр	Рис. 3.9 Аудіо плеєр	Інтерфейсний елемент для керування відтворенням аудіо
Транскрипція розмови	 Рис. 3.10 Транскрипція розмови	Представляє повну текстову версію записаної телефонної розмови
Резюме дзвінка	 Рис. 3.11 Резюме дзвінка	Відображає стислий виклад змісту розмови, автоматично сформований системою постобробки

Кожен із зазначених блоків є компонентом інтерфейсу, розробленим з метою забезпечення максимальної інформативності та зручності для користувача при аналізі даних голосової взаємодії.

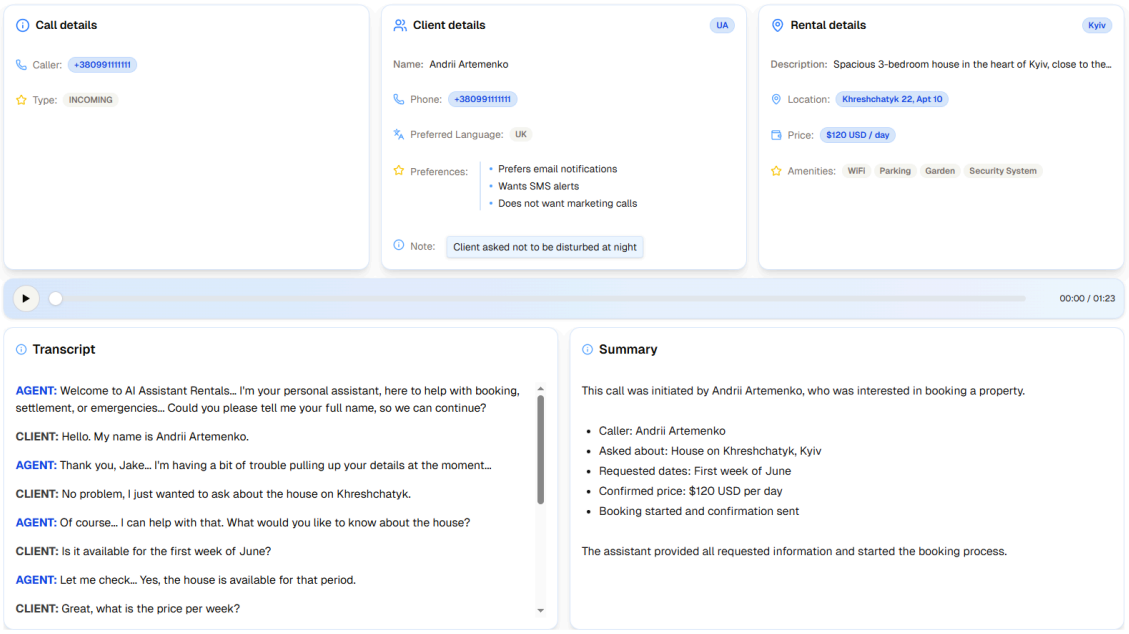


Рис. 3.12 Екранна форма мікрофронтенду Calls dashboard

3.6 Опис функціоналу та графічного інтерфейсу Twenty CRM

Система Twenty CRM слугує централізованим сховищем та інструментом управління даними, що стосуються клієнтів, об'єктів нерухомості, бронювань та пов'язаних з ними взаємодій – зокрема, телефонних дзвінків. Інтерфейс системи організований для забезпечення ефективного доступу та управління цими даними.

Дані клієнтів представлені у форматі таблиці, де кожен рядок відповідає окремому клієнту. Для кожного клієнта відображається набір ключових атрибутів, таких як повне ім'я (розділене на ім'я, прізвище та по батькові), контактний номер телефону, дати створення та останнього оновлення запису, а також інформація про користувача, який створив або оновив запис.

Доступні базові операції для навігації по списку та керування представленням даних – зокрема, можливість фільтрації та сортування записів за різними атрибутами.

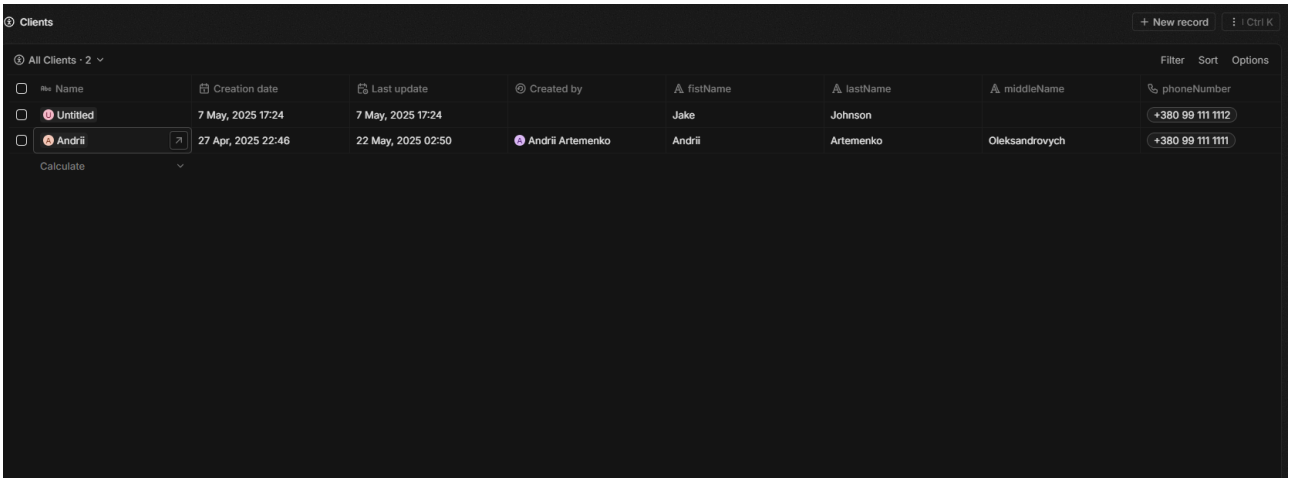


Рис. 3.13 Перегляд та управління списком клієнтів

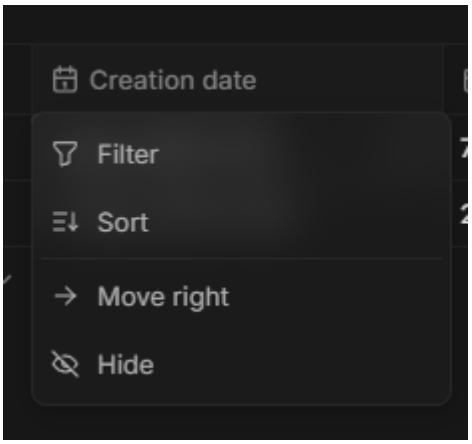


Рис. 3.14 Меню фільтрування та сортування записів в таблиці

Детальна інформація про конкретний об'єкт оренди представлена у вигляді картки або форми, що містить структуровані блоки даних. Відображаються вичерпні відомості про об'єкт – включно з його назвою, переліком зручностей (у форматі тегів або списку), опису, місцезнаходження, ціни за добу, інструкцією для надзвичайних ситуацій, інструкцією щодо заселення, датою останнього оновлення.

Список усіх бронювань представлений у табличному форматі. Кожен запис бронювання містить такі атрибути, як дати створення та останнього оновлення, інформацію про користувача, який створив запис, дати початку та завершення періоду оренди, а, також, поточний статус бронювання.

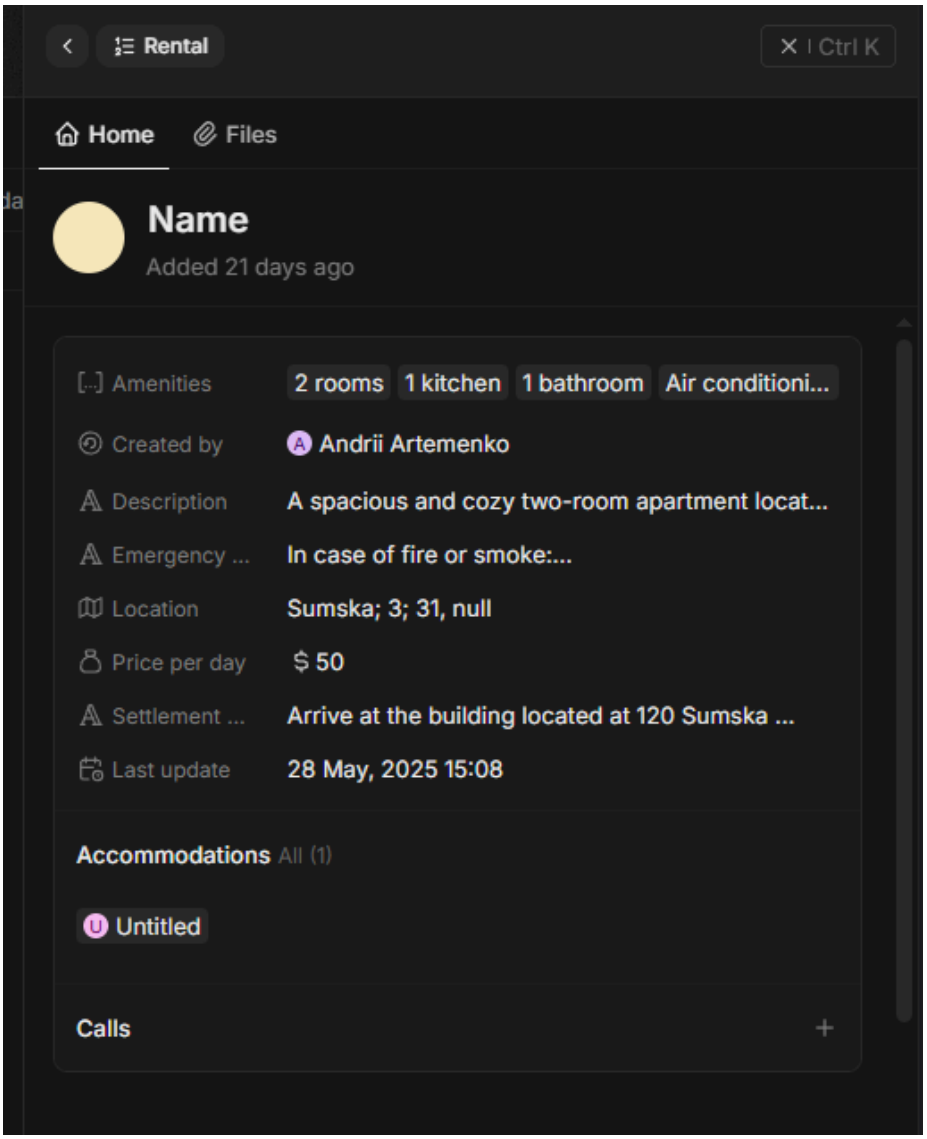


Рис. 3.15 Перегляд та управління окремим записом

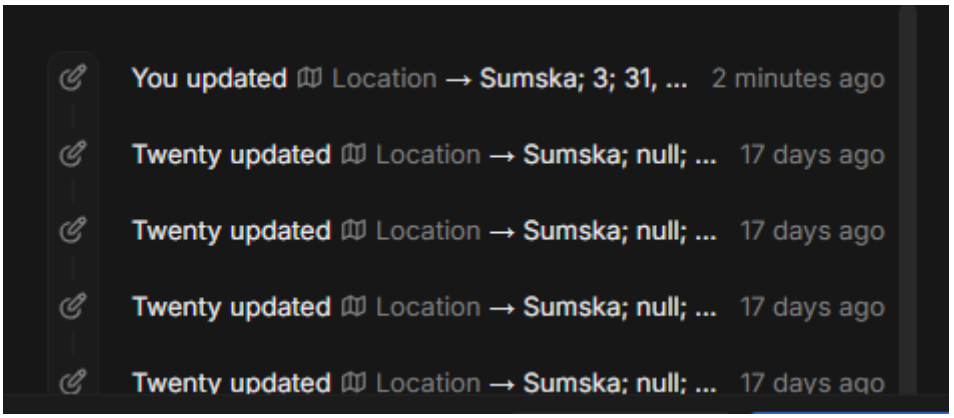


Рис. 3.16 Журнал активності

Для всіх сутностей доступний перегляд історії змін (журналу активності). Журнал фіксує інформацію про внесені зміни, вказуючи, хто і коли здійснив зміну, а також сутність та атрибут, що були модифіковані. Надає можливість відстежувати історію роботи з конкретними записами даних, що корисно для аудиту та розуміння еволюції даних.

Інтерфейс Twenty CRM реалізує патерни, типові для систем управління даними, забезпечуючи структуроване представлення інформації та інструменти для навігації, пошуку та базового управління записами.

3.7 Демонстрація функціоналу реєстрації нових співробітників в системі

Ефективне функціонування системи голосового асистента вимагає чіткої ідентифікації та авторизації співробітників, які взаємодіють з різними компонентами. Зокрема, співробітники потребують доступу до Twenty CRM для управління даними клієнтів та бронювань, а також до Calls dashboard для аналізу деталей голосових взаємодій. Архітектурне рішення про розподіл цих функцій між незалежними системами зумовлює специфічний підхід до управління користувачами. Процес реєстрації має двоетапний характер, оскільки кожна система підтримує власну модель управління користувачами: Twenty CRM використовує внутрішню систему workspace-орієнтованої авторизації, тоді як Calls dashboard інтегровано з сервісом Clerk для гарантування безпечного доступу до конфіденційних даних дзвінків.

3.7.1 Реєстрація співробітників у системі Twenty CRM

Користувачі Twenty CRM являють собою співробітників, які безпосередньо працюють з операційними даними – клієнтською інформацією, бронюваннями нерухомості та супутніми сервісами. Система надає адміністраторам комплексний інтерфейс управління користувачами через розділ

"Workspace / Members", де централізовано контролюється доступ до робочого простору організації.

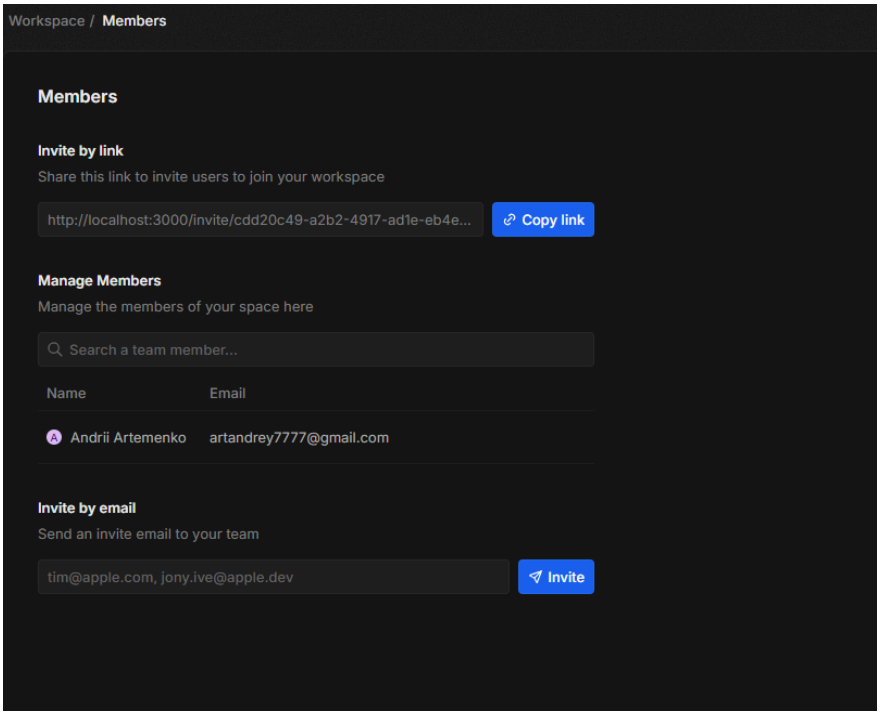


Рис. 3.17 Розділ "Workspace / Members" Twenty CRM

Процес запрошення нового співробітника може здійснюватися двома способами. Перший спосіб передбачає використання унікального посилання запрошення, яке генерується системою та може поширюватися серед потенційних користувачів. Це посилання дозволяє кандидатам самостійно приєднатися до робочого простору. Другий спосіб базується на персоналізованих email-запрошеннях, де адміністратор вводить конкретні email-адреси у відповідне поле і надсилає індивідуальні запрошення.

3.7.2 Реєстрація співробітників для доступу до Calls dashboard через Clerk

У системі Calls dashboard автентифікація та управління обліковими записами здійснюються через панель Clerk, де кожного нового користувача необхідно створювати вручну. Адміністратор переходить у розділ Users,

натискає кнопку Create user – після чого відкривається модальне вікно із двома вкладками: Create User та Invite User. Оскільки в контексті даного дипломного проекту застосовується запрошувальний підхід, обирається вкладка Invite User, куди вводиться електронна адреса нового співробітника. За потреби, можна також встановити термін дії запрошення. Після натискання Invite user, Clerk відправляє лист із посиланням на завершення реєстрації, а створений обліковий запис з'являється у списку користувачів.

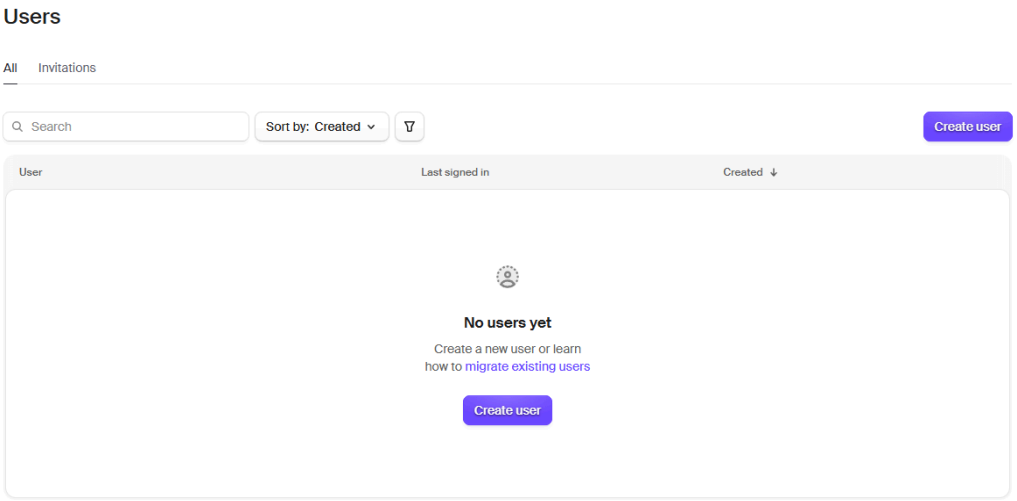


Рис. 3.18 Розділ Users в Clerk

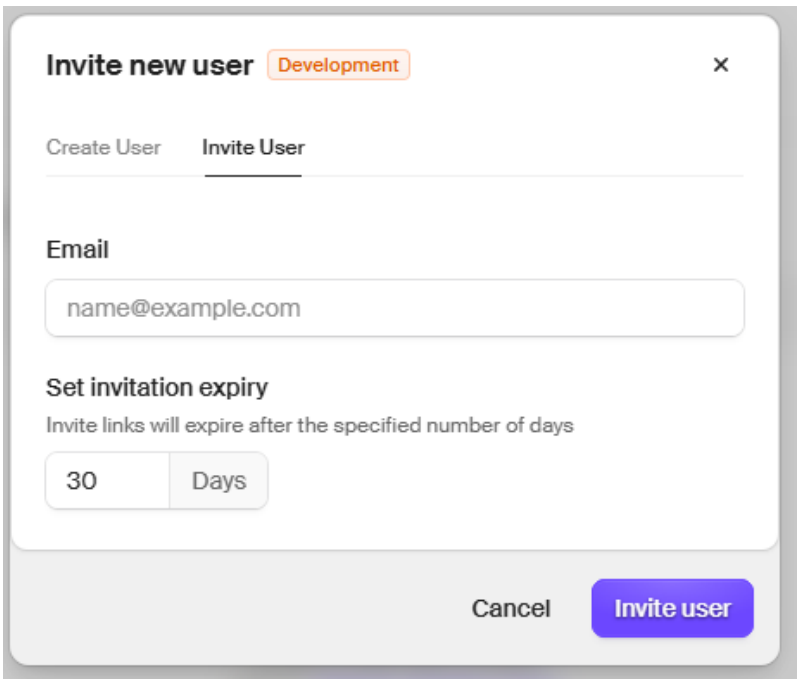


Рис. 3.19 Модальне вікно запрошення нового користувача

Архітектурне рішення щодо двоетапної реєстрації відображає принцип розділення відповідальності між різними компонентами системи. Для забезпечення повноцінної роботи з усім функціоналом системи голосового асистента співробітник має створити облікові записи в обох системах незалежно.

3.8 Підготовка конфігурації для розгортання. Отримання API ключів для TTS/STT/LLM провайдерів

Процес налаштування охоплює два ключові аспекти: внутрішні параметри системи, такі як мовні налаштування для забезпечення багатомовної підтримки, та зовнішні залежності у вигляді автентифікованого доступу до API провайдерів. Правильна конфігурація забезпечує не лише технічну інтеграцію компонентів, але й дотримання принципів інформаційної безпеки та оптимізацію використання зовнішніх ресурсів.

3.8.1 Отримання API-ключів для TTS/STT/LLM провайдерів

API-ключі представляють собою унікальні ідентифікатори або токени, що функціонують як цифрові облікові дані для авторизації звернень до програмних інтерфейсів зовнішніх сервісів. У контексті голосового асистента, ці ключі виконують три фундаментальні функції. Автентифікація підтверджує дозвіл на надсилання запитів системи до API провайдерів, включаючи Deepgram для розпізнавання мовлення, Eleven Labs для синтезу голосу та OpenAI для обробки природної мови. Контроль доступу обмежує спектр операцій, які система може виконувати через API, відповідно до дозволів, асоційованих з конкретним ключем. Моніторинг та облік використання дозволяють провайдерам відстежувати споживання ресурсів, застосовувати тарифні плани та виявляти аномальні паттерни використання.

Генерація API-ключів здійснюється через панелі управління відповідних провайдерів з дотриманням стандартизованої процедури створення та налаштування параметрів доступу. Процес ініціюється через інтерфейс управління ключами, де адміністратор обирає опцію створення нового ключа. Наступний етап передбачає надання ідентифікатора або дружнього імені ключа для полегшення подальшого управління та ідентифікації у списку активних ключів.

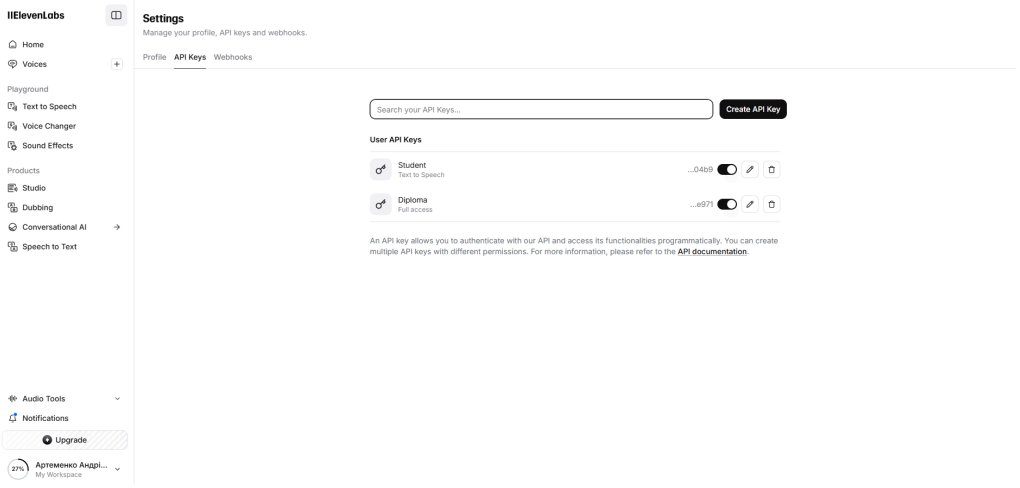


Рис. 3.20 Dashboard керування API-ключами провайдера ElevenLabs

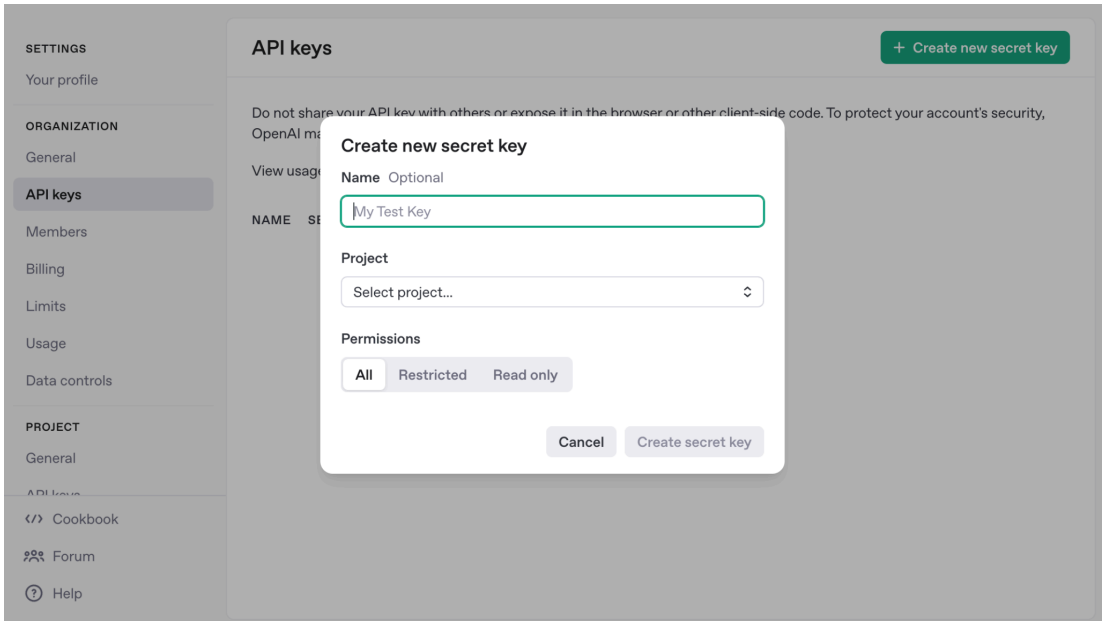


Рис. 3.21 Dashboard керування API-ключами провайдера Open AI

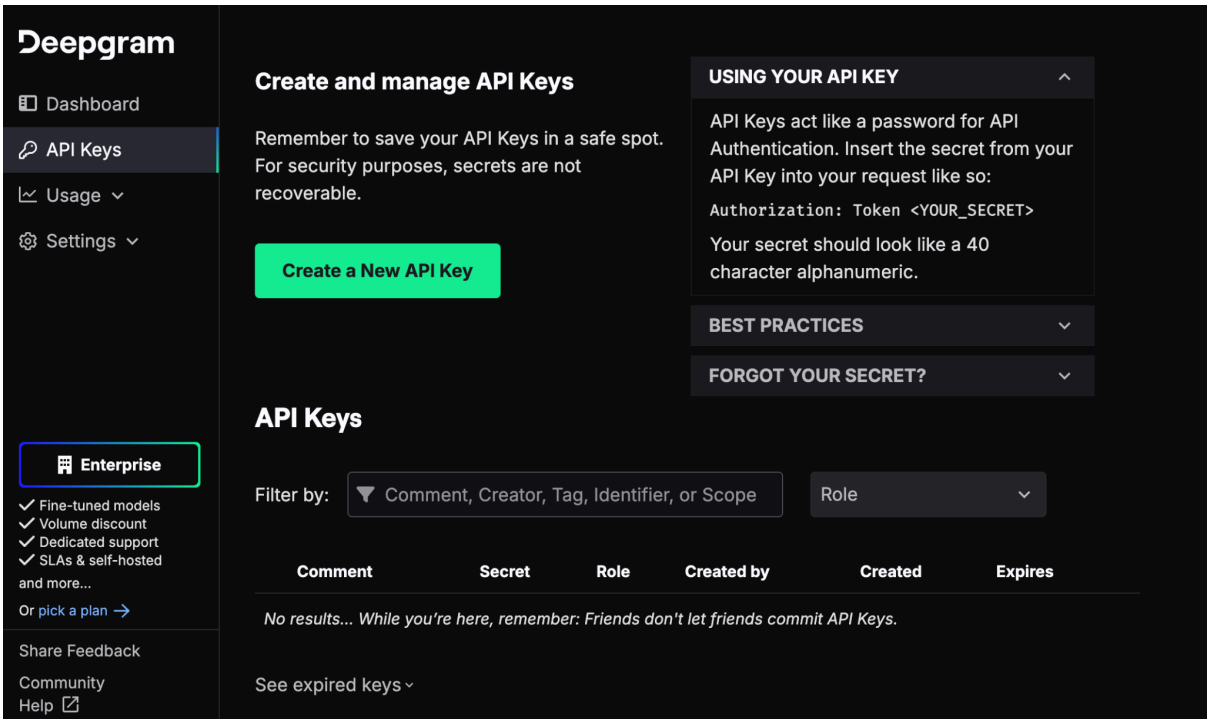


Рис. 3.22 Dashboard керування API-ключами провайдера Deepgram

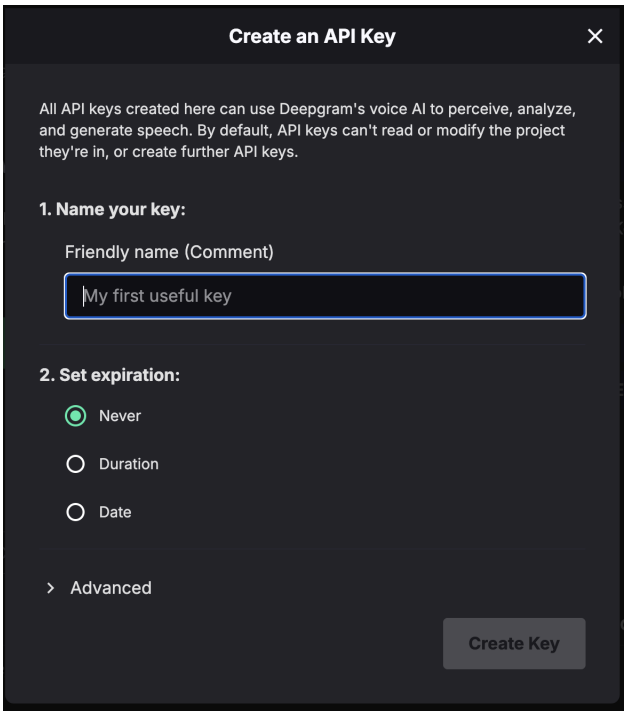


Рис. 3.23 Модальне вікно створення API-ключа для провайдера Deepgram

3.8.2 Управління конфігурацією за допомогою змінних середовища

Змінні середовища є стандартизованим механізмом зберігання конфігураційної інформації, що дозволяє системі адаптуватися до різних операційних середовищ без модифікації вихідного коду. Цей підхід особливо критичний для систем, що використовують чутливі дані автентифікації та потребують гнучкості розгортання.

Відокремлення конфігурації від коду забезпечує чистоту кодової бази та спрощує управління параметрами системи без необхідності модифікації програмного коду. Це дозволяє одному і тому ж коду функціонувати в середовищах розробки, тестування та production з різними наборами API-ключів та конфігураційних параметрів.

Аспект безпеки реалізується через уникнення збереження ключів доступу та інших конфіденційних даних безпосередньо у вихідному коді. Використання змінних середовища створює додатковий захисний шар між чутливими даними та публічно доступним кодом.

3.8.3 Конфігурація мовних параметрів

Багатомовна підтримка системи реалізується через динамічну конфігурацію мовних параметрів для всіх компонентів голосового pipeline, включаючи розпізнавання мовлення, синтез голосу та обробку природної мови. Це забезпечує коректне функціонування системи з українською та англійською мовами відповідно до законодавчих вимог та потреб міжнародних користувачів.

Конфігурація STT та TTS здійснюється через передачу специфічних мовних параметрів при зверненні до API Deepgram та Eleven Labs. Кожен запит містить код мови або ідентифікатор мовної моделі, що дозволяє провайдерам застосувати оптимізовані алгоритми для конкретної мови. Хоча деякі сучасні моделі підтримують автоматичне виявлення мови, явне вказування цільової мови забезпечує вищу точність розпізнавання та природність синтезу.

Конфігурація LLM передбачає гібридний підхід, де базові системні інструкції формулюються універсальною мовою (зазвичай англійською для

забезпечення максимальної ефективності моделі), але через додаткові контекстуальні повідомлення модель отримує вказівки щодо генерації відповідей цільовою мовою. Це дозволяє використовувати переваги англомовного навчання моделей при забезпеченні локалізованого виводу.

3.10 Розгортання мікросервісів за допомогою Docker

Контейнеризація за допомогою Docker забезпечує ізоляцію застосунків разом з усіма їхніми залежностями в легких, віртуальних середовищах виконання. Цей підхід особливо ефективний для систем голосових асистентів, де різні компоненти мають різноманітні технологічні залежності та вимоги до середовища виконання.

Docker Compose виступає інструментом для визначення та управління багатоконтейнерними застосунками, що дозволяє описувати складні архітектури через декларативні YAML-файли. Для проєкту голосового асистента цей інструмент забезпечує централізоване управління всіма сервісами та їх взаємозалежностями, значно спрощуючи процеси локальної розробки та тестування. Дослідження емпіричних методів роботи з Docker Compose підтверджують значне зменшення часу розробки та кількості помилок при визначенні конфігурацій контейнерів.

3.10.1 Використання Docker Compose для локальної розробки

Файл docker-compose.yaml (рис. 3.24) централізовано визначає базові сервіси проєкту, включаючи Localstack для емуляції AWS S3 та Redis для черг повідомлень. Конфігурація Localstack включає експозицію портів 4566 та 4572, налаштування змінних середовища для S3-сервісу та монтування томів для збереження даних. Redis контейнер налаштовано з портом 6379 та сховищем через volume redis-data, що забезпечує збереження стану черг повідомлень між перезапусками.

vag-microservices - docker-compose.yaml

```yaml
services:
  localstack:
    image: localstack/localstack:latest
    container_name: localstack_s3
    ports:
      - '4566:4566'
      - '4572:4572'
    environment:
      - SERVICES=s3
      - DEBUG=1
      - DATA_DIR=/var/lib/localstack/data
      - AWS_ACCESS_KEY_ID=test
      - AWS_SECRET_ACCESS_KEY=test
    volumes:
      - './localstack-data:/var/lib/localstack/data'
      - './.localstack:/var/lib/localstack'
      - '/var/run/docker.sock:/var/run/docker.sock'
  redis:
    image: redis:7.2-alpine
    container_name: redis_bullmq
    ports:
      - '6379:6379'
    volumes:
      - './redis-data:/data'

```

Рис. 3.24 Файл docker-compose.yaml

Розширений файл конфігурації охоплює складніші сервіси архітектури, включаючи Twenty CRM та worker-процеси. Цей файл демонструє використання health checks для моніторингу стану сервісів, налаштування залежностей між контейнерами та комплексну конфігурацію змінних середовища для інтеграції з зовнішніми API-провайдерами. Дослідження показують, що такий підхід до containerized deployment мікросервісів забезпечує ефективне використання ресурсів та швидкі запуски.

3.10.2 Розгортання голосового агента на платформі Daily.co з використанням Pipecat

Деплой компонента Voice Agent здійснюється через інтеграцію з фреймворком Pipecat, що надає спеціалізовані можливості для розгортання голосових функцій на платформі Daily.co. Цей процес передбачає контейнеризацію Voice Agent Service разом з усіма його залежностями, включаючи Silero VAD та Pipeline-компоненти, у формат, сумісний з інфраструктурою Daily.co.

Використання Pipecat дозволяє абстрагуватися від складнощів інтеграції з платформою та зосередитися на бізнес-логіці голосового асистента. Фреймворк забезпечує standardized interface для взаємодії з аудіо-потоками та управління сесіями користувачів, що критично важливо для забезпечення низької латентності та високої якості голосової взаємодії. Дослідження мікросервісних архітектур підтверджують, що контейнеризація значно покращує можливості горизонтального масштабування для real-time застосунків.

3.11 Огляд shared-модуля для CRM Post-processing модулів

Shared-модуль виконує функцію архітектурного фундаменту, що уніфікує ключові абстракції для модулів CRM та Post-processing. Його основним завданням є інкапсуляція спільних паттернів та контрактів, що забезпечує дотримання принципів DRY через усунення дублювання базової логіки. Цей підхід дозволяє синхронізувати архітектурні рішення між різними доменами системи, зберігаючи при цьому чіткі межі відповідальності кожного компонента. Детально класи показано на Рис. 3.25.

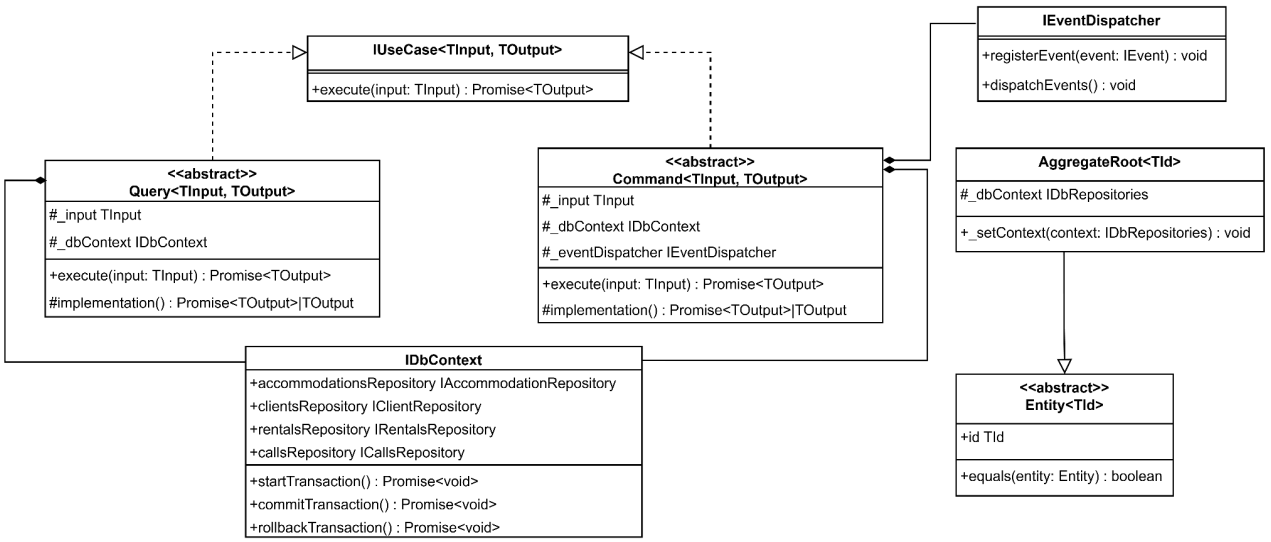


Рис. 3.25 Діаграма класів shared-модуля

Опис класів shared-модулю з рисунка 3.25:

– інтерфейс IUseCase<TInput, TOutput> – визначає контракт для інкапсуляції атомарних операцій бізнес-логіки у форматі варіантів використання. Параметри TInput (вхідні дані) та TOutput (результат) забезпечують типобезпечну реалізацію use case-ів, де кожен екземпляр відповідає за окрему бізнес-операцію. Ця абстракція сприяє слабкій зв'язності між компонентами, дозволяючи легко замінювати або тестувати окремі сценарії взаємодії;

– абстрактний клас Query<TInput, TOutput> – надає базову структуру для операцій читання даних у рамках CQRS. Успадковуючись від IUseCase, він формалізує принцип "запит не змінює стан системи", інкапсулюючи логіку отримання інформації зі сховищ. Специфікація TInput дозволяє параметризувати умови вибірки, тоді як TOutput гарантує типізоване представлення результатів;

– абстрактний клас Command<TInput, TOutput> – реалізує шаблон команд для мутаційних операцій, що змінюють стан системи. Як похідний від IUseCase, він забезпечує єдиний інтерфейс для операцій оновлення даних, включаючи валідацію вхідних параметрів (TInput) та повернення

структурованих результатів (TOutput). Це дозволяє централізовано керувати транзакціями та обробкою помилок;

- інтерфейс IDbContext – абстрагує доступ до сховища даних, інкапсулюючи специфіку ORM. Реалізуючи принцип інверсії залежностей, він дозволяє доменній моделі залишатися незалежною від інфраструктурних деталей. Похідні класи можуть імплементувати конкретні механізми для Entity Framework, MongoDB чи інших технологій;
- інтерфейс IEventDispatcher – забезпечує механізм публікації доменних подій, що критично важливо для реалізації подієво-орієнтованої архітектури. Відокремлюючи генерацію подій від їх обробки, він дозволяє створювати слабко зв'язані системи, де нові обробники можна додавати без змін у вихідному коді use case-ів;
- абстрактний клас Entity<TId> – служить базою для всіх доменних сутностей, визначаючи унікальний ідентифікатор (TId) та методи порівняння. Інкапсулює логіку життєвого циклу об'єкта, включно з відстеженням змін стану та забезпечення цілісності даних через інваріанти;
- клас AggregateRoot<TId> – успадковуючи Entity<TId>, визначає корені агрегатів – кластери сутностей, які змінюються як єдине ціле. Відповідає за координацію змін у межах агрегату, генерацію подій змін та забезпечення консистентності через бізнес-правила. Це дозволяє уникати розмиття меж відповідальності між пов'язаними об'єктами.

Shared-модуль формує концептуальний каркас, що забезпечує структурну цілісність архітектури. Через уніфікацію базових абстракцій, він усуває розбіжності в реалізації бізнес-логіки між CRM Manager та Post-processing.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ ПОДАЛЬШОГО РОЗВИТКУ

4.1 Аналіз результатів

Запропонована архітектура успішно продемонструвала свою спроможність поєднати мікросервісні практики із підходом Retrieval-Augmented Generation я Чітке застосування принципів Domain-Driven Design та модульного моноліту знизило складність розробки та дозволило ізолювати відповідальності між ядром CRM-сервісу та допоміжним модулем пост-обробки. У результаті, вдалося забезпечити час відгуку голосового асистента, що залишався нижчим за поставлену межу у 1000 мілісекунд, а рівень помилок розпізнавання мовлення (WER) тримався в межах допустимих 10% у контрольованих акустичних умовах. Інтеграція з Twenty CRM та зовнішніми AI-сервісами підтвердила доцільність поєднання LLM-моделей із корпоративними даними для обробки запитів клієнтів. Двоетапний механізм автентифікації через Clerk і Twenty CRM виявився достатньо гнучким для реалізації гранульованого доступу, а застосування BullMQ для асинхронної передачі подій забезпечило стійкість під час пікових навантажень – а, також, надає можливості для подальшого горизонтального масштабування сервісів.

4.2 Перспективи подальшого розвитку

Наступним кроком у еволюції системи стане впровадження підтримки multitenancy, що дозволить обслуговувати кілька організацій на єдиній інфраструктурі без необхідності створювати окремі інстанси. Це вимагатиме реалізації механізмів ізоляції даних, наприклад, через окремі схеми в базі даних або віртуальні робочі простори, з одночасним збереженням продуктивності та безпеки кожного клієнта.

Паралельно планується розширити мовні можливості агента, забезпечивши підтримку одночасного використання української та англійської мов у ході однієї сесії. Для цього необхідне створення гібридної мультимовної моделі, доповненої розширеною бібліотекою українських голосів, а також оптимізація конвеєрів STT/TTS для динамічного перемикання мовного контексту на рівні окремої репліки.

Удосконалення процесів управління користувачами передбачає розробку централізованої системи авторизації, інтегрованої з усіма компонентами екосистеми. Використання єдиного Identity Provider із підтримкою OAuth 2.1 та OpenID Connect разом із ролями на основі атрибутів значно спростить реєстрацію та управління обліковими записами клієнтів і співробітників.

Підвищення консистентності даних досяжне завдяки впровадженню транзакційної системи збереження записів, реалізованої на основі патерну Event Sourcing. Фіксація атомарних подій із наступною синхронізацією з CRM-реплікацією дасть змогу відтворювати стан системи в будь-який момент і аналізувати історію змін без втрати цілісності.

Розширення інтеграційної здатності системи передбачає створення адаптера для підключення до сторонніх CRM-рішень. Універсальний API-шар з підтримкою плагінної архітектури забезпечить трансформацію специфічних моделей даних різних провайдерів у єдиний доменний формат.

Економічну ефективність платформи слід досліджувати через детальний аналіз тарифних моделей хмарних провайдерів і порівняння продуктивності різних LLM, STT та TTS-рішень. Застосування кешування запитів і оптимізація промптів допоможуть знизити операційні витрати без зниження якості обслуговування.

Врешті, створення автоматизованого evaluation pipeline відкриє можливість безперервного контролю якості. Інтеграція метрик розпізнавання та генерації з фідбеком від користувачів та проведення A/B-тестів нових версій моделей сприятимуть своєчасному виявленню дрейфу даних і дозволять підтримувати високий рівень надійності голосового асистента в умовах постійної еволюції бізнес-вимог.

ВИСНОВКИ

В рамках даного дипломного проекту, було досліджено характерні риси сфери гостинності, виділено сегмент бізнесу за родом діяльності, а також виокремлено основні процеси та канали спілкування, характерні для цього сегменту. У результаті дослідження було виділено подобову оренду житла як цільовий сегмент; ідентифіковано ключові бізнес-процеси; визначено основні канали комунікації; а також встановлено специфічні характеристики вербального спілкування для голосових інтерфейсів.

Проаналізовано існуючі аналоги програмних засобів, призначених для обслуговування вербальних запитів у сфері гостинності. Аналіз 7 рішень показав, що переважна більшість існуючих платформ орієнтована на великі корпоративні структури або ресторани, що робить їх недоступними для малого бізнесу подобової оренди. Виявлено відсутність підтримки української мови у всіх проаналізованих рішеннях.

Проведено детальний аналіз функціональних та нефункціональних вимог до розроблюваного програмного забезпечення – при цьому було враховано особливості людського вербального спілкування.

Здійснено обґрунтований вибір програмних засобів, сервісів та платформ, необхідних для реалізації програмного продукту. Обрано платформу Daily.co з фреймворком Pipecat Flows для розгортання голосового асистента, технологічний стек з використанням NestJS, Next.js. провайдерів генеративних моделей та моделей розпізнавання мовлення Deepgram, ElevenLabs, OpenAI. Використано підхід RAG для забезпечення актуальності інформації.

Спроєктовано архітектуру програмного забезпечення, приділяючи особливу увагу мінімізації ризиків на етапі розробки та забезпеченню масштабованості системи в майбутньому. Було використаномікросервісну архітектуру з застосуванням принципів Domain-Driven Design, що забезпечило чітке розподілення відповідальності між підсистемами.

Розроблено та протестовано програмне забезпечення, яке відповідає раніше визначеним функціональним та нефункціональним вимогам. Створено систему голосового асистента з підтримкою української та англійської мов, реалізовано інтеграцію з Twenty CRM, розроблено BFF та мікрофронтенд з авторизацією через Clerk для зручного перегляду дзвінків. Впроваджено комплексну стратегію тестування – включно з 24 інтеграційними тестами для репозиторіїв CRM, а, також, юніт-тести для бізнес-логіки. Забезпечено контейнеризацію через Docker для спрощення розгортання системи.

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

Тези доповідей:

1. Артеменко А.О., Худік Б.О. Нефункціональні вимоги для модуля голосового асистента системи обробки клієнтських запитів у сфері гостинності. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 370–372.

2. Артеменко А.О. Безпека голосового AI-агента: автентифікація та захист мікросервісної архітектури. XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ ім. Б. Грінченка, 2025. С. 258–259. ISSN 2664-2638.

3. Артеменко А.О. Забезпечення безпечної передачі аудіоданих між модулем голосового асистента та TTS, STT моделями. XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ ім. Б. Грінченка, 2025. С. 257–258. ISSN 2664-2638.

4. Артеменко А.О., Худік Б.О. Огляд Python фреймворку Pipecat Flow для побудови голосових асистентів. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку).

Стаття у збірнику:

1. Артеменко А.О., Худік Б.О. Організація безпечного функціонування голосових агентів на базі великих мовних моделей. Кібербезпека: освіта, наука, техніка. Київський столичний університет імені Бориса Грінченка, 2025. (подано до друку)

ПЕРЕЛІК ПОСИЛАНЬ

1. Бондаренко Л. П., Дишлюк О. О. Аналіз факторів формування ринку оренди житлової нерухомості в Україні. Наукові розробки, передові технології, інновації: збірник наукових праць та тез наукових доповідей за матеріалами VI Міжнародної науково-практичної конференції, Прага, 6–8 жовт. 2020 р.
2. Про забезпечення функціонування української мови як державної : Закон України від 25.04.2019 р. № 2704-VIII. URL: https://zakon.rada.gov.ua/laws/show/2704-19 (дата звернення: 06.06.2025).
3. Світлична В. Ю., Александрова С. А. Ефективні комунікації в готельно-ресторанному господарстві : навчальний посібник. Харків : ХНУМГ ім. О. М. Бекетова, 2025.
4. Стаценко О. А., Дорогий Я. Ю. Аналіз технологій та перспектив розвитку голосових помічників на основі штучного інтелекту. Інтелектуальний ресурс сьогодення: наукові задачі, розвиток та запитання : збірник наукових праць з матеріалами IV міжнародної наукової конференції, Вінниця, Україна, 14 березня 2025 р. Вінниця: ТОВ «УКРЛОГОС Груп», 2025. С. 252.
5. A Functional Software Reference Architecture for LLM-Integrated Systems / A. Bucaioni, M. L. Bernardi, C. Ghezzi, P. Pelliccione. 2025. URL: https://www.themoonlight.io/en/review/a-functional-software-reference-architecture-for-llm-integrated-systems (дата звернення: 06.06.2025).
6. A Multi-Agent Approach for REST API Testing with Semantic Graphs and LLM-Driven Input / M. Kim, J. Lee, S. Park, H. Kim. 2024.
7. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges / X. Li, Y. Wang, Z. Liu, T. Zhang, J. Li. Vicinagearth. 2024.
8. AgentMonitor: A Plug-and-Play Framework for Predictive and Secure Multi-Agent Systems / C.-M. Chan, Y. Li, M. Zhang, H. Wang, X. Wang. 2024.

URL: https://www.themoonlight.io/en/review/agentmonitor-a-plug-and-play-framework-for-predictive-and-secure-multi-agent-systems (дата звернення: 06.06.2025).

9. Asynchronous Tool Usage for Real-Time Agents / A. A. Ginart, N. Kodali, J. Lee, C. Xiong, S. Savarese, J. Emmons. Salesforce AI Research. 2024. arXiv:2410.21620.
10. AutoSafeCoder: A Multi-Agent Framework for Securing LLM Code Generation through Static Analysis and Fuzz Testing / A. Nunez, J. Chen, Y. Zhang, S. Liu. 2024.
11. Calabrèse A., Cheong A. M. Baseline MNREAD Measures for Normally Sighted Subjects From Childhood to Old Age. Investigative Ophthalmology & Visual Science. 2016.
12. Coevolving with the Other You: Fine-Tuning LLM with Sequential Cooperative Multi-Agent Reinforcement Learning / H. Ma, Y. Wang, H. Li, X. Zhang. 2025.
13. Competition Reduces Response Times in Multiparty Conversation / J. Holler та ін. Frontiers in Psychology. 2021.
14. CREFT: Sequential Multi-Agent LLM for Character Relation Extraction / Y. Eun Chun, T. Hwang, S.-w. Hwang, B.-H. Kim. arXiv:2505.24553 [cs.CL]. 2025. URL: https://arxiv.org/abs/2505.24553 (дата звернення: 06.06.2025).
15. Cross-domain Activity Recognition via Substructural Optimal Transport / W. Lu та ін. 2021.
16. Domain-Driven Design Representation of Monolith Candidate Decompositions Based on Entity Accesses / M. Levezinho та ін. 2024.
17. Elijah A. How to Choose STT & TTS for AI Voice Agents in 2025: A Comprehensive Guide. Softcery. URL: https://softcery.com/lab/how-to-choose-stt-tts-for-ai-voice-agents-in-2025-a-comprehensive-guide/ (дата звернення: 13.05.2025).

18. Enhancing Function-Calling Capabilities in LLMs: Strategies for Prompt Formats, Data Integration, and Multilingual Translation / C. Yi-Chang, H. Lee, S. Kim. 2024.
19. Kalhor B., Das S. Evaluating the Security and Privacy Risk Postures of Virtual Assistants. 2023.
20. Lando B., Hasselbring W. Toward Bundler-Independent Module Federations: Enabling Typed Micro-Frontend Architectures. 2025.
21. Livekit. Deepgram STT integration guide. LiveKit Docs. URL: https://docs.livekit.io/agents/integrations/stt/deepgram/ (дата звернення: 12.05.2025).
22. Open AI. OpenAI developer platform. Open AI Platform. URL: https://platform.openai.com/docs/overview/ (дата звернення: 12.05.2025).
23. Park C., Chen M., Hain T. Automatic Speech Recognition System-Independent Word Error Rate Estimation. Speech and Hearing Research Group, University of Sheffield. 2024.
24. Requirements Quality Research: a harmonized Theory, Evaluation, and Roadmap / J. Frattini та ін. Springer Nature, 2023.
25. RichRAG: Crafting Rich Responses for Multi-faceted Queries in Retrieval-Augmented Generation / S. Wang та ін. 2024.
26. Specification 2025-03-26 (Latest). Modelcontextprotocol.io. URL: https://modelcontextprotocol.io/specification/2025-03-26 (дата звернення: 25.05.2025).
27. Stephen S. C., Torreira F. Timing in turn-taking and its implications for processing models of language. Frontiers in Psychology. 2015.
28. Suo X. Signed-Prompt: A New Approach to Prevent Prompt Injection Attacks Against LLM-Integrated Applications. 2025.
29. Szeider S. MCP-Solver: Integrating Language Models with Constraint Programming Systems. Algorithms and Complexity Group. 2025.
30. Towards a Middleware for Large Language Models / N. Guran, F. Knauf, M. Ngo, S. Petrescu, J. S. Rellermeyer. 2024. URL:

https://www.linkedin.com/posts/stephen-gpt_title-towards-a-middleware-for-large-language-activity-7268910270553509888-ZFK4 (дата звернення: 06.06.2025).

31. Towards Efficient LLM Grounding for Embodied Multi-Agent Collaboration / Y. Zhang, X. Li, J. Liu, H. Wang. 2024.
32. W3schools. Python Introduction. W3schools. Python Tutorial. URL: https://www.w3schools.com/python/python_intro.asp (дата звернення: 12.05.2025).
33. W3schools. TypeScript Introduction. W3schools. TypeScript tutorial. URL: https://www.w3schools.com/typescript/typescript_intro.php (дата звернення: 12.05.2025).
34. Wen W., Zhenyue Z., Tianshu S. Customizing Large Language Models for Business Context: Framework and Experiments.
35. Y. Chang E., Geng L. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка голосового агента для автоматизації бізнес-процесів з обслуговування клієнтських запитів на базі RAG з використанням мікросервісної архітектури

Виконав студент 4 курсу
групи ПД-44
Артеменко Андрій Олександрович
Керівник роботи
доктор філософії (PhD), доцент кафедри ІПЗ Худік Богдан Олександрович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – автоматизувати обслуговування вербальних клієнтських запитів та оцінити потенціал заміщення людини шляхом розробки програмного забезпечення.

Об'єкт дослідження – процеси надання послуг з оренди житла на подобовій основі та супроводу споживача впродовж періоду проживання в орендованій оселі, використовуючи засоби голосового зв'язку.

Предмет дослідження – програмне забезпечення, націлене на обробку запитів клієнтів в контексті подобової оренди житла в реальному часі.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. дослідити характерні риси сфери гостинності, виділити сегмент бізнесу за родом діяльності, виокремити основні процеси та канали спілкування характерні для цього сегменту;
2. проаналізувати існуючі аналоги програмних засобів для обслуговування вербальних запитів в сфері гостинності;
3. провести аналіз функціональних та нефункціональних вимог до програмного забезпечення, беручи до уваги особливості людського вербального спілкування;
4. обрати програмні засоби, сервіси та платформи для реалізації програмного продукту;
5. спроєктувати архітектуру програмного забезпечення з фокусом на стабільність роботи, зменшення ризиків під час розробки та масштабованість системи;
6. розробити та протестувати отримане в кінцевому підсумку програмне забезпечення відповідно до поставлених вимог

3

АНАЛІЗ АНАЛОГІВ

Особливість	SynXisVoice Agent	Aiello AVA	Verbio Solutions	SoundHound AI	HiJiffy	RestoHost.AI	Голосовий асистент для обробки запитів
Можливість інтеграції з телефонними дзвінками	Так	Так	Так	Ні	Ні	Ні	Так
Інтеграція з CRM	Так	Ні	Ні	Ні	Ні	Ні	Так
Розуміння природної мови	Так	Так	Так	Ні	Ні	Ні	Так
Підтримка української мови	Ні	Ні	Ні	Ні	Ні	Ні	Так
Цільова група клієнтів	Великі готелі	Мережі готелів	Великі готелі	Ресторани	Середні готельні бізнеси	Ресторани	Малі та середні бізнеси подобової оренди житла
Дистанційне бронювання голосом	Так	Так	Так	Ні	Ні	Ні	Так
Сумаризація дзвінків	Ні	Ні	Ні	Ні	Ні	Ні	Так
Можливість обробляти запити пов'язані з екстреними випадками	Ні	Так	Ні	Ні	Ні	Ні	Так
Обробка інформаційних запитів	Так	Так	Так	Ні	Ні	Ні	Так

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні:

1. Обробка голосових запитів на оформлення бронювання нерухомості
2. Обробка голосових запитів на скасування раніше створеного бронювання
3. Агент повинен надавати покрокові інструкції під час дистанційного заселення користувача на місце проживання
4. Надання оперативної інформації та інструкцій у разі надзвичайних ситуацій
5. Агент має оновлювати записи у внутрішній CRM системі під час проведення дзвінка
6. Адміністратор повинен мати можливість редагувати список нерухомості
7. Адміністратор повинен мати можливість завантажувати текстові інструкції для обробки інформаційних та екстрених запитів до бази знань нерухомості

Нефункціональні:

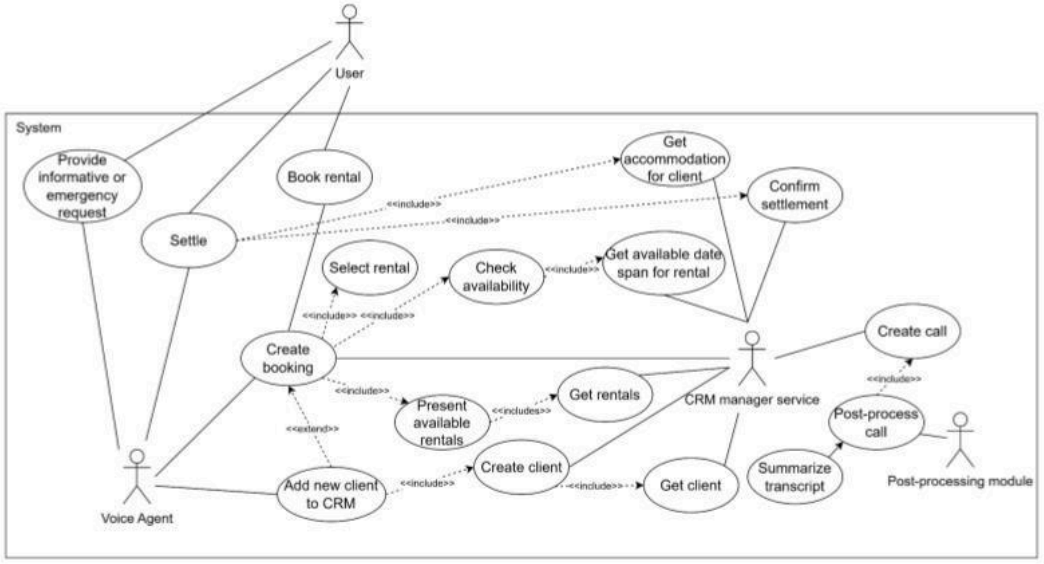
1. Час відповіді від завершення репліки користувачем до початку отримання відповіді від голосового асистента не має не перевищувати 1000 мс
2. Рівень помилок розпізнавання мовлення (WER) не вище 10 % в середовищах без сторонніх голосів та рівнем шуму до 60 дБ
3. Голосовий асистент повинен підтримувати українську та англійську мови на ввід та вивід мовлення
4. Користувацький інтерфейс CRM та системи моніторингу дзвінків повинен підтримувати англійську мову

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



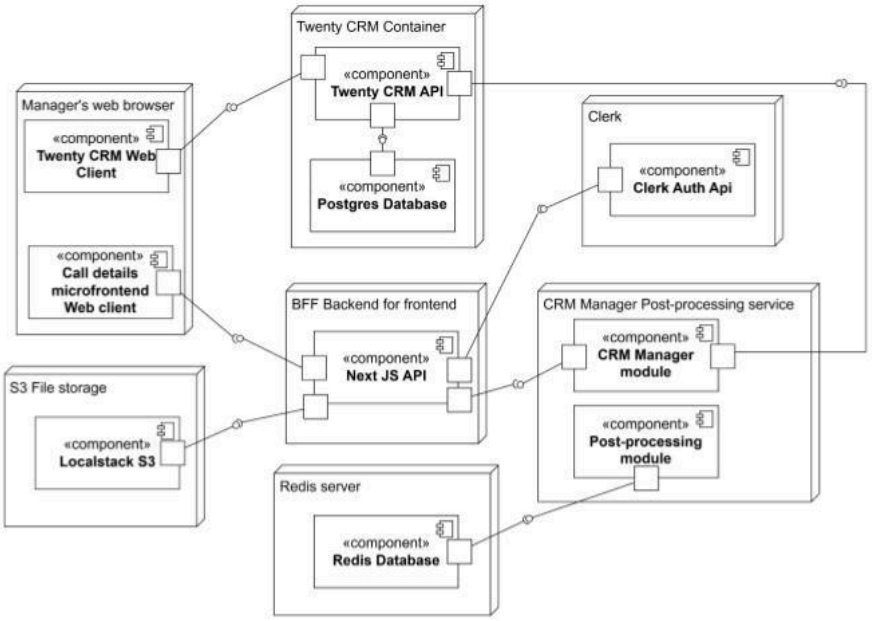
7

ТИПИ ДОМЕНІВ

Generic	Core	Supporting
CRM	CRM Manager	Call post-processing
Authorization	Voice agent	File storage
LLM, TTS, STT		

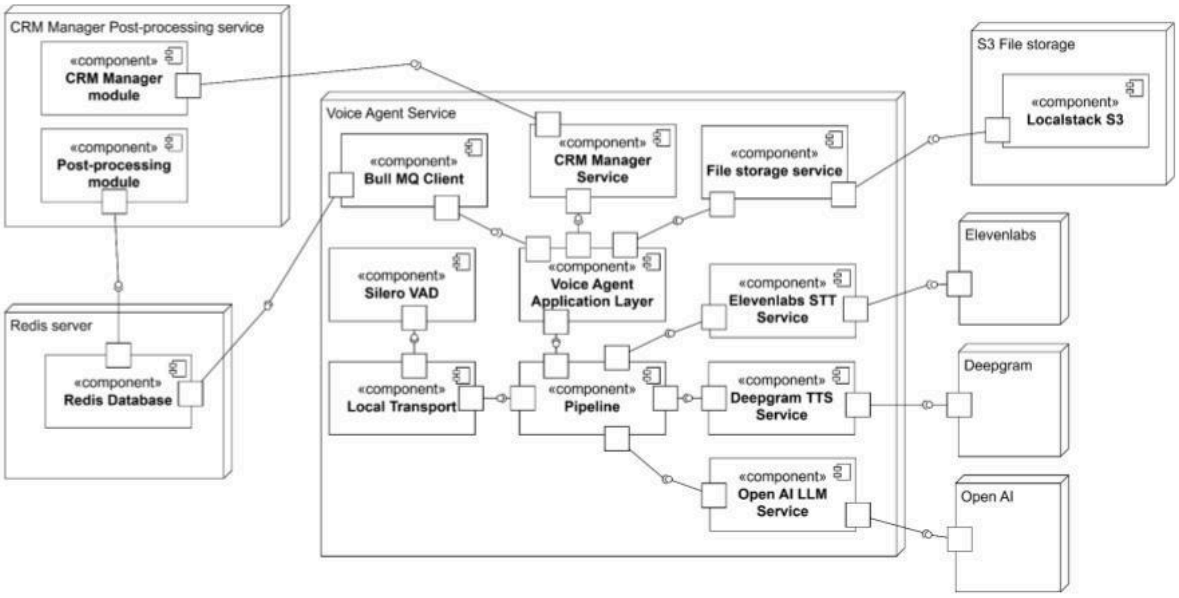
8

АРХІТЕКТУРА СИСТЕМИ



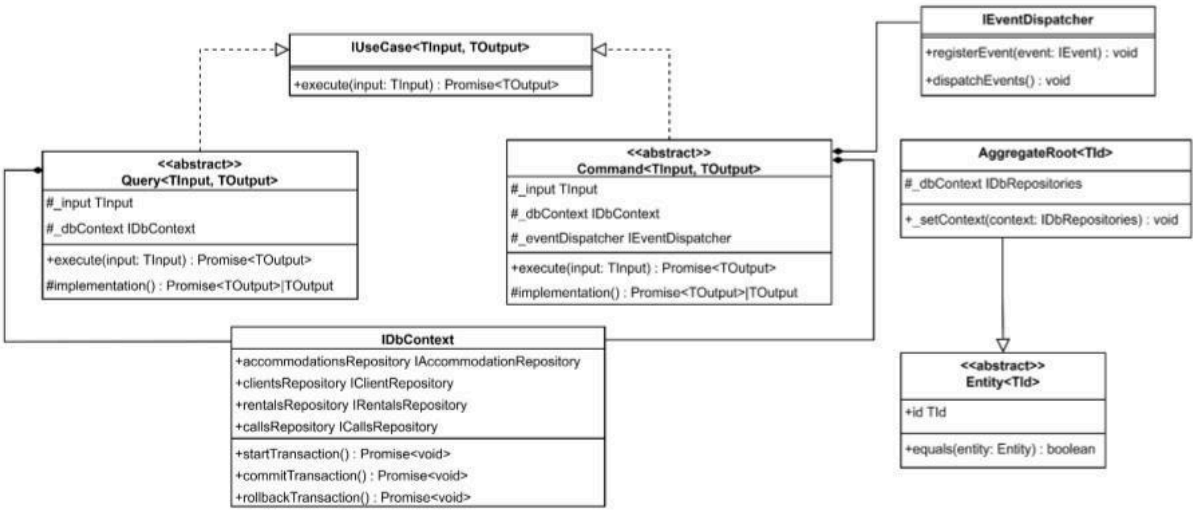
9

АРХІТЕКТУРА СИСТЕМИ



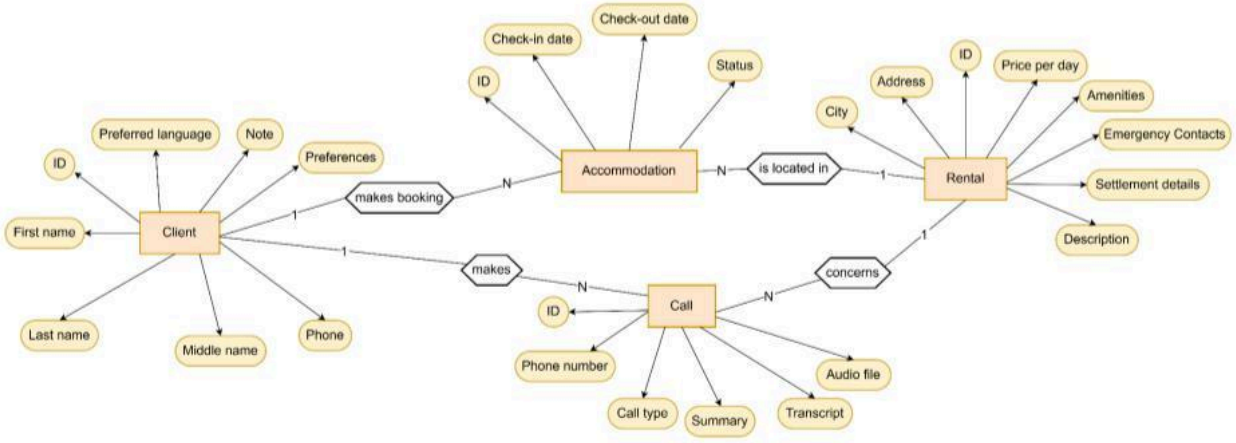
10

ДІАГРАМА КЛАСІВ SHARED МОДУЛЮ СЕРВІСІВ CRM-MANAGER ТА POST-PROCESSING



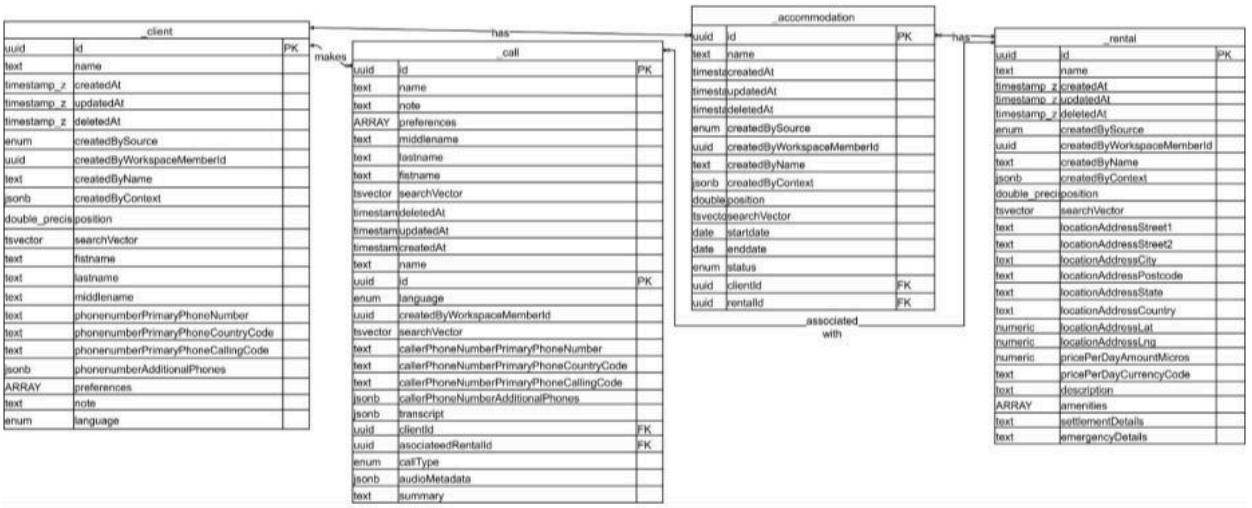
11

ІНФОЛОГІЧНА МОДЕЛЬ БАЗИ ДАНИХ CRM



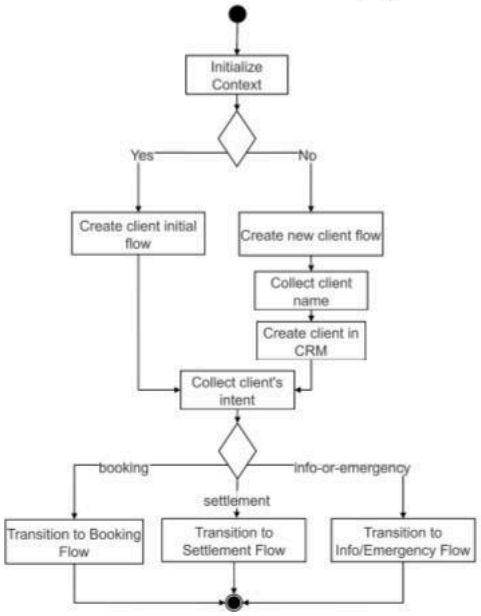
12

ДАТАЛОГІЧНА МОДЕЛЬ БАЗИ ДАНИХ CRM

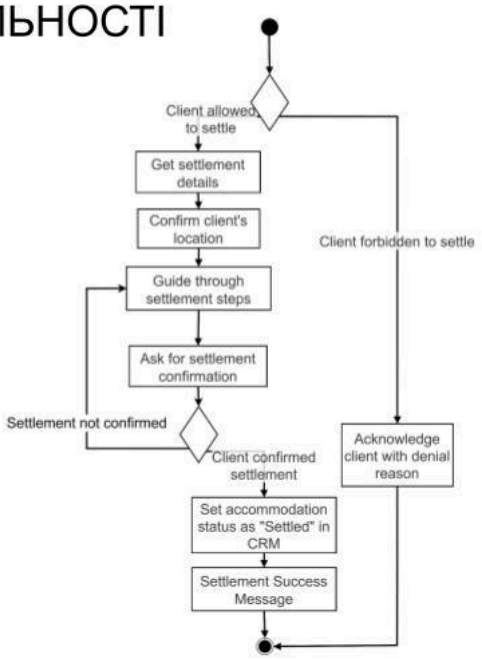


13

ДІАГРАМА ДІЯЛЬНОСТІ



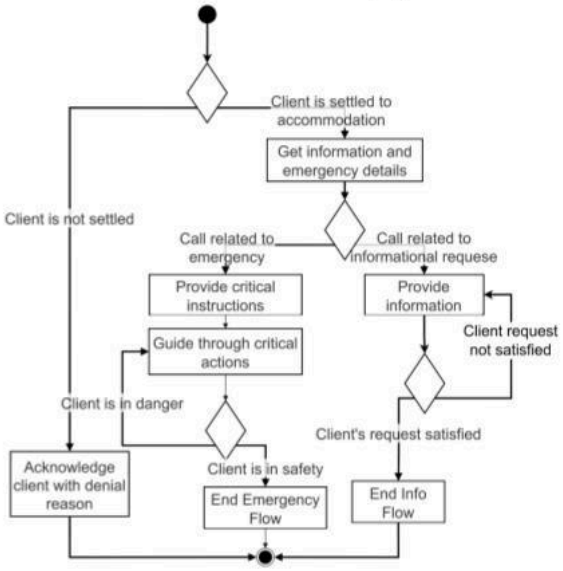
Початок



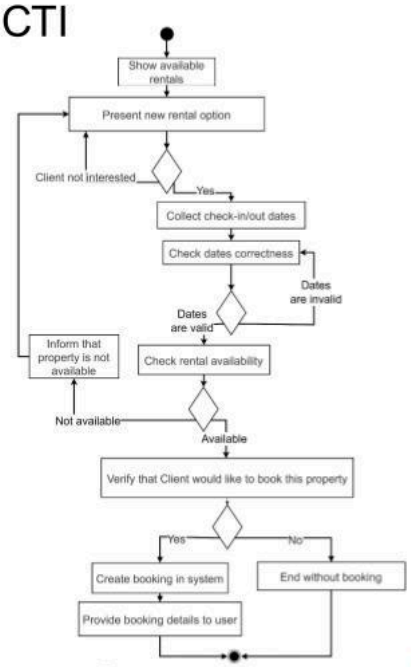
Заселення

14

ДІАГРАМА ДІЯЛЬНОСТІ



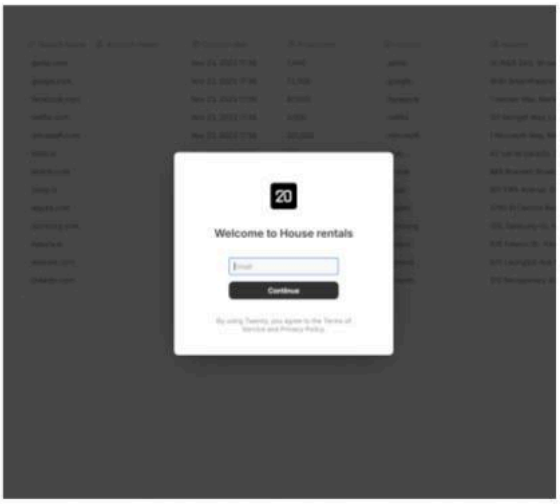
Обробка надання інформації
або допомога під час надзвичайної ситуації



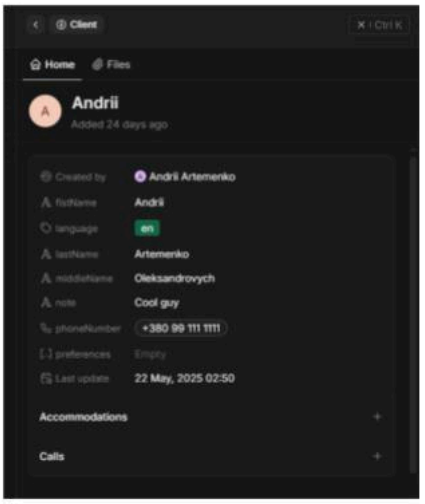
Бронювання житла

15

ЕКРАННІ ФОРМИ



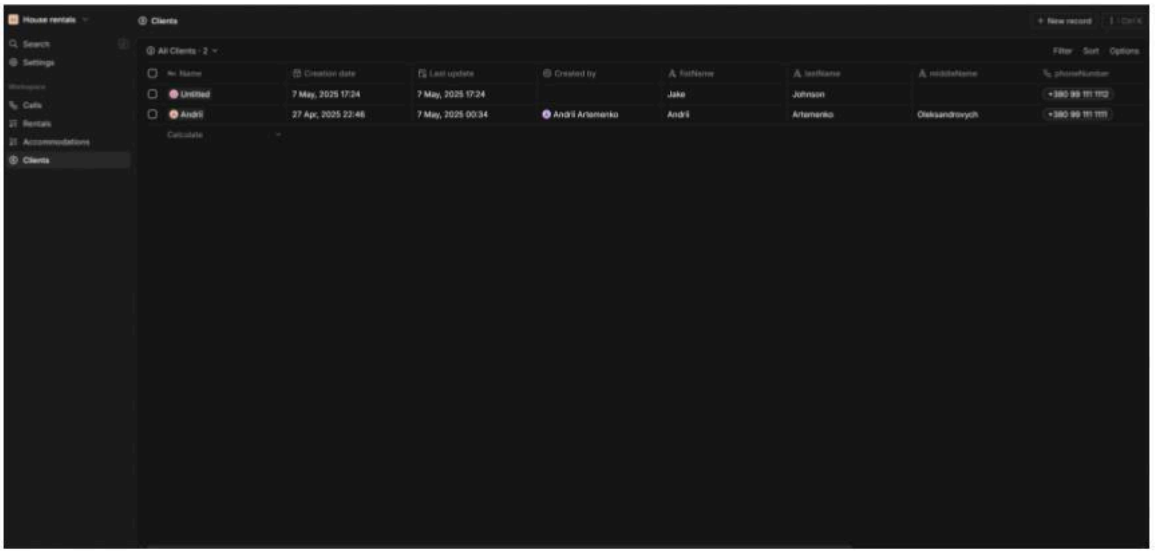
Екран входу в систему
CRM



Меню редагування даних клієнта

16

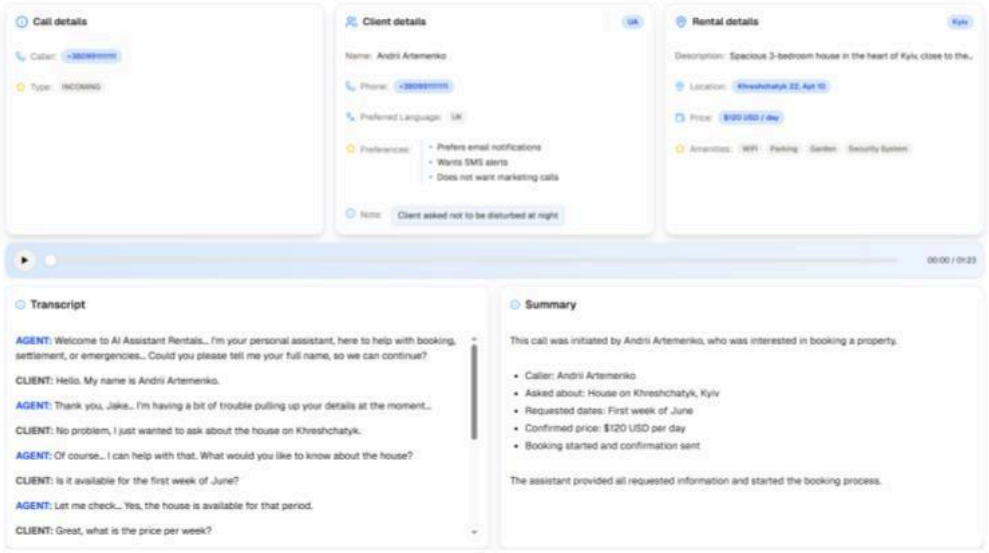
ЕКРАННІ ФОРМИ



Табличний вигляд даних клієнтів

17

ЕКРАННІ ФОРМИ



Інтерфейс для перегляду дзвінка

18

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей

1. Артеменко А.О., Худік Б.О. Нефункціональні вимоги для модуля голосового асистента системи обробки клієнтських запитів у сфері гостинності. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 370–372.

2. Артеменко А.О. Безпека голосового AI-агента: автентифікація та захист мікросервісної архітектури. XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ ім. Б. Грінченка, 2025. С. 258–259. ISSN 2664-2638.

3. Артеменко А.О. Забезпечення безпечної передачі аудіоданих між модулем голосового асистента та TTS, STT моделями. XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ ім. Б. Грінченка, 2025. С. 257–258. ISSN 2664-2638.

4. Артеменко А.О., Худік Б.О. Огляд Python фреймворку Pipecat Flow для побудови голосових асистентів. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку).

Стаття у збірнику

5. Артеменко А.О., Худік Б.О. Організація безпечного функціонування голосових агентів на базі великих мовних моделей. Кібербезпека: освіта, наука, техніка. Київський столичний університет імені Бориса Грінченка, 2025. (подано до друку).

ВИСНОВКИ

1. В рамках даного дипломного проекту, було досліджено характерні риси сфери гостинності, виділено сегмент бізнесу за родом діяльності, а також виокремлено основні процеси та канали спілкування, характерні для цього сегменту. У результаті дослідження було виділено подобову оренду житла як цільовий сегмент; ідентифіковано ключові бізнес-процеси; визначено основні канали комунікації; а також встановлено специфічні характеристики вербального спілкування для голосових інтерфейсів.

2. Проаналізовано існуючі аналоги програмних засобів, призначених для обслуговування вербальних запитів у сфері гостинності. Аналіз 7 рішень показав, що переважна більшість існуючих платформ орієнтована на великі корпоративні структури або ресторани, що робить їх недоступними для малого бізнесу подобової оренди. Виявлено відсутність підтримки української мови у всіх проаналізованих рішеннях.

3. Проведено детальний аналіз функціональних та нефункціональних вимог до розроблюваного програмного забезпечення – при цьому було враховано особливості людського вербального спілкування.

4. Здійснено обгрунтований вибір програмних засобів, сервісів та платформ, необхідних для реалізації програмного продукту. Обрано платформу Daily.co з фреймворком Pipecat Flows для розгортання голосового асистента, технологічний стек з використанням NestJS, Next.js. провайдерів генеративних моделей та моделей розпізнавання мовлення Deepgram, ElevenLabs, OpenAI. Використано підхід RAG для забезпечення актуальності інформації.

5. Спроєктовано архітектуру програмного забезпечення, приділяючи особливу увагу мінімізації ризиків на етапі розробки та забезпеченню масштабованості системи в майбутньому. Було використаномікросервісну архітектуру з застосуванням принципів Domain-Driven Design, що забезпечило чітке розподілення відповідальності між підсистемами.

6. Розроблено та протестовано програмне забезпечення, яке відповідає раніше визначеним функціональним та нефункціональним вимогам. Створено систему голосового асистента з підтримкою української та англійської мов, реалізовано інтеграцію з Twenty CRM, розроблено BFF та мікрофронтенд з авторизацією через Clerk для зручного перегляду дзвінків. Впроваджено комплексну стратегію тестування – включно з 24 інтеграційними тестами для репозиторіїв CRM, а, також, юніт-тестидля бізнес-логіки. Забезпечено контейнеризацію через Docker для спрощення розгортання системи.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Модуль Voice Agent

```
import asyncio
from datetime import datetime
import io
import sys
import os
import json
from typing import Tuple
from uuid import uuid4
import wave
import boto3

import aiofiles
from dotenv import load_dotenv
from loguru import logger
from
crm_api_client.crm_manager_client.models.boo
k_rental_dto import BookRentalDto
from
crm_api_client.crm_manager_client.api.clients
import
clients_controller_get_current_accommodation
from
crm_api_client.crm_manager_client.models.crea
te_client_dto import CreateClientDto
from
crm_api_client.crm_manager_client.models.clie
nt_dto import ClientDto
from domain.models.conversation_context
import ConversationContext
from
crm_api_client.crm_manager_client.api.clients
import clients_controller_find_client_by_phone,
clients_controller_create_client
from
crm_api_client.crm_manager_client.api.rentals
import rentals_controller_get_rentals,
rentals_controller_get_rental_by_id,
rentals_controller_get_rental_available_date_sp
ans,
rentals_controller_get_rental_emergency_details
,
rentals_controller_get_rental_settlement_details
from
crm_api_client.crm_manager_client.api.accomm
odations import
accommodations_controller_confirm_settlement
, accommodations_controller_create_booking
from select_audio_device import AudioDevice,
run_device_selector
from pipecat.services.elevenlabs.tts import
ElevenLabsTTSService
from pipecat.frames.frames import Frame,
TranscriptionFrame
from pipecat.pipeline.pipeline import Pipeline
from pipecat.pipeline.runner import
PipelineRunner
from pipecat.pipeline.task import PipelineTask
from pipecat.processors.frame_processor import
FrameDirection, FrameProcessor
from
pipecat.processors.audio.audio_buffer_processor
import AudioBufferProcessor
from pipecat.transports.local.audio import
LocalAudioTransport,
LocalAudioTransportParams
from pipecat.services.deepgram.stt import
DeepgramSTTService
from pipecat.services.deepgram.stt import
LiveOptions
from pipecat.services.cartesia.tts import
CartesiaTTSService
from pipecat.audio.vad.silero import
SileroVADAnalyzer
from pipecat.audio.vad.vad_analyzer import
VADParams
from pipecat.services.openai.llm import
OpenAILLMService
from pipecat.services.openai.llm import
OpenAILLMContext
from pipecat.pipeline.task import
PipelineParams
from pipecat_flows import FlowManager,
FlowsFunctionSchema, FlowArgs, NodeConfig
from
crm_api_client.crm_manager_client.models.date
_day_dto import DateDayDto
from domain.events.call_completed_event
import CallCompletedEvent, Replica
from bullmq import Queue

load_dotenv(override=True)
S3_BUCKET = os.getenv("S3_BUCKET")
S3_ENDPOINT_URL =
os.getenv("S3_ENDPOINT_URL")
AWS_REGION = os.getenv("AWS_REGION",
"us-east-1")
DEFAULT_LANGUAGE =
os.getenv("DEFAULT_LANGUAGE", "en")

REDIS_HOST = os.getenv("REDIS_HOST",
"localhost")
REDIS_PORT = int(os.getenv("REDIS_PORT",
6379))
REDIS_PASSWORD =
os.getenv("REDIS_PASSWORD", None)
REDIS_USERNAME =
os.getenv("REDIS_USERNAME", None)
REDIS_PROTOCOL =
```

```
os.getenv("REDIS_PROTOCOL", "redis")
if REDIS_USERNAME and
REDIS_PASSWORD:
    REDIS_URL =
f"{REDIS_PROTOCOL}://{REDIS_USERNA
ME}:{REDIS_PASSWORD}@{REDIS_HOST
}:{REDIS_PORT}"
elif REDIS_PASSWORD:
    REDIS_URL =
f"{REDIS_PROTOCOL}://:{REDIS_PASSWO
RD}@{REDIS_HOST}:{REDIS_PORT}"
else:
    REDIS_URL =
f"{REDIS_PROTOCOL}://{REDIS_HOST}:{R
EDIS_PORT}"

s3_client = boto3.client(
    "s3",

aws_access_key_id=os.getenv("AWS_ACCESS
_KEY_ID"),

aws_secret_access_key=os.getenv("AWS_SECR
ET_ACCESS_KEY"),
    endpoint_url=S3_ENDPOINT_URL,
    region_name=AWS_REGION,
    config=boto3.session.Config(
        signature_version='s3v4',
        s3={'addressing_style': 'path'}
    )
)
async def save_audio(audio: bytes, sample_rate:
int, num_channels: int, name: str):
    if len(audio) > 0:
        timestamp =
datetime.now().strftime('%Y%m%d_%H%M%S
')
        object_key =
f"{name}_conversation_recording{timestamp}.
wav"
        with io.BytesIO() as buffer:
            with wave.open(buffer, "wb") as wf:
                wf.setsampwidth(2)
                wf.setnchannels(num_channels)
                wf.setframerate(sample_rate)
                wf.writeframes(audio)
            buffer.seek(0)
            loop = asyncio.get_running_loop()

            try:
                await loop.run_in_executor(None,
lambda:
s3_client.head_bucket(Bucket=S3_BUCKET))
            except Exception:
                try:
                    create_bucket_kwargs = {'Bucket':
S3_BUCKET}
                    if AWS_REGION != 'us-east-1':

create_bucket_kwargs['CreateBucketConfigurati
on'] = {
                        'LocationConstraint':
AWS_REGION
                    }
                await loop.run_in_executor(None,
lambda:
s3_client.create_bucket(**create_bucket_kwarg
s))
                print(f"Created bucket:
{S3_BUCKET}")
            except Exception as e:
                print(f"Error creating bucket:
{str(e)}")
                return None

            try:
                await loop.run_in_executor(None,
s3_client.upload_fileobj, buffer, S3_BUCKET,
object_key)
                print(f"Merged audio saved to
s3://{S3_BUCKET}/{object_key}")
                return object_key
            except Exception as e:
                print(f"Error uploading to S3:
{str(e)}")
                return None
    else:
        print("No audio data to save")
        return None

voice_instructions = """
You are a helpful assistant in a call center. You
are talking to a client.
Organization is called "AI Rentals".

**Core Principle: Sound-First
Communication**
*   **Conciseness is Key:** Keep your
responses to 1 or 2 sentences maximum. Avoid
long, complex sentences.
*   **Avoid Lists (Unless Asked):** Do not
present long lists or enumerations of options
unless the user explicitly asks for them. If you
have multiple points, deliver them one at a time,
or ask if the user wants to hear more.
*   **Punctuation for Pauses:** Use punctuation
sparingly. Primarily, use commas or ellipses (...)
to indicate natural pauses in speech, rather than
for strict grammatical correctness. Avoid
excessive exclamation marks or question marks
unless truly necessary for tone.
*   **Aural Primacy:** Construct all
communications exclusively for auditory
delivery. Envision the entire exchange as a
spoken dialogue, where information is conveyed
and received solely through the medium of
sound.
*   **Purely Verbal Rendition:** Since the
interaction is entirely verbal, all output must be
translatable into pure sound. Eliminate any
reliance on visual cues, textual embellishments
(like asterisks or bullet points not meant to be
spoken as "asterisk" or "bullet point"), or
symbolic representations that are not naturally
vocalized in conversation.
*   **Phonetic Accessibility:** Prioritize
```

```
vocabulary and sentence structures that ensure
effortless articulation and clear phonetic
reception. Select words that are commonly
understood and phonetically straightforward to
minimize auditory processing load and
maximize comprehension.
*   **Dynamic Conversational Flow:** Emulate
the natural cadence and rhythm of human
speech. Integrate authentic prosodic variations
(changes in pitch, tone, stress), natural
hesitations (e.g., "um," "er" if contextually
appropriate and not overused), and common
conversational connectors (e.g., "so," "well,"
"alright," "now," "actually") to cultivate an
organic, engaging, and relatable interaction
style. The aim is to create a sense of unscripted,
fluid dialogue.
*   **Modulated Pacing and Deliberate
Pauses:** Vary the speed of delivery and
strategically incorporate pauses (guided by your
punctuation use). This not only mirrors natural
speech patterns but also aids listener
comprehension by allowing moments for
information absorption.
*   **Structured Auditory Information
(Briefly):** When presenting detailed or
multi-part information (if requested by the user),
structure it for easy listening in short segments.
Employ techniques such as framing the
information and using brief verbal signposting
("Firstly...", "Next...").
*   **Implicit Communicative Intent:** Subtly
convey the underlying purpose or tone (e.g.,
informative, inquisitive, supportive) through
natural vocal nuances rather than explicit
declarations of emotion, unless the context
specifically calls for it. The expression should
feel inherent to the way the words are spoken.
*   **Fostering Listener Engagement:** Where
appropriate, use intonation and phrasing that
naturally invites continued interaction or signals
attentiveness to the conversational partner. This
can include slight upward inflections for
questions or statements that invite a response,
without being overtly demanding.
"""

role_instructions = """
You are a manager at "AI Rentals," a
voice-based rental agency.
Your primary goal is to assist the customer
efficiently and courteously with their inquiries
about booking, settlement, or emergencies
related to our properties.
You must only use information explicitly
provided to you through the system (like rental
details, booking information, or specific
instructions from functions).
Do not invent information, make assumptions
beyond what's given, or use any external or
global knowledge. If you don't have the
information, politely state that.
Maintain a professional, helpful, and friendly
tone at all times.
"""

async def
initial_collect_full_name_handler(args:
FlowArgs, flow_manager: FlowManager):
    """Handler for collecting user's full name."""
    first_name = args.get("first_name")
    last_name = args.get("last_name")
    middle_name = args.get("middle_name", "")
    logger.info(f"Collected full name:
{first_name} {last_name} {middle_name}")

    created_client = await
clients_controller_create_client.asyncio(
        client=crm_client,
        body=CreateClientDto(
            first_name=first_name,
            last_name=last_name,
            middle_name=middle_name,

phone_number=flow_manager.state['context'].ge
t_phone_number()
        )
    )

flow_manager.state['context'].set_client(created_
client)
    return {"status": "success", "first_name":
first_name, "last_name": last_name,
"middle_name": middle_name}

initial_collect_full_name_schema =
FlowsFunctionSchema(
    name="initial_collect_full_name",
    description="Record user's full name with
middle name. Call this function only once you
have collected all parameters. Do not call it
twice.",
    properties={"first_name": {"type": "string"},
"last_name": {"type": "string"}, "middle_name":
{"type": "string"}},
    required=["first_name", "last_name"],
    handler=initial_collect_full_name_handler,
)

from crm_api_client.crm_manager_client.client
import Client

crm_client =
Client(base_url=os.getenv("CRM_MANAGER_
URL"))

async def
search_client_by_phone_number(phone_number
: str):
    result = await
clients_controller_find_client_by_phone.asyncio
(
        client=crm_client,
        phone_number=phone_number
    )
```

```
    return result


def create_unknown_client_initial_flow(context:
ConversationContext):
    language_code = context.get_language()
    language_instruction = {
        "role": "system",
        "content": f"You MUST respond ONLY in
the {language_code} language. Do not use any
other language in your responses."
    }
    flow_config = {
            "role_messages": [
                {
                    "role": "system",
                    "content": role_instructions
                },
                {
                    "role": "system",
                    "content": voice_instructions
                },
                language_instruction
            ],
            "task_messages": [
                {
                    "role": "system",
                    "content": """Start by greeting the
user with message. Use introduction message:
                    "Welcome to AI Assistant Rentals. I
am your personal assistant. I can help you with
booking, settlement and emergencies
                    Could you please provide me with
your full name so we can continue?"
                    """
                }
            ],
            "functions":
[initial_collect_full_name_schema]
    }
    return flow_config


async def
get_additional_rental_details(rental_id: str):
    rental_details = await
rentals_controller_get_rental_by_id.asyncio(
        client=crm_client,
        id=rental_id
    )
    return rental_details


async def get_rental_availability_handler(args:
FlowArgs, flow_manager: FlowManager):
    """Handler for getting rental availability."""
    rental_id = args.get("rental_id")
    start_date = args.get("start_date")
    end_date = args.get("end_date")

    # Validate date formats and values
    from datetime import datetime
    current_date = datetime.now().date()

    # Date format validation function
    def is_valid_date_format(date_str):
        try:
            datetime.strptime(date_str,
"%d-%m-%Y")
            return True
        except ValueError:
            return False

    # Perform validations
    if not is_valid_date_format(start_date):
        logger.error(f"Invalid start_date format:
{start_date}")
        return {"status": "error", "message":
f"Invalid start date format. Please use
DD-MM-YYYY format."}

    if not is_valid_date_format(end_date):
        logger.error(f"Invalid end_date format:
{end_date}")
        return {"status": "error", "message":
f"Invalid end date format. Please use
DD-MM-YYYY format."}

    # Convert strings to date objects for
comparison
    start_date_obj = datetime.strptime(start_date,
"%d-%m-%Y").date()
    end_date_obj = datetime.strptime(end_date,
"%d-%m-%Y").date()

    # Check if dates are in the past
    if start_date_obj < current_date:
        logger.error(f"Start date {start_date} is in
the past")
        return {"status": "error", "message": f"Start
date cannot be in the past. Today is
{current_date.strftime('%d-%m-%Y')}."}

    # Check if end date is before start date
    if end_date_obj < start_date_obj:
        logger.error(f"End date {end_date} is
before start date {start_date}")
        return {"status": "error", "message": "End
date cannot be before start date."}

    logger.info(f"Getting availability for rental
ID: {rental_id}, start_date: {start_date},
end_date: {end_date}")
    try:
        availability_spans = await
rentals_controller_get_rental_available_date_sp
ans.asyncio(
            client=crm_client,
            id=rental_id,
            start_date=start_date,
            end_date=end_date
        )
        # The API might return a complex object.
We need to format it or decide what to return.
        # For now, returning the raw response. This
might need adjustment based on actual API
```

```
response structure.
        logger.info(f"Availability spans:
{availability_spans}")
        return {"status": "success", "rental_id":
rental_id, "availability": availability_spans}
    except Exception as e:
        logger.error(f"Error getting rental
availability: {e}")
        return {"status": "error", "message": str(e)}

get_rental_availability_schema =
FlowsFunctionSchema(
    name="get_rental_availability",
    description="Get available date spans for a
specific rental property. Call this function after
the user has selected a rental and wants to know
its availability. The dates must be in
DD-MM-YYYY format.",
    properties={
        "rental_id": {"type": "string", "description":
"The ID of the rental property to check
availability for."},
        "start_date": {"type": "string",
"description": "The start date in
DD-MM-YYYY format."},
        "end_date": {"type": "string", "description":
"The end date in DD-MM-YYYY format."}
    },
    required=["rental_id", "start_date",
"end_date"],
    handler=get_rental_availability_handler,
)

async def create_booking_handler(args:
FlowArgs, flow_manager: FlowManager):
    rental_id = args.get("rental_id")
    start_date_str = args.get("start_date")
    end_date_str = args.get("end_date")

    def parse_ddmmyyyy(date_str):
        day, month, year = map(int,
date_str.split("-"))
        return DateDayDto(year=year,
month=month, day=day)

    start_date = parse_ddmmyyyy(start_date_str)
    end_date = parse_ddmmyyyy(end_date_str)

    response = await
accommodations_controller_create_booking.asy
ncio_detailed(
        client=crm_client,
        body=BookRentalDto(
            rental_id=rental_id,
            start_date=start_date,
            end_date=end_date,

client_id=flow_manager.state['context'].get_clie
nt().id
        )
    )
    print(response)
    if response.status_code < 200 or
response.status_code >= 300:
        message = response.content.decode("utf-8")
if isinstance(response.content, bytes) else
str(response.content)
        try:
            import json
            error_json = json.loads(message)
            message = error_json.get("message",
message)
        except Exception:
            pass
        return {"status": "error", "message":
message}
    return {"status": "success", "rental_id":
rental_id, "start_date": start_date_str,
"end_date": end_date_str}

create_booking_schema =
FlowsFunctionSchema(
    name="create_booking",
    description="Create a booking for the client.",
    properties={"rental_id": {"type": "string",
"description": "The ID of the rental to book."},
"start_date": {"type": "string", "description":
"The start date in DD-MM-YYYY format."},
"end_date": {"type": "string", "description":
"The end date in DD-MM-YYYY format."}},
    required=["rental_id", "start_date",
"end_date"],
    handler=create_booking_handler,
)

async def create_booking_flow(context:
ConversationContext):
    rentals_list = await
rentals_controller_get_rentals.asyncio(
        client=crm_client,
    )
    from datetime import datetime
    current_date =
datetime.now().strftime("%d-%m-%Y")
    language_code = context.get_language()
    language_instruction = {
        "role": "system",
        "content": f"You MUST respond ONLY in
the {language_code} language. Do not use any
other language in your responses."
    }

    flow_config = {
        "role_messages": [
            {
                "role": "system",
                "content": role_instructions
            },
            {
                "role": "system",
                "content": voice_instructions
            },
            language_instruction
        ],
        "task_messages": [
            {
```

```
            "role": "system",
            "content": f"""Your goal is to help the client book accommodation.
            You are able to retrieve information about available rentals and book them.
            Today's date is {current_date}. Speak dates naturally, for example, instead of "10-05-2025", say "the tenth of May".
            Follow these steps:
            1. Present available rentals to the client one by one from the <rentals> section. Provide only a brief description (e.g., name of the rental or type of apartment and general location). Wait for the user to ask for more details if they are interested.
                Example:
                - User: "I'm looking for a place to stay."
                - Assistant: "Okay, I can help with that. We have a 'Comfortable apartment on the Black Avenue'. Does that sound interesting?"
                - User: "Tell me more about it."
                - Assistant: [Provides more details]
                - User: "No, could you suggest another one?"
                - Assistant: "Sure, how about the 'Luxury suite in the center of the city'?"
            2. Once the client expresses interest in a specific rental and you've provided more details if requested, ask them for their desired check-in and check-out dates. Do not ask for a specific format; simply ask "When would you like to check in?" and "And when would you like to check out?".
            3. Use the `get_rental_availability` function to fetch available date spans. Provide the `rental_id` of the selected rental along with the `start_date` and `end_date` parameters in DD-MM-YYYY format. You will need to convert the user's date input to this format.
                IMPORTANT DATE VALIDATIONS (for your internal processing before calling the function):
                - Ensure dates are converted to DD-MM-YYYY format.
                - Check that the start_date is not in the past (today is {current_date}, speak it naturally).
                - Verify that the end_date is after the start_date.
                - If validation fails, the function will return an error message - inform the client naturally and ask for new dates.
            4. Inform the client about the availability, speaking dates naturally.
            5. If the client confirms a date and wants to book, use the `create_booking` function. Provide the `rental_id`, `start_date` (check-in), and `end_date` (check-out) for the booking. Ensure all dates are in DD-MM-YYYY format internally.
            6. After successfully calling `create_booking`, confirm the booking details (rental, check-in date, check-out date, spoken naturally) with the client and inform them that their booking is complete.
            7. Once the booking is confirmed and the client has no more questions, or if the client wishes to end the conversation at any point, call the `booking_end_quote` function to terminate the call.
            """
        },
        {
            "role": "system",
            "content": f"""
            <rentals>
            {rentals_list}
            </rentals>
            When calling `get_rental_availability` or `create_booking`, ensure you use the correct `rental_id` from the list above for the rental the user is interested in.
            Internally, always use DD-MM-YYYY format for dates when calling functions. When speaking to the user, express dates naturally (e.g., "the tenth of May").
            Remember these validation rules for `get_rental_availability` and before calling `create_booking`:
            1. Dates must be in DD-MM-YYYY format for function calls.
            2. Start date must not be before today ({current_date}, spoken naturally).
            3. End date must be after start date.
            If the user provides dates that don't meet these criteria, explain the issue naturally and ask for new dates before proceeding.
            """
        }
    ],
    "functions": [get_rental_availability_schema, create_booking_schema, booking_end_quote_schema]
  }
  return flow_config


async def create_settlement_success_end_node(context: ConversationContext) -> NodeConfig:
  """Creates a node to confirm successful settlement and end the call."""
  language_code = context.get_language()
  language_instruction = {
    "role": "system",
    "content": f"You MUST respond ONLY in the {language_code} language. Do not use any other language in your responses."
  }
  return {
    "role_messages": [
```

```
            {
                "role": "system",
                "content": role_instructions
            },
            {
                "role": "system",
                "content": voice_instructions
            },
            language_instruction
        ],
        "task_messages": [
            {
                "role": "system",
                "content": f"Great news,
{context.get_client().first_name}! Your
settlement has been successfully confirmed. We
hope you have a wonderful stay at AI Rentals. If
you need anything else, feel free to reach out.
Goodbye!"
            }
        ],
        "functions": [],
        "post_actions": [{"type":
"end_conversation"}],
    }

async def get_settlement_details_handler(args:
FlowArgs, flow_manager: FlowManager):
    settlement_details = await
rentals_controller_get_rental_settlement_details.
asyncio(
        client=crm_client,

id=flow_manager.state['context'].get_client_acco
mmodation().rental_id
    )
    print(settlement_details)
    return {"status": "success",
"settlement_details":
settlement_details.settlement_details}

get_settlement_details_schema =
FlowsFunctionSchema(
    name="get_settlement_details",
    description="Get settlement details for the
accommodation.",
    properties={},
    required=[],
    handler=get_settlement_details_handler,
)
        return flow_config
```