ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інтелектуальна хмарна платформа медичного страхування на основі взаємодії спеціалізованих AI-агентів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

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

	Дмитро ДЕНИСОВ
(підпис)	*Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:	здобувач вищої освіти гр. ІСД-41 Дмитро ДЕНИСОВ *Ім'я, ПРІЗВИЩЕ*
Керівник: *науковий ступінь, вчене звання*	Каміла СТОРЧАК, д. т. н., професор *Ім'я, ПРІЗВИЩЕ*
Рецензент: *науковий ступінь, вчене звання*	 *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____Каміла СТОРЧАК
«____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Денисов Дмитро Вячеславович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інтелектуальна хмарна платформа медичного страхування на основі взаємодії спеціалізованих AI-агентів,

керівник кваліфікаційної роботи Каміла СТОРЧАК, д. т. н., професор,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» 02 2026р. № 51

2. Строк подання кваліфікаційної роботи «01» 06 2026р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань мультиагентних систем, великих мовних моделей, технології RAG, хмарних платформ та автоматизації клієнтського обслуговування; документація Amazon Bedrock Agents, AWS Lambda, Amazon DynamoDB, Nuxt 4; матеріали щодо медичного страхування домашніх тварин і ветеринарного консультування.

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

1. Дослідити теоретичні основи побудови мультиагентних хмарних платформ для автоматизації страхових послуг.
2. Спроєктувати архітектуру платформи «ВетЕксперт» та алгоритми взаємодії спеціалізованих ШІ-агентів.
3. Реалізувати програмний прототип платформи, провести тестування та оцінити ефективність її впровадження.

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «20» 02 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	12.03.2026	
2	Дослідження теоретичних основ мультиагентних систем, LLM та RAG	21.03.2026	
3	Аналіз існуючих підходів до автоматизації страхових і ветеринарних консультацій	04.04.2026	
4	Проєктування архітектури платформи «ВетЕксперт» та алгоритмів взаємодії агентів	10.04.2026	
5	Програмна реалізація, тестування та оцінка ефективності платформи	09.05.2026	
6	Розробка демонстраційних матеріалів	15.05.2026	
7	Попередній захист квалфікаційної роботи	19.05.2026	
8	Подання роботи в деканат	01.06.2026	

Здобувач вищої освіти _____ Дмитро ДЕНИСОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 33 рис., 25 табл., 36 джерел.

Мета роботи: розробка та програмна реалізація інтелектуальної хмарної платформи медичного страхування домашніх тварин «ВетЕксперт» на основі взаємодії спеціалізованих ШІ-агентів, здатної в межах одного діалогу автоматизувати страхове та ветеринарне консультування.

Об'єкт дослідження: процес автоматизованого обслуговування клієнтів у сфері медичного страхування домашніх тварин.

Предмет дослідження: методи та засоби побудови мультиагентної хмарної платформи на основі великих мовних моделей (LLM), технології RAG і безсерверної інфраструктури.

Короткий зміст роботи: у першому розділі проаналізовано роль інтелектуальних систем у сучасній автоматизації послуг, порівняно монолітну, мікросервісну та мультиагентну архітектури, розглянуто методи обробки природної мови та протоколи взаємодії агентів, сформульовано постановку задачі та технічні вимоги до платформи «ВетЕксперт». У другому розділі формалізовано бізнес-процеси страхового та ветеринарного консалтингу в нотаціях AS-IS і TO-BE, спроєктовано архітектуру мультиагентної системи з оркестратором і чотирма спеціалізованими агентами, описано алгоритми класифікації намірів, маршрутизації запитів, чотирирівневої оцінки терміновості ветеринарних звернень та RAG-конвеєра з міжагентною передачею контексту за патерном blackboard, обґрунтовано вибір технологічного стеку (Amazon Bedrock Agents, AWS Lambda, Amazon DynamoDB, Nuxt 4). У третьому розділі наведено програмну реалізацію серверної логіки та клієнтського інтерфейсу, результати модульного, інтеграційного, сценарного та навантажувального тестування, а також техніко-економічне обґрунтування впровадження. Експериментально підтверджено, що платформа скорочує типовий цикл обслуговування з 15-25 хвилин до 1-2 хвилин, забезпечує

цілодобову доступність, точність класифікації намірів 95% і коректну категоризацію невідкладних ветеринарних станів у 90% сценаріїв; розрахункова вартість обробки одного звернення становить 0,04 USD проти 12,50 USD при роботі оператора. Наукова новизна полягає в узгодженому поєднанні контекстно-залежної маршрутизації між страховим і ветеринарним агентами, формалізованого алгоритму оцінки терміновості та механізму спільного сховища контексту сесії в безсерверній мультиагентній архітектурі. Практична значущість: отримано програмний прототип платформи «ВетЕксперт», придатний до випробування в реальних умовах страхово-ветеринарного обслуговування.

КЛЮЧОВІ СЛОВА: мультиагентна система, великі мовні моделі, Retrieval-Augmented Generation, безсерверні обчислення, медичне страхування домашніх тварин, автоматизація обслуговування клієнтів, хмарна платформа.

ABSTRACT

Text part of the bachelor's qualification work: 65 pages, 33 figures, 25 tables, 36 sources.

The purpose of the work: to develop and implement an intelligent cloud platform for pet health insurance, VetExpert, based on interaction of specialized AI agents capable of automating insurance and veterinary consulting within a single dialogue.

Object of research: the process of automated customer service in pet health insurance.

Subject of research: methods and tools for building a multi-agent cloud platform based on large language models (LLM), RAG technology, and serverless infrastructure.

Summary of the work: the first chapter analyzes the role of intelligent systems in modern service automation, compares monolithic, microservice, and multi-agent architectures, reviews natural language processing methods and agent interaction protocols, and formulates the problem statement and technical requirements for the VetExpert platform. The second chapter formalizes insurance and veterinary consulting business processes in AS-IS and TO-BE notations, designs a multi-agent system architecture with an orchestrator and four specialized agents, describes algorithms for intent classification, request routing, four-level veterinary urgency assessment, and a RAG pipeline with inter-agent context transfer using the blackboard pattern, and justifies the technology stack (Amazon Bedrock Agents, AWS Lambda, Amazon DynamoDB, Nuxt 4). The third chapter presents server-side logic and client interface implementation, results of unit, integration, scenario, and load testing, and a technical and economic justification of deployment. Experiments confirm that the platform reduces the typical service cycle from 15-25 minutes to 1-2 minutes, provides 24/7 availability, achieves 95% intent classification accuracy, and correctly categorizes emergency veterinary cases in 90% of scenarios; the estimated cost per request is USD 0.04 versus USD 12.50 for a human operator. Scientific

novelty lies in the integrated combination of context-dependent routing between insurance and veterinary agents, a formalized urgency assessment algorithm, and a session context store in a serverless multi-agent architecture. Practical significance: a software prototype of the VetExpert platform suitable for pilot deployment in insurance and veterinary customer service.

KEYWORDS: multi-agent system, large language models, Retrieval-Augmented Generation, serverless computing, pet health insurance, customer service automation, cloud platform.

ЗМІСТ

ВСТУП 10

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ МУЛЬТИАГЕНТНИХ СИСТЕМ ДЛЯ АВТОМАТИЗАЦІЇ СТРАХОВИХ ПОСЛУГ 13

1.1. Роль інтелектуальних систем у сучасних процесах автоматизації послуг 13

1.2. Аналіз існуючих підходів до автоматизації та актуальність мікросервісних та агентних архітектур 18

1.3. Огляд методів обробки природної мови (NLP) та протоколів взаємодії агентів для предметної області 22

1.4. Постановка задачі дослідження та формування технічних вимог до системи «ВетЕксперт» 25

РОЗДІЛ 2. СИСТЕМНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПЛАТФОРМИ 30

2.1. Формалізація бізнес-процесів страхування та ветеринарного консалтингу 30

2.2. Проєктування архітектури мультиагентної системи 41

2.3. Формалізація алгоритмів прийняття рішень та маршрутизації запитів 53

2.4. Обґрунтування вибору технологічного стека (LLM-моделі, фреймворки оркестрації, хмарні бази даних) 63

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПЛАТФОРМИ «ВЕТЕКСПЕРТ» 71

3.1. Особливості програмної реалізації серверної логіки та механізму взаємодії агентів 71

3.2. Розробка клієнтського інтерфейсу та інтеграція системи із зовнішніми API 77

3.3. Тестування платформи: перевірка сценаріїв діалогів, навантажувальне тестування та валідація відповідей 86

3.4. Техніко-економічне обґрунтування проєкту та оцінка ефективності впровадження 90

ВИСНОВКИ 99

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 102

ДОДАТОК А. ЕКРАННІ ФОРМИ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ ПЛАТФОРМИ «ВЕТЕКСПЕРТ» 108

А.1. Головний екран діалогу з мультиагентною системою 108

А.2. Адміністративна панель улюбленців та полісів 109

А.3. Панель ветеринарних записів 111

ДОДАТОК Б. ПОСИЛАННЯ НА РЕПОЗИТОРІЙ ВИХІДНОГО КОДУ 113

ПРЕЗЕНТАЦІЯ 114

ВСТУП

Останнє десятиліття помітно змінило ландшафт сервісної автоматизації. Великі мовні моделі (LLM), технологія генерації з доповненням пошуком (RAG), мультиагентні архітектури та безсерверні хмарні обчислення зробили можливим перехід від жорстко сценарних чат-ботів до діалогових систем, які розуміють контекст і одночасно працюють із кількома предметними областями. За оцінками провідних аналітичних компаній, сьогодні вже понад дві третини організацій використовують принаймні одне ШІ-рішення у своїх процесах, причому найшвидше зростає частка таких рішень саме в автоматизації клієнтського обслуговування. Деталізовані кількісні показники та динаміка впровадження розглянуті в підрозділі 1.1.

На тлі цього загальносвітового тренду ринок медичного страхування домашніх тварин залишається слабко автоматизованим. Типове звернення клієнта зачіпає одразу дві предметні області: страхову (умови полісу, розрахунок премії, оформлення страхового випадку) і ветеринарну (оцінка симптомів, перша допомога, визначення терміновості). Наявні ж платформи спираються переважно на кол-центри або шаблонні чат-боти, які не забезпечують консультування в межах обох доменів одночасно. Завдання об’єднати ці компетенції в єдиному діалоговому інтерфейсі без втрати якості відповіді розв’язується лише на стику кількох технологій, що й визначає напрям цієї кваліфікаційної роботи.

Теоретичну базу такого поєднання утворюють дослідження з інтелектуальних і мультиагентних систем, трансформерних моделей та RAG-генерації із заземленням відповідей на зовнішні джерела знань. Проте у цих роботах практично не розглянуто питання інженерного поєднання

перелічених компонентів у мультидоменну LLM-платформу з єдиним оркестрованим діалогом і механізмом міжагентної передачі контексту. Саме ця прогалина зумовлює актуальність обраної теми.

Метою роботи є розробка та програмна реалізація інтелектуальної хмарної платформи медичного страхування домашніх тварин «ВетЕксперт» на основі взаємодії спеціалізованих ШІ-агентів. Об'єктом дослідження виступає процес автоматизованого обслуговування клієнтів у сфері медичного страхування домашніх тварин, предметом дослідження є методи та засоби побудови мультиагентної хмарної платформи на основі LLM, RAG і безсерверної інфраструктури. Повний перелік із семи задач дослідження та методологічна основа наведені у підрозділі 1.4.

У межах дослідження розроблено цілісну мультиагентну архітектуру платформи «ВетЕксперт», в якій узгоджено поєднано схему контекстно-залежної класифікації клієнтських намірів і маршрутизації запитів між страховим та ветеринарним агентами, алгоритм чотирирівневої оцінки терміновості ветеринарних звернень на основі аналізу симптомів і профілю тварини та RAG-конвеєр з міжагентною передачею контексту за патерном blackboard. Відповідні архітектурні та алгоритмічні рішення подано в підрозділі 1.4; формалізацію цих рішень розкрито у підрозділі 2.3.

Наукова новизна роботи полягає в узгодженому поєднанні трьох елементів, які в джерелах зазвичай описують окремо: система сама визначає, коли звернення обробляє страховий агент, а коли - ветеринарний, залежно від наміру клієнта й того, що вже було в діалозі; додано алгоритм оцінки терміновості у чотири рівні з урахуванням симптомів і профілю тварини; дві RAG-бази знань працюють узгоджено й передають між агентами вже зібраний контекст за патерном blackboard. Усе це подано у формалізованому вигляді, реалізовано як безсерверну платформу й підтверджено експериментально за якістю відповіді, швидкістю реакції та вартістю звернення порівняно з оператором-людиною. Робота не зводиться

до демонстрації великої мовної моделі: отримано цілісну модель спільного страхового й ветеринарного діалогу, яку можна відтворити в іншому проєкті - від формулювання вимог до обґрунтування економічної доцільності. Практична значущість полягає в тому, що платформу доведено до рівня, на якому її можна випробовувати в реальних умовах. «ВетЕксперт» працює цілодобово й відповідає на страхові та ветеринарні питання в межах одного діалогу, скорочує типовий цикл обслуговування з 15-25 хвилин до 1-2 хвилин і побудований на сервісах AWS Lambda, Amazon Bedrock і Amazon DynamoDB, тож не потрібно тримати власні сервери. За оцінками McKinsey, автоматична обробка звернення коштує приблизно в 5-8 разів менше, ніж робота оператора; власні розрахунки для платформи також показують суттєве здешевлення порівняно з оператором.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розкрито роль інтелектуальних систем в автоматизації послуг, проаналізовано монолітну, мікросервісну та агентну архітектури, оглянуто методи обробки природної мови і протоколи взаємодії агентів та сформульовано постановку задачі з технічними вимогами до «ВетЕксперт». У другому розділі формалізовано бізнес-процеси страхового та ветеринарного консалтингу в нотаціях AS-IS / TO-BE, спроєктовано архітектуру мультиагентної системи, описано алгоритми маршрутизації запитів і оцінки терміновості та обґрунтовано технологічний стек. У третьому розділі викладено особливості програмної реалізації серверної логіки і клієнтського інтерфейсу, наведено результати функціонального та навантажувального тестування, а також техніко-економічне обґрунтування ефективності впровадження платформи.

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ МУЛЬТИАГЕНТНИХ СИСТЕМ ДЛЯ АВТОМАТИЗАЦІЇ СТРАХОВИХ ПОСЛУГ

1.1. Роль інтелектуальних систем у сучасних процесах автоматизації послуг

За звітом McKinsey [1], у 2024 році близько 72% опитаних організацій уже впровадили хоча б одну функцію на основі штучного інтелекту, тоді як у 2017 році таких було лише 20%. Найбільш помітне зростання припадає саме на автоматизацію клієнтського обслуговування, яку називають провідним напрямом понад 40% респондентів; регулярне використання генеративних моделей при цьому підтверджують близько двох третин компаній. Загальну динаміку цього процесу протягом 2017-2024 рр. відображено на рис. 1.1.

Рис. 1.1 Динаміка впровадження штучного інтелекту в організаціях у 2017-2024 рр. [1]

Світовий ринок генеративних і діалогових ШІ-рішень для обслуговування клієнтів перебуває у фазі прискореного зростання, що зумовлено не лише технологічним прогресом, а й прямим економічним ефектом: вартість обробки одного звернення автоматизованою системою є в кілька разів нижчою, ніж за участі оператора-людини [1].

Під інтелектуальною системою розуміють сукупність програмних та апаратних засобів, що реалізують функції сприйняття, обробки та інтерпретації інформації з метою формування обґрунтованих рішень без постійного втручання людини-оператора [2]. На відміну від традиційних систем за жорстко визначеними алгоритмами «якщо, то», інтелектуальні системи характеризуються здатністю до навчання, адаптації та контекстного розуміння вхідних запитів.

Еволюція таких систем умовно пройшла три етапи. Першим поколінням стали експертні системи 1980-1990-х років із формалізованими базами знань та правилами логічного виведення; їхня принципова слабкість полягала у так званій «проблемі вузького місця знань», тобто складності формалізації й оновлення правил [2]. Друге покоління (2000-2010-ті) спиралося на методи машинного навчання та статистичного аналізу, однак залишалося обмеженим у роботі з природною мовою та слабо утримувало контекст діалогу. Сучасне, третє покоління побудоване на великих мовних моделях (LLM), технології RAG та мультиагентних архітектурах і здатне генерувати зв’язний, контекстуально адекватний текст у відкритих доменах [2]. Узагальнену схему цієї еволюції подано на рис. 1.2.

Рис. 1.2 Еволюція інтелектуальних систем автоматизації послуг: від експертних систем до мультиагентних LLM-платформ

Сучасні інтелектуальні системи класифікуються за ступенем автономності (системи підтримки рішень, напівавтономні та повністю автономні), за типом інформації (текстові, голосові, мультимодальні) та за архітектурою (монолітні та мультиагентні) [2].

Мультиагентні системи найбільш повно відповідають складності реальних бізнес-процесів. Порівняльний аналіз [2] підтверджує їхні переваги: гнучкість, здатність до самоорганізації, відмовостійкість та модульність. Бідочко та Виклюк запропонували архітектуру LLMAgentNet, тобто колаборативну мережу автономних LLM-агентів із оркестратором завдань, RESTful-комунікацією та векторною пам’яттю для складних багатокрокових сценаріїв [5]. У промисловій практиці такі підходи застосовуються у торгівлі, логістиці, робототехніці, телекомунікаціях та енергетиці [2].

Ключовим елементом мультиагентної системи є механізм оркестрації, тобто компонент, який визначає, який саме агент має обробити

конкретний запит, керує послідовністю їхньої взаємодії та забезпечує передачу контексту між ними. Підходи до оркестрації варіюються від централізованих, де єдиний диспетчер-координатор розподіляє завдання, до децентралізованих, у яких агенти домовляються про розподіл ролей через обмін повідомленнями. У сучасних LLM-фреймворках, таких як LangGraph і CrewAI, оркестрація зазвичай реалізується як граф станів або як ієрархія ролей із менеджером-координатором, що дозволяє гнучко комбінувати детермінованість процесу з автономністю окремих агентів.

Завдяки трансформерним архітектурам [3] та великим мовним моделям системи здатні розуміти контекст, виявляти наміри та генерувати відповіді рівня кваліфікованого спеціаліста. За бенчмарком MMLU [20], провідні LLM досягають точності 86-90% у широкому спектрі предметних задач.

Технологія RAG (Retrieval-Augmented Generation) поєднує генеративні здібності LLM із динамічним доступом до зовнішніх джерел знань: перед формуванням відповіді модель отримує релевантні фрагменти з векторної бази або корпоративного архіву документів і використовує їх як контекст. Такий підхід дозволяє відокремити логіку обробки запиту від його знаннєвої основи, спрощує оновлення інформації без перенавчання моделі та підвищує достовірність відповідей у вузькоспеціалізованих доменах. За даними [4, 22], застосування RAG зменшує частку галюцинацій з 15-25% до 3-5%, що є критичним для сфер із високою ціною помилки, зокрема медицини, фінансів і страхування.

За даними Accenture [6], переважна більшість організацій підтверджують, що інвестиції у генеративний ШІ виправдали їхні очікування, а компанії з модернізованими на основі ШІ процесами демонструють помітно вищі темпи зростання доходів порівняно з конкурентами. Окремо у страховому сегменті близько 81% керівників

вважають, що ШІ не скоротить персонал, а перерозподілить його функції, створивши нові ролі та забезпечивши додатковий приріст ефективності [6].

Подібні тенденції спостерігаються і на національному рівні. Дослідження цифровізації страхового ринку України [7] показують, що в умовах розвитку ШІ на передній план виходять автоматизація обслуговування, персоналізація продуктів і підвищення якості даних для оцінки ризиків, тоді як регуляторні обмеження, вартість впровадження та вимоги до захисту персональної інформації залишаються стійкими стримуючими чинниками. Водночас галузеві огляди фіксують зростання ролі компаньйонських тварин у домогосподарствах і відповідне збільшення попиту на послуги, де страхова та ветеринарна тематики природно перетинаються в межах одного звернення [25]. Такі сценарії добре узгоджуються саме з мультиагентним підходом: окремі предметні компетенції (страхова й ветеринарна) можуть бути винесені в самостійних агентів без втрати цілісності діалогу й без різкого збільшення складності окремо взятої моделі.

До основних викликів належать забезпечення достовірності генерованих відповідей (схильність LLM до галюцинацій [26]), захист персональних даних, інтерпретованість рішень та специфічні ризики: висока вартість розробки, ймовірність непередбачуваної поведінки та складність гарантування надійності [2, 5].

Таким чином, інтелектуальні системи поступово переходять з ролі допоміжного інструмента до становища основної виконавчої ланки в типових сервісних процесах: вони беруть на себе первинну обробку звернень, пошук потрібної інформації та формування відповіді, залишаючи людині-оператору переважно складні й нестандартні випадки. Саме поєднання LLM, RAG та мультиагентних архітектур формує технологічну базу, яка робить такий перехід можливим, і водночас визначає практичні

вимоги до архітектурних рішень, розглянутих у наступному підрозділі [1, 6].

1.2. Аналіз існуючих підходів до автоматизації та актуальність мікросервісних та агентних архітектур

Автоматизація бізнес-процесів пройшла кілька етапів. Автоматизація на основі правил: за оцінками Deloitte [8], понад 60% підприємств використовували такі системи, проте лише 23% вважали їх достатньо гнучкими. Роботизована автоматизація процесів (RPA) орієнтована на структуровані сценарії та операції за жорсткими скриптами, що обмежує її застосування у роботі з неструктурованими запитами природною мовою [8]. Діалогові системи першого покоління на основі шаблонних сценаріїв, за даними McKinsey [1], розв’язують лише від 40 до 55 відсотків звернень.

Монолітна архітектура характеризується єдиним процесом розгортання та нерозривним зв’язком між модулями [9], тоді як мікросервісна [10] передбачає декомпозицію на незалежно розгортані сервіси, а агентна архітектура розширює мікросервісний підхід, наділяючи компоненти автономністю та проактивністю. За М. Вулдріджем [11], інтелектуальний агент характеризується автономністю, реактивністю, проактивністю та соціальністю. Фреймворки Amazon Bedrock Agents [13], LangGraph та CrewAI забезпечують інфраструктуру для мультиагентних систем на основі LLM. Візуальне порівняння трьох архітектурних підходів наведено на рис. 1.3.

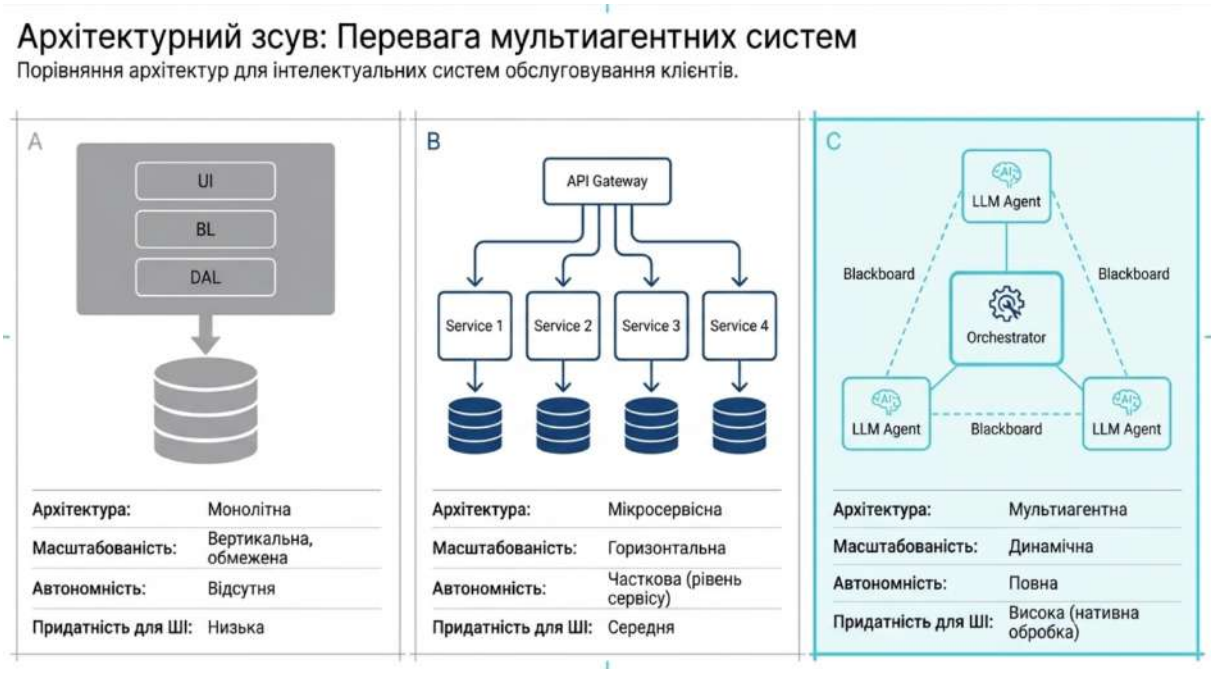


Рис. 1.3 Порівняння монолітної, мікросервісної та мультиагентної архітектур для систем обслуговування клієнтів

Наведена схема підкреслює принципові відмінності у розподілі відповідальності, межах автономії компонентів і типових потоках обробки звернень між трьома підходами. Деталізований порівняльний аналіз тих самих архітектур за шістьма критеріями: масштабованість, автономність компонентів, обробка неструктурованих запитів, стійкість до відмов, додавання нових функцій та придатність для ШІ-задач, систематизовано у табл. 1.1.

Табл. 1.1

Порівняльний аналіз архітектурних підходів для інтелектуальних систем обслуговування

Критерій	**Монолітна**	**Мікросервісна**	**Мультиагентна**
Масштабованість	Вертикальна, обмежена	Горизонтальна, незалежна	Горизонтальна, динамічна
Автономність компонентів	Відсутня	Часткова	Повна
Обробка неструктурованих запитів	Обмежена	Залежить від реалізації	Нативна через LLM
Стійкість до відмов	Єдина точка відмови	Ізоляція на рівні сервісів	Ізоляція + перенаправлення
Додавання нових функцій	Зміна всього коду	Новий сервіс	Новий агент
Придатність для ШІ-задач	Низька	Середня	Висока

Джерело: складено автором на основі [8, 9, 10, 11, 12]

Мультиагентна архітектура найкраще відповідає вимогам інтелектуальної системи обслуговування. Практична реалізація спирається на парадигму безсерверних обчислень [14]: за Datadog [15], понад 70% організацій застосовують безсерверні функції у виробничих системах.

Система має одночасно оперувати знаннями зі страхування та ветеринарії через спеціалізованих агентів. За NAPHIA [16], 62% звернень стосуються уточнення умов покриття, а 24% припадають на ветеринарну

консультацію, що підтверджує потенціал автоматизації мультиагентною системою. Розподіл типів звернень візуалізовано на рис. 1.4.

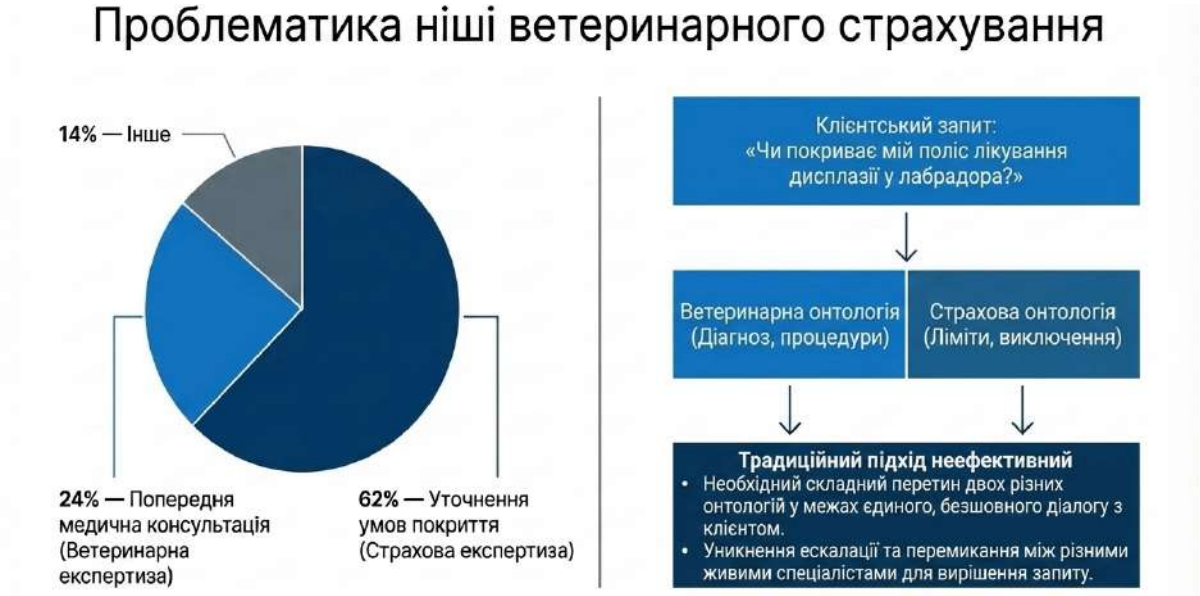


Рис. 1.4 Розподіл типів клієнтських звернень у ветеринарному страхуванні та потенціал їх автоматизації [16]

Аналіз ринку засвідчує, що більшість платформ ветеринарного страхування використовують традиційні кол-центри або прості чат-боти, не здатні забезпечити інтегровану страхово-ветеринарну консультацію [16]. Поєднання мікросервісної та агентної архітектур із безсерверною інфраструктурою є придатною архітектурною основою для платформи «ВетЕксперт».

Отже, еволюція від монолітних систем до мікросервісної та мультиагентної парадигм разом із безсерверними обчисленнями відповідає ключовим вимогам обслуговування у ветеринарному страхуванні: незалежне масштабування сервісів, автономність спеціалізованих агентів та зниження операційних витрат. Водночас реалізація такої архітектури неможлива без ефективних механізмів обробки природної мови та

протоколів узгодженої взаємодії агентів, що розкривається у наступному підрозділі.

1.3. Огляд методів обробки природної мови (NLP) та протоколів взаємодії агентів для предметної області

Трансформерна архітектура, запропонована А. Васвані та співавторами у 2017 р. [3], обробляє усі позиції вхідної послідовності паралельно завдяки механізму багатоголового самоуважнення (Multi-Head Self-Attention). На її основі побудовано кодувальні (BERT [18]) та декодувальні (GPT [19]) моделі; для діалогових систем найбільш релевантними є саме декодувальні, зокрема GPT-4 та Claude, які за бенчмарком MMLU досягають точності 86-90% [20].

Обробка клієнтських звернень потребує розпізнавання наміру та витягування сутностей (NER, named entity recognition) [17]. LLM вирішують обидві задачі через промптинг: in-context learning у режимі few-shot дає точність близько 82-88% без донавчання, а після налаштування промпту показники сягають 91-94% [19]. У системі «ВетЕксперт» класи намірів узгоджені з бізнес-правилом R-1 і формалізовані як чотири дискретні мітки (детально описані у підрозділі 2.3). Приклад обробки типового запиту цієї предметної області наведено на рис. 1.5.

Рис. 1.5 Процес розпізнавання наміру та витягування сутностей для запиту у сфері ветеринарного страхування

RAG-архітектура [4] складається з модуля індексації документів (векторні представлення у базі даних), модуля пошуку (retriever) та генеративної моделі. Її якість оцінюють трьома метриками: повнота пошуку (Recall), точність генерації (Faithfulness) та релевантність відповіді (Answer Relevance); оптимізована система досягає Recall 85-93% та Faithfulness 90-96% [22].

Для взаємодії між агентами класичним стандартом залишається протокол FIPA ACL [23], однак сучасні LLM-платформи частіше використовують координацію через оркестратор на основі промптів [12, 24]. У такій моделі виділяють три патерни: централізовану оркестрацію, ланцюгову передачу та паралельне виконання. Практичним прикладом є Amazon Bedrock Agents [13], де централізована маршрутизація намірів працює з латентністю 2-4 с [24].

У «ВетЕксперті» знання розділено на дві спеціалізовані онтології: страхову та ветеринарну, побудовані з орієнтацією на вимоги RAG-

заземлення [22]. Для достовірності відповідей застосовується принцип grounding: кожне твердження підтверджується фрагментом із бази знань, що, за даними [22], підвищує рівень довіри користувачів на 40-60%. Концептуальну модель, яка поєднує описані NLP-компоненти з агентною архітектурою платформи, подано на рис. 1.6.

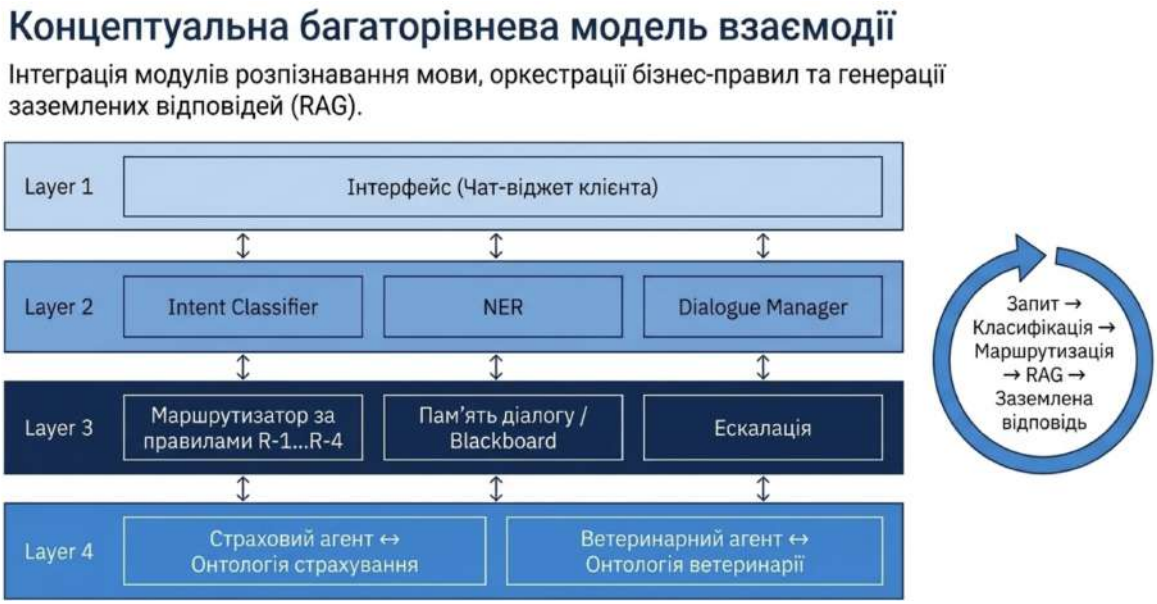


Рис. 1.6 Концептуальна модель взаємодії NLP-компонентів та агентів платформи «ВетЕксперт»

Разом з тим, практичне застосування LLM має низку обмежень. Найвідомішою є проблема галюцинацій: навіть GPT-4 та Claude 3.5 дають 3-8% помилок на спеціалізованих запитах, що робить обов'язковим використання RAG та верифікації [26]. У тривалих діалогах до цього додається зміщення контексту, яке частково знімається керуванням вікном контексту та резюмуванням історії [12]. Окремо слід згадати мовний бар'єр: для менш представлених мов, зокрема української, якість генерації зазвичай нижча за англомовну; точні значення залежать від моделі та навчального корпусу, тому далі в роботі ця відмінність враховується як якісне проєктне обмеження [12, 17].

Таким чином, поєднання декодувальних LLM, RAG та централізованої агентної оркестрації дозволяє досягти точності розпізнавання намірів на рівні 91-94% і втримати частку галюцинацій у межах 3-5%, чого достатньо для діалогової системи у сфері ветеринарного страхування. Водночас виявлені обмеження, зокрема зміщення контексту, мовний бар'єр і варіативність генерації, формують перелік ризиків, які далі враховуються при постановці задачі та формулюванні технічних вимог до «ВетЕксперта».

1.4. Постановка задачі дослідження та формування технічних вимог до системи «ВетЕксперт»

Існує суттєвий розрив між пріоритетами цифровізації страхових сервісів і впровадження ШІ, окресленими для вітчизняного ринку [7], та обмеженою ефективністю поширених шаблонних рішень (40-55% звернень за даними опитувань [1]). Як показано у підрозділі 1.3, серед проаналізованих платформ жодна не поєднує страхову та ветеринарну експертизу в єдиному діалоговому інтерфейсі. Водночас LLM (86-90% за MMLU [20]), RAG (галюцинації 3-5% [4, 22]) та мультиагентні архітектури (латентність 2-4 с [24]) забезпечують технічну базу, достатню для автоматизації обслуговування у зазначених доменах, проте комплексних рішень, що поєднують страхову та ветеринарну експертизу в єдиній мультиагентній діалоговій системі, на практиці ще немає.

Метою дослідження є розробка та практична реалізація інтелектуальної хмарної платформи медичного страхування домашніх тварин «ВетЕксперт» на основі взаємодії спеціалізованих ШІ-агентів. Для досягнення мети визначено такі задачі:

1) формалізувати бізнес-процеси обслуговування клієнтів у нотаціях AS-IS та TO-BE;

2) спроєктувати архітектуру мультиагентної системи;

3) формалізувати алгоритми прийняття рішень та маршрутизації запитів;

4) обґрунтувати вибір технологічного стека;

5) реалізувати серверну логіку та механізм взаємодії агентів;

6) розробити клієнтський інтерфейс платформи;

7) провести тестування та оцінити ефективність запропонованих рішень.

Об’єктом дослідження виступає процес автоматизації обслуговування клієнтів у сфері медичного страхування домашніх тварин, предметом дослідження є методи та засоби побудови мультиагентної хмарної платформи на основі LLM, RAG та безсерверної інфраструктури. Методологічну основу складають: системний аналіз [27], нотація моделювання бізнес-процесів (BPMN) [28], формалізація алгоритмів, UML-проєктування та емпіричні методи тестування [27].

Конкурентною перевагою «ВетЕксперт» порівняно з проаналізованими у підрозділі 1.3 рішеннями є інтеграція страхового та ветеринарного доменів у межах однієї мультиагентної платформи. Функціонально платформа забезпечує три блоки: страхове консультування, яке передбачає інформування про поліси та тарифи, розрахунок вартості, перевірку покриття та супровід подання страхового випадку; ветеринарне консультування, що охоплює оцінку стану здоров’я за симптомами, рекомендації щодо першої допомоги й інформування про захворювання та процедури; а також оркестрацію діалогу, яка відповідає за визначення наміру, маршрутизацію до профільного агента, безшовну передачу контексту та автоматичну ескалацію до оператора.

Кожен блок реалізується окремим агентом із власною RAG-базою знань. Діаграма на рис. 1.7 ілюструє функціональну декомпозицію.

Рис. 1.7 Функціональна декомпозиція платформи «ВетЕксперт»: зони відповідальності спеціалізованих агентів та потоки даних

До платформи висуваються чотири групи нефункціональних вимог. За показником продуктивності час однієї відповіді агента не повинен перевищувати 10 с для типових запитів (p95) та 15 с для мультидисциплінарних, а повний цикл обслуговування (3-5 обмінів) становить 2 та 3 хв відповідно. За масштабованістю система має підтримувати від 10 до 1000 одночасних сесій. Достовірність задається граничним рівнем помилок $\leq 5\%$ за рахунок RAG-заземлення, а доступність забезпечується режимом 24/7 із рівнем надання послуги $\geq 99{,}5\%$. До технологічних обмежень віднесено залежність від зовнішніх LLM-провайдерів, обмеження контекстного вікна моделі та необхідність захисту персональних даних [26]. Повну систематизацію сформульованих вимог представлено на рис. 1.8.

Рис. 1.8 Систематизація функціональних та нефункціональних вимог до платформи «ВетЕксперт»

Основні архітектурно-алгоритмічні рішення роботи, які реалізуються у другому та третьому розділах, є такими: (а) схема контекстно-залежної класифікації клієнтських намірів і маршрутизації запитів між спеціалізованими агентами страхового та ветеринарного доменів із підтримкою мультидисциплінарних звернень; (б) алгоритм оцінки терміновості ветеринарних звернень за чотирирівневою шкалою на основі аналізу симптомів та профілю тварини; (в) схема інтеграції RAG-конвеєра у мультидоменну діалогову систему з міжагентною передачею контексту через blackboard pattern [22, 31]. Очікуваний ефект від їх впровадження полягає у зниженні вартості обробки звернення в 5-8 разів порівняно з оператором-людиною [1], підвищенні ефективності страхових процесів за рахунок генеративного ШІ [6] та цілодобовій доступності консультацій.

Отже, у підрозділі сформульовано проблему дослідження, визначено мету, об’єкт та предмет, деталізовано сім задач, окреслено функціональні блоки платформи «ВетЕксперт», систематизовано нефункціональні вимоги та зафіксовано ключові архітектурно-алгоритмічні рішення, що становлять

основні результати роботи. Сформульовані вимоги задають критерії, за якими у наступному розділі здійснюватиметься системний аналіз предметної області та проєктування архітектури мультиагентної системи, починаючи з формалізації бізнес-процесів у нотаціях AS-IS / TO-BE.

РОЗДІЛ 2. СИСТЕМНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПЛАТФОРМИ

2.1. Формалізація бізнес-процесів страхування та ветеринарного консалтингу

У першому розділі було визначено предметну область, проаналізовано існуючі рішення та сформульовано функціональні й нефункціональні вимоги до платформи «ВетЕксперт». Для переходу від концептуальних вимог до проєктних рішень необхідно формалізувати бізнес-процеси, що підлягають автоматизації. У цьому підрозділі застосовано методологію порівняльного аналізу AS-IS / TO-BE із використанням нотації BPMN 2.0 [28], яка дозволяє структуровано описати поточний стан обслуговування клієнтів, виявити вузькі місця та обґрунтувати доцільність впровадження мультиагентної системи.

Об’єктом формалізації є три ключові бізнес-процеси: консультація щодо страхового покриття, ветеринарна консультація за симптомами та подання повідомлення про страховий випадок. Для кожного з них визначено склад учасників, послідовність операцій, часові характеристики та точки неефективності у поточному стані, а також відповідні трансформації у цільовому стані з інтеграцією ШІ-агентів. Першим кроком є ідентифікація акторів та систематизація їхніх ролей в обох станах, що подано у табл. 2.1.

Табл. 2.1

Учасники бізнес-процесів страхування та ветеринарного консалтингу

Актор	**Роль у процесі AS-IS**	**Роль у процесі TO-BE**
Клієнт (власник тварини)	Ініціює звернення телефоном, поштою або через веб-форму	Ініціює звернення через чат-інтерфейс; отримує миттєву відповідь
Оператор кол-центру	Приймає звернення, класифікує, маршрутизує до спеціаліста	Рутинні функції делегуються агенту-оркестратору; фокусується на ескалаціях
Страховий консультант	Консультує щодо полісів, тарифів; перевіряє умови договорів	Типові запити обробляються ШІ-агентом з RAG-базою знань
Ветеринарний спеціаліст	Надає медичні консультації; оцінює симптоми	Типові запити обробляються ШІ-агентом з RAG-базою
Андерайтер	Оцінює ризики; затверджує відшкодування	Залишається; отримує структуровані дані від ШІ-системи

Джерело: складено автором на основі [16, 28, 30]

У поточному стані до обслуговування клієнта залучають три взаємопов’язані бізнес-процеси, які доповнюють один одного в рамках єдиної лінії звернень. Першим процесом виступає консультація щодо страхового покриття: оператор приймає звернення, класифікує та переадресовує консультанту, який здійснює пошук у базі полісів; середній час становить 15-25 хв, а до 40% припадає на очікування [16]. Далі йде

ветеринарна консультація за симптомами, коли оператор передає звернення ветеринару, а за потреби уточнення щодо покриття виконується повторна маршрутизація; у таких умовах загальний час становить 30-45 хв [16]. Третім елементом ланцюга є подання повідомлення про страховий випадок: оператор реєструє звернення, передає консультанту матеріали для формування переліку документів і висновку для андерайтера, причому до 60% часу займає збір документів [8, 30]. Узагальнену BPMN-діаграму, що відображає зазначені процеси у поточному стані, наведено на рис. 2.1.

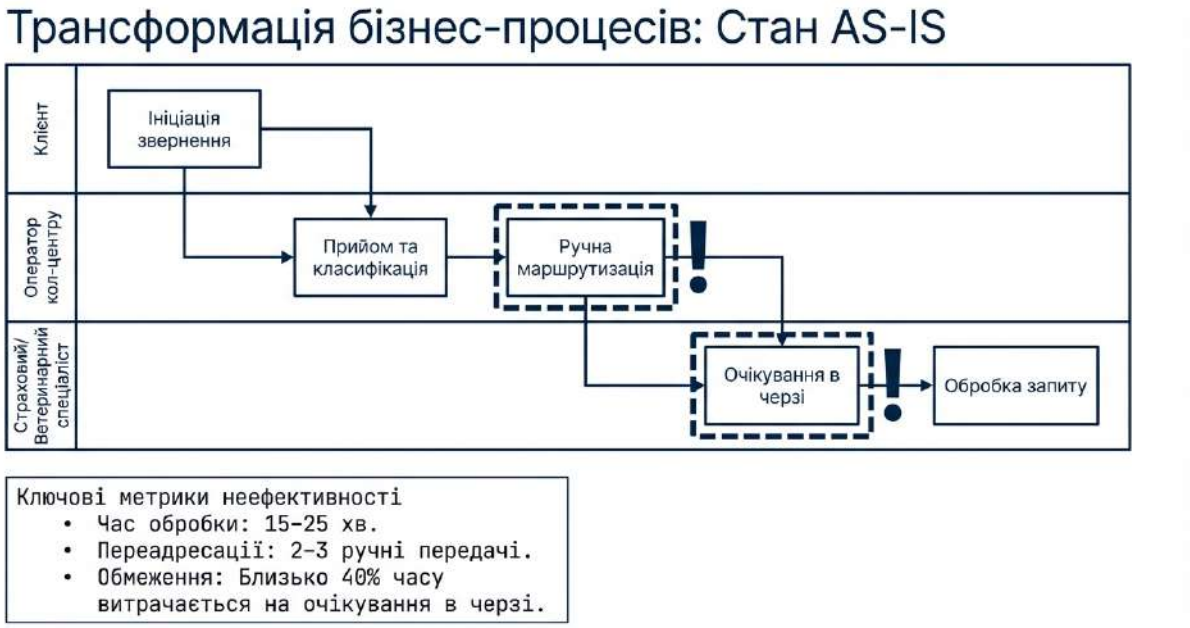


Рис. 2.1 Узагальнена BPMN-діаграма бізнес-процесу обслуговування клієнтів у стані AS-IS

У цільовому стані агент-оркестратор аналізує запит, розпізнає намір (R-1) і маршрутизує його до страхового агента (R-2), який виконує семантичний пошук у RAG-базі з цитуванням (R-3); час однієї відповіді становить 5-15 с [13, 24], а повний цикл консультації щодо покриття (3-5 обмінів: уточнення полісу, пошук, відповідь) триває 1-2 хв. Для ветеринарної допомоги оркестратор направляє звернення агенту-ветеринару, який аналізує симптоми й формує оцінку із застереженням (R-

5); коли з’являється запит про покриття, контекст безшовно передається страховому агенту. Час однієї відповіді в цьому контурі становить 8-15 с [13], а повний цикл (4-6 обмінів: уточнення симптомів, оцінка терміновості, рекомендації, за потреби залучення страхового агента) триває 2-3 хв. Окремим сценарієм є подання повідомлення про страховий випадок: у ньому страховий агент через серію уточнюючих питань збирає обов’язкові атрибути страхового випадку, зокрема дату інциденту, опис обставин, перелік проведених ветеринарних процедур і первинних діагнозів, після чого перевіряє чинність полісу клієнта та відповідність випадку умовам покриття відповідно до правила R-6. На основі зібраних даних агент формує попередній перелік підтверджувальних документів (рахунки ветеринарної клініки, медичні висновки, а також за потреби фото- чи відеоматеріали) і передає структурований звіт андерайтеру у форматі, придатному для автоматизованої перевірки. Такий підхід усуває повторні звернення для донесення документів і скорочує час обробки повідомлення на 60-70% [6, 8]. Відповідну BPMN-діаграму цільового стану з мультиагентною платформою подано на рис. 2.2.

Рис. 2.2 Узагальнена BPMN-діаграма бізнес-процесу обслуговування клієнтів у стані TO-BE із мультиагентною платформою «ВетЕксперт»

Діаграма відображає скорочення ручних переадресацій, перехід рутинних операцій до агентів і збереження ролі людини переважно на ескалаціях та андерайтингу. Щоб узагальнити ефект трансформації в числовій формі, ключові показники тривалості циклів, доступності, паралелізму сесій, частки автоматизованих звернень та вартості одного звернення зіставлено для моделей AS-IS та TO-BE у табл. 2.2.

Табл. 2.2

Порівняльний аналіз показників бізнес-процесів AS-IS та TO-BE

Показник	**AS-IS**	**TO-BE**	**Покращення**
Середній час обробки типового запиту (повний цикл)	15-25 хв	1-2 хв (3-5 обмінів по 5-15 с)	У 8-25 разів
Середній час мультидисциплінарного запиту (повний цикл)	30-45 хв	2-3 хв (4-6 обмінів по 8-15 с)	У 10-22 рази
Кількість переадресацій між спеціалістами	2-3	0 (автоматична оркестрація)	100% усунення
Доступність обслуговування	8-12 год/добу	24/7/365	Цілодобова
Кількість одночасних сесій	1-3	10-100 (у межах квот AWS Bedrock)	На порядок вище
Частка звернень без залучення людини	0%	70-85% (авторська оцінка[1])	-
Вартість обробки одного звернення	5-15 дол. США	0,04 дол. США (з амортизацією розробки; розрахунок наведено у підрозділі 3.4)	У 125-375 разів

Оцінка виведена на основі розподілу типів звернень за даними NAPHIA [16]: 62% звернень стосуються уточнення умов покриття (типові FAQ-запити з високим потенціалом автоматизації), 24% становлять ветеринарні консультації (частково автоматизовуються через RAG), 14% припадають на складні випадки, що потребують людського втручання. При коефіцієнті успішності ШІ-автоматизації 80-90% для FAQ-запитів [1] та 60-70% для ветеринарних консультацій [4, 22] сукупна частка повністю

автоматизованих звернень оцінюється як зважене середнє за типами звернень:

$$A_{full} = \sum_{i=1}^{3} w_i a_i = w_1 a_1 + w_2 a_2 + w_3 a_3 \approx 0{,}68 \quad (68\%) \qquad (1.1)$$

де A_{full}: очікувана частка звернень, повністю оброблених без залучення людини; w_i: частка звернень типу i у загальному обсязі, $\sum_{i=1}^{3} w_i = 1$ (за NAPHIA: $w_1 = 0{,}62$, $w_2 = 0{,}24$, $w_3 = 0{,}14$); a_i: коефіцієнт повної ШІ-автоматизації для типу i (частка кейсів типу i, які завершуються без ескалації до оператора): для FAQ узято верхню межу діапазону 0,80-0,90, $a_1 = 0{,}85$; для ветеринарних консультацій взято значення в межах 0,60-0,70, $a_2 = 0{,}65$; для складних випадків $a_3 = 0$.

З урахуванням поступового вдосконалення бази знань та промптів верхня межа оцінюється у 85%, що формує діапазон 70-85%.

Джерело: складено автором на основі [1, 6, 8, 13, 16]

Окрім структури процесів та розподілу ролей, автоматизована система потребує чітко визначених інваріантів, що гарантують коректність прийняття рішень на кожному етапі обробки звернення. Такі інваріанти формалізовано у вигляді бізнес-правил (табл. 2.3), які охоплюють класифікацію намірів, маршрутизацію запитів, заземлення відповідей на верифіковані джерела та умови ескалації до оператора-людини.

Табл. 2.3

Бізнес-правила платформи «ВетЕксперт»

Код	**Назва правила**	**Опис**	**Пріоритет**
R-1	Класифікація наміру	Кожен запит класифікується за категоріями намірів оркестратором	Критичний
R-2	Маршрутизація до агента	Запит маршрутизується до агента відповідно до наміру	Критичний
R-3	Заземлення відповідей	Фактичні твердження підтверджуються фрагментом із RAG-бази	Критичний
R-4	Ескалація до оператора	При впевненості < 0,7 або запиті клієнта відбувається ескалація до людини	Високий
R-5	Медичне застереження	Ветеринарна консультація супроводжується застереженням	Критичний
R-6	Перевірка чинності полісу	Перед наданням інформації перевіряється чинність полісу у БД	Високий
R-7	Збереження контексту	Контекст діалогу зберігається та передається між агентами	Високий

Джерело: складено автором на основі [13, 16, 22, 26]

Бізнес-правила задають послідовність перевірок оркестратора від класифікації наміру до рішення про ескалацію; цю логіку візуалізовано як дерево прийняття рішень на рис. 2.3.

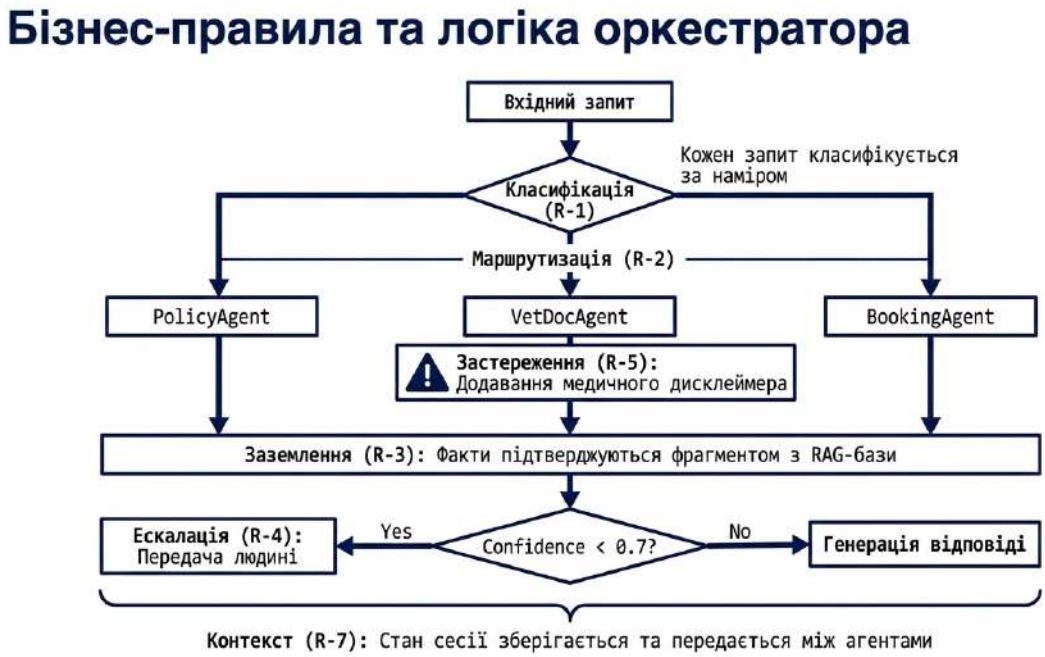


Рис. 2.3 Дерево прийняття рішень оркестратора платформи «ВетЕксперт» на основі бізнес-правил від R-1 до R-7

Платформа надає ветеринарні та страхові консультації через ШІ-агентів, що створює специфічні ризики, які потребують системної мітигації ще на рівні проєктування.

Насамперед слід розглянути ризик помилкової медичної поради. ШІ-агент може сформувати некоректну оцінку стану тварини, зокрема занизити рівень терміновості (наприклад, призначити u_4 замість u_1), що потенційно затримує звернення власника до клініки. Для мінімізації цього ризику бізнес-правило R-5 вимагає обов'язкового застереження при кожній ветеринарній відповіді, а алгоритм оцінки терміновості (рівняння (2.7)) реалізовано за консервативною стратегією: наявність будь-якого критичного симптому з множини E_critical автоматично призначає запиту найвищий рівень u_1 (emergency) незалежно від інших ознак.

Окрему категорію становлять маніпулятивні промпти (prompt injection), тобто спроби зловмисника обійти системну інструкцію агента через спеціально сформульований запит. Архітектурне рішення тут

ґрунтується на двох принципах: системні інструкції агентів визначено декларативно у файлі instruction.txt і не модифікуються під час виконання, а самі агенти обмежені набором дозволених інструментів (Action Groups), що унеможливлює доступ до даних інших клієнтів або сторонніх ресурсів.

Ще один клас проблем пов’язаний із некоректною оцінкою впевненості класифікатора намірів: при низькому значенні впевненості запит може бути спрямований до невідповідного агента, що призведе до хибної відповіді. Для запобігання таким сценаріям застосовано бізнес-правило R-4, згідно з яким при впевненості нижче порогового значення 0,7 запит автоматично ескалюється до оператора-людини.

Нарешті, важливим є правовий статус AI-консультації. Платформа позиціонується як інформаційний помічник, а не заміна ліцензованого ветеринарного спеціаліста; усі відповіді ветеринарного агента супроводжуються відповідним застереженням (R-5), а повна історія діалогів зберігається у сховищі ChatHistory для подальшого аудиту та аналізу інцидентів.

На основі аналізу предметної області виділено шість основних сутностей із ключовими атрибутами та зв’язками (табл. 2.4).

Табл. 2.4

Основні сутності предметної області та їхні атрибути

Сутність	**Ключові атрибути**	**Зв’язки**
Policy	policyId, planType, startDate, endDate, coverageLimit, deductible, exclusions, status	Customer (N:1), Pet (1:1), Claim (1:N)
Pet	petId, species, breed, birthDate, weight, sex, chronicConditions, vaccinationHistory	Policy (1:1), Customer (N:1)
Customer	customerId, name, email, phone, accountId	Policy (1:N), Pet (1:N), DialogSession (1:N)
Claim	claimId, filingDate, incidentDate, description, status, approvedAmount	Policy (N:1), Pet (N:1), MedicalProcedure (N:M)
MedicalProcedure	procedureId, name, category, estimatedCostRange, relatedDiagnoses	Claim (N:M)
DialogSession	sessionId, messages, detectedIntents, involvedAgents, status, qualityScore	Customer (N:1)

Джерело: складено автором

Зв’язки між сутностями, їхню кардинальність та напрямок залежностей подано на концептуальній ER-діаграмі (рис. 2.4).

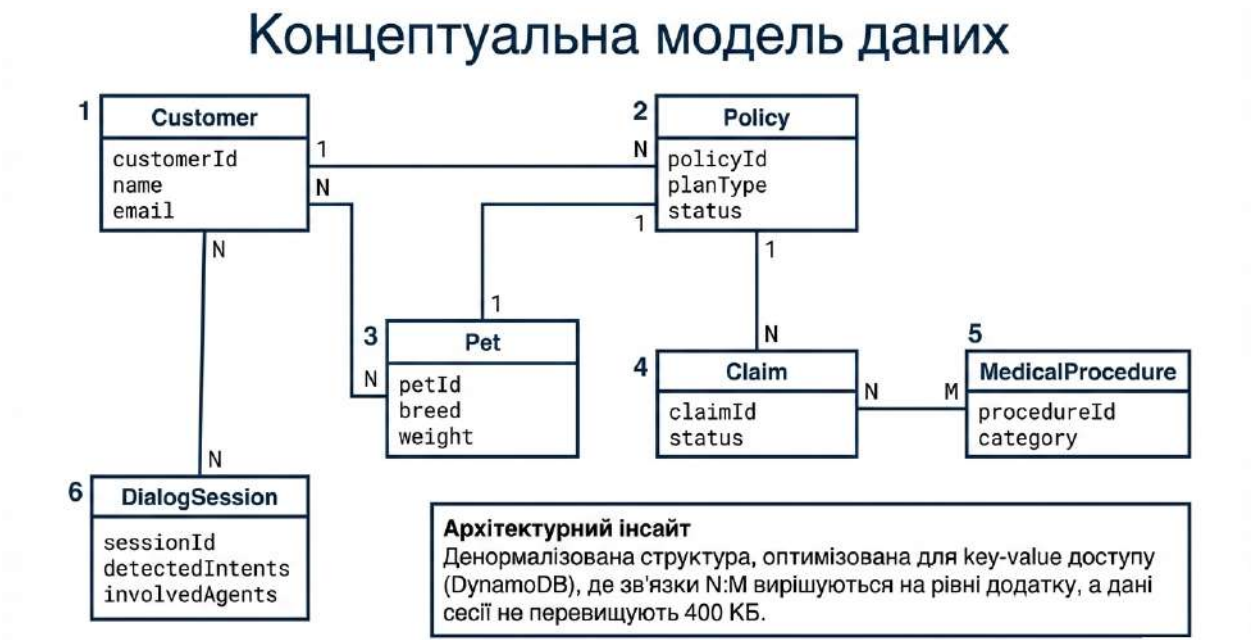


Рис. 2.4 Концептуальна ER-діаграма предметної області платформи «ВетЕксперт»

Отже, формалізовано бізнес-процеси у нотаціях AS-IS та TO-BE. Порівняння засвідчило потенціал скорочення часу повного циклу обслуговування у 8-25 разів та вартості у 125-375 разів (з урахуванням амортизації розробки). Визначено сім бізнес-правил та шість сутностей.

2.2. Проєктування архітектури мультиагентної системи

Проєктування архітектури поєднує безсерверну мікросервісну, мультиагентну та подієво-орієнтовану парадигми. В основу покладено п’ять принципів. Принцип єдиної відповідальності означає, що кожен мікросервіс реалізує рівно одну функцію і не містить побічної логіки. Принцип незалежного масштабування передбачає розгортання кожного компонента як окремої функції AWS Lambda, що дозволяє масштабувати навантажувані вузли без впливу на суміжні сервіси. Принцип декларативної поведінки переносить бізнес-логіку агента з програмного коду у текстову інструкцію, яка передається LLM разом із переліком доступних інструментів. Принцип слабкої зв’язаності забезпечується відмовою від прямих викликів між агентами на користь координації через спільне

сховище контексту за патерном blackboard [31]. Нарешті, принцип подієво-орієнтованої комунікації регламентує асинхронну доставку змін стану клієнту через окрему підсистему подій, а не через основну HTTP-відповідь.

Для огляду наскрізного ланцюга обробки запиту користувача наведено узагальнену компонентну схему (рис. 2.5): клієнтський застосунок на базі Nuxt звертається до Amazon API Gateway; далі запит обробляє сервіс маршрутизації на AWS Lambda, який залучає агент-класифікатор намірів (Intention Classifier) і один із доменних агентів платформи Amazon Bedrock (Policy Manager, Vet Doctor, Booking Manager). Кожен доменний агент викликає закріплені за ним інструменти у вигляді окремих Lambda-функцій для читання та запису даних у таблиці Amazon DynamoDB, для звернення до бази знань у ветеринарному сценарії та для публікації подій; останні через підсистему публікації та SSE-потік доставляються клієнту без додаткового ручного оновлення сторінки.

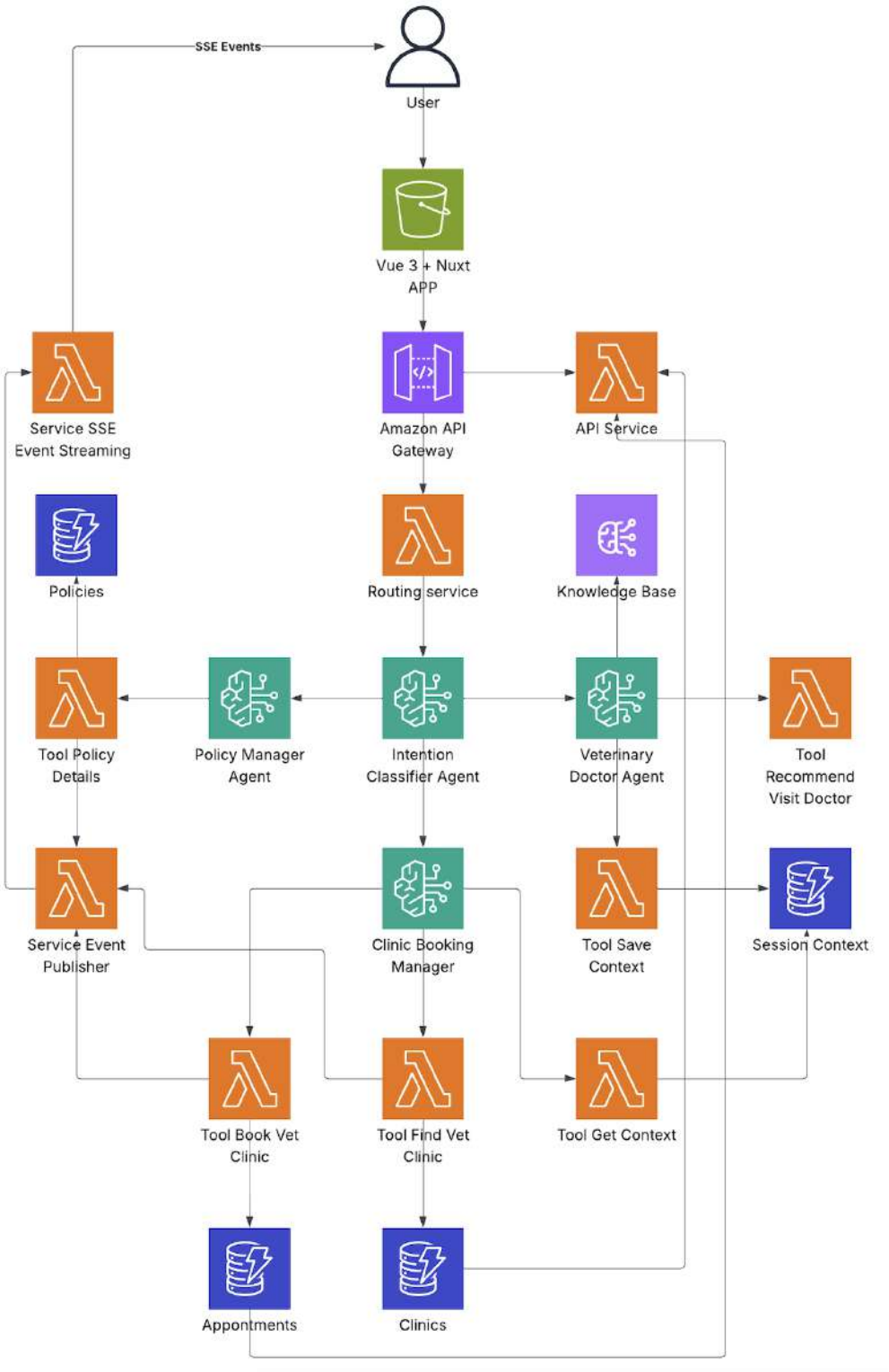


Рис. 2.5 Компонентна схема функціонування платформи «ВетЕксперт»

Загальну трирівневу організацію системи подано на рис. 2.6. Рівень представлення побудовано на фреймворку Nuxt 4 із використанням Tailwind CSS та сховища стану Pinia; статичні ресурси розміщуються в

Amazon S3. Рівень бізнес-логіки об’єднує три доменні ШІ-агенти (Policy Manager, Vet Doctor, Booking Manager) та шість Lambda-інструментів, які вони викликають для взаємодії з персистентним сховищем і зовнішніми даними. До інфраструктурного підрівня бізнес-логіки належать Service Router та агент-класифікатор намірів (Intention Classifier), що разом утворюють маршрутизаційну підсистему. Рівень даних об’єднує чотири таблиці Amazon DynamoDB, базу знань на основі Amazon S3 та Bedrock Knowledge Base, а також підсистему подій для асинхронної доставки повідомлень клієнту.

За функціональним призначенням агенти поділено на дві категорії. Доменні агенти (Policy Manager, Vet Doctor, Booking Manager) безпосередньо обслуговують запити клієнтів, оперуючи предметними знаннями страхового та ветеринарного доменів і викликаючи відповідні інструменти. Агент-класифікатор (Intention Classifier) не має власної предметної спеціалізації та реалізований як Bedrock Agent виключно для того, щоб використати LLM-класифікацію з few-shot промптингом замість евристичних правил чи окремої ML-моделі.

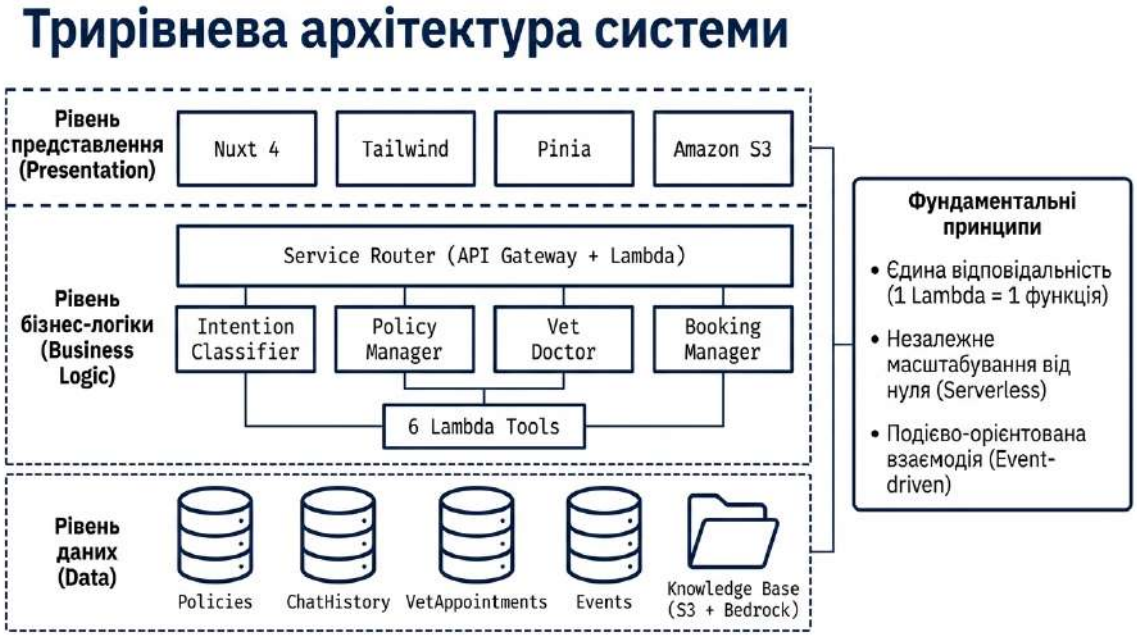


Рис. 2.6 Загальна трирівнева архітектура системи «ВетЕксперт»

Повну декомпозицію системи на 14 мікросервісів із зазначенням категорій, типів ресурсів AWS і функцій наведено у табл. 2.5.

Табл. 2.5

Декомпозиція системи на мікросервіси

Категорія	**Мікросервіс**	**Тип ресурсу**	**Функція**
Інфраструктурні сервіси	service-router	Lambda + API Gateway	Точка входу API, класифікація намірів, маршрутизація
	service-event-publisher	Lambda + API Gateway + DynamoDB	Прийом та збереження подій від інструментів
	service-sse-stream	Lambda Function URL (RESPONSE_STREAM)	Потокова доставка подій клієнту через SSE (події, надіслані сервером, Server-Sent Events)
	service-backend	Lambda + API Gateway	REST API для отримання записів на прийом
Агент-класифікатор	agent-intention-classifier	Bedrock Agent	Класифікація намірів (інфраструктурний, частина маршрутизації)
Доменні ШІ-агенти	agent-policy-manager	Bedrock Agent	Консультація щодо страхових полісів
	agent-vet-doctor	Bedrock Agent + Knowledge Base	Ветеринарна діагностична консультація
	agent-booking-manager	Bedrock Agent	Пошук клінік та бронювання прийомів

Категорія	**Мікросервіс**	**Тип ресурсу**	**Функція**
Інструменти агентів	tool-policy-details	Lambda (Action Group)	Отримання даних полісу з DynamoDB
	tool-context-details	Lambda (Action Group)	Читання контексту сесії з ChatHistory
	tool-context-save	Lambda (Action Group)	Збереження контексту сесії в ChatHistory
	tool-find-vet-clinic	Lambda (Action Group)	Пошук ветеринарних клінік
	tool-book-vet-clinic	Lambda (Action Group)	Створення запису на прийом
	tool-recommend-doctor-visit	Lambda (Action Group)	Публікація рекомендації візиту до лікаря

Джерело: складено автором

Кожен агент є ресурсом AWS::Bedrock::Agent із інструкцією, action groups та моделлю Claude 3.5 Haiku.

Агент класифікації намірів (Intention Classifier Agent) визначає намір клієнта (R-1) та повертає одну з чотирьох міток: PolicyAgent, VetDocAgent, BookingAgent або UNKNOWN. Мітка UNKNOWN транслюється Service Router у виняток AgentNotFoundException, який ініціює сценарій ескалації до оператора. Класифікація реалізована через few-shot промптинг із вісьмома прикладами для кожної доменної категорії.

Агент керування страховими полісами (Policy Manager Agent) за допомогою інструментів GetPolicyDetails та SaveContext витягує дані полісу (R-6), зберігає контекст (R-7), надає персоналізоване привітання та деталі покриття.

Агент ветеринарного лікаря (Vet Doctor Agent) виконує попередню діагностичну оцінку стану тварини. Він завантажує контекст сесії через GetContextDetails (R-7), у діалозі збирає перелік симптомів та анамнез,

виконує оцінку терміновості за чотирирівневою шкалою (R-5), а для формування діагностичної гіпотези звертається до RAG-бази знань з обов’язковим додаванням медичного дисклеймера (R-3). У разі потреби в очній консультації агент публікує відповідну рекомендацію через RecommendDoctorVisit, а зібраний медичний контекст зберігає через SaveContext для подальшого використання агентом бронювання.

Агент бронювання прийомів (Booking Manager Agent) послідовно застосовує інструменти GetContextDetails, FindVetClinic, SaveContext та BookVetClinic: завантажує контекст, підбирає клініки, після підтвердження користувача створює запис. Розподіл інструментів між агентами узагальнено у матриці (табл. 2.6).

Табл. 2.6

Матриця розподілу інструментів між агентами

Інструмент	**Intention Classifier**	**Policy Manager**	**Vet Doctor**	**Booking Manager**
GetPolicyDetails	-	✓	-	-
GetContextDetails	-	-	✓	✓
SaveContext	-	✓	✓	✓
RecommendDoctorVisit	-	-	✓	-
FindVetClinic	-	-	-	✓
BookVetClinic	-	-	-	✓
Knowledge Base (RAG)	-	-	✓	-

Джерело: складено автором

Оркестрація побудована за патерном Router Pattern. Service Router приймає вхідний HTTP POST, ініціалізує контекст сесії та зберігає повідомлення користувача в таблиці ChatHistory. Далі викликається агент-класифікатор для визначення наміру (R-1); залежно від отриманої мітки Service Router або маршрутизує запит до відповідного доменного агента разом із sessionState (R-2), або, у разі мітки UNKNOWN, повертає клієнту повідомлення про неможливість автоматизованого обслуговування із запрошенням до оператора. Відповідь доменного агента зберігається в історії сесії та повертається ініціатору запиту.

Для обміну проміжними результатами між доменними агентами використано спрощений варіант патерну blackboard [31]. Роль спільного

сховища («дошки») виконує виділений запис у таблиці ChatHistory, доступ до якого здійснюється через інструменти SaveContext та GetContextDetails, а функції централізованого контролера бере на себе Service Router на етапі маршрутизації. Запис контексту сесії містить сім ключів: policy_details, diagnosis, symptoms, assessment, urgency_level, visit_recommendation, selected_clinic.

Послідовність обміну контекстом наведено на рис. 2.7. Policy Manager першим записує у спільне сховище дані полісу клієнта. Vet Doctor при обробці медичного звернення зчитує цей запис для врахування умов страхового покриття, після чого долучає до нього результати діагностичної оцінки: перелік симптомів, рівень терміновості та рекомендацію щодо візиту. Booking Manager на фінальному етапі використовує обидва блоки контексту одночасно: страхові обмеження враховуються під час пошуку клінік, а медична оцінка визначає пріоритетність запису на прийом.

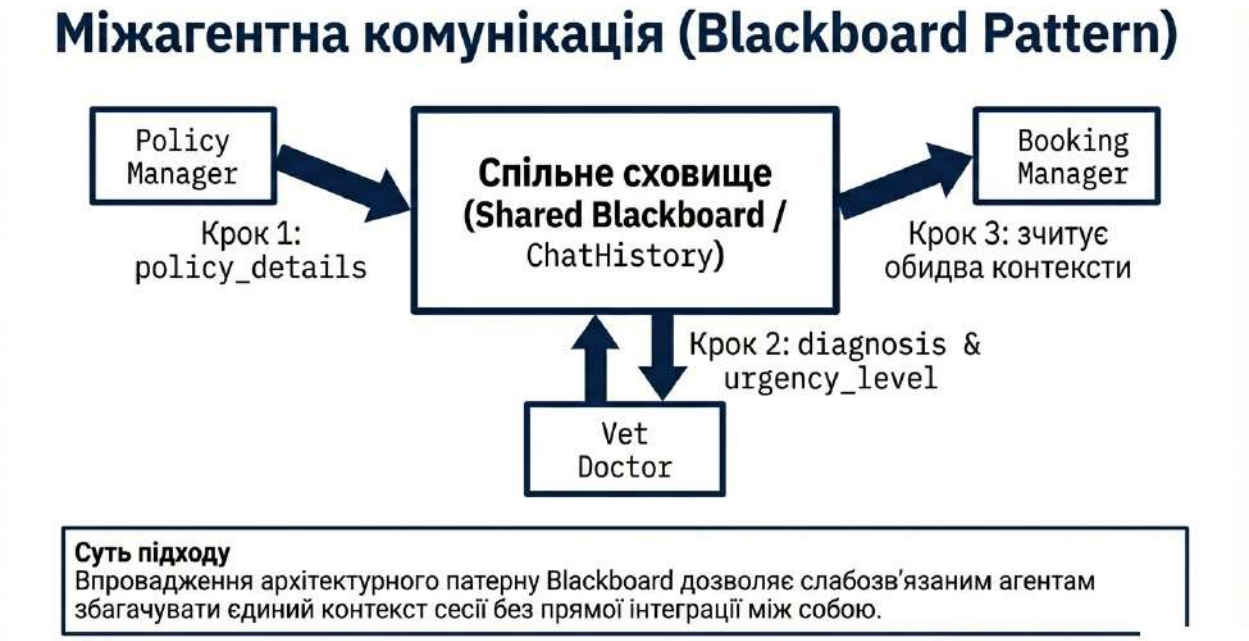


Рис. 2.7 Потік контексту між агентами через спільне сховище (blackboard pattern)

Підсистема подій реалізує чотириетапний ланцюжок доставки: інструмент агента формує подію та надсилає її до service-event-publisher,

який зберігає запис у таблиці Events; сервіс service-sse-stream із заданою періодичністю опитує цю таблицю і транслює нові записи клієнтові через канал SSE. Чотири типи подій, що передаються цим каналом, зведено у табл. 2.7.

Табл. 2.7

Типи подій підсистеми асинхронної доставки повідомлень

Тип події	**Джерело**	**Реакція клієнта**
AgentToolInvocation	Усі інструменти	Візуальна індикація активного інструменту
PolicyDetailsRetrieved	tool-policy-details	Оновлення профілю тварини у бічній панелі
RecommendDoctorVisit	tool-recommend-doctor-visit	Відображення картки рекомендації з кнопкою запису
AppointmentBooked	tool-book-vet-clinic	Відображення картки підтвердження з деталями

Джерело: складено автором

Персистентний рівень містить 4 таблиці DynamoDB (табл. 2.8).

Табл. 2.8

Таблиці DynamoDB системи «ВетЕксперт»

Таблиця	**Ключ партиціонування**	**Ключ сортування**	**Призначення**
Policies	policyId (S)	-	Страхові поліси, дані тварин, власників, покриття
ChatHistory	sessionId (S)	timestamp (N)	Історія діалогу та виділений запис контексту сесії
VetAppointments	appointmentId (S)	-	Записи на прийом до ветеринарних клінік
Events	partitionKey (S)	timestamp (N)	Події підсистеми для SSE-потоку

Джерело: складено автором

Для верифікації повноти архітектури ключові функціональні та нефункціональні вимоги з підрозділу 1.4 зіставлено з ухваленими архітектурними рішеннями та компонентами реалізації (табл. 2.9).

Табл. 2.9

Відповідність архітектурних рішень вимогам

Вимога	**Архітектурне рішення**	**Компоненти реалізації**
FR-01 (Класифікація намірів)	Виділений агент-класифікатор із few-shot інструкцією	Intention Classifier Agent
FR-03 (Інформація про поліс)	Спеціалізований агент із GetPolicyDetails	Policy Manager Agent + tool-policy-details
FR-04 (Діагностична оцінка)	Агент із підключеною базою знань для RAG	Vet Doctor Agent + Knowledge Base
FR-07, FR-08 (Пошук та бронювання)	Інструменти з урахуванням контексту терміновості	Booking Manager Agent + tool-find/book-vet-clinic
FR-09 (Спільний контекст)	Blackboard pattern через виділений запис у ChatHistory	tool-context-save + tool-context-details
FR-10 (SSE-потік подій)	Публікатор → сховище → потоковий доставник	Event Publisher + Events Table + SSE Stream
NFR-04, NFR-05 (Масштабованість)	Кожен компонент є окремою Lambda з автомасштабуванням	14 Lambda-функцій із окремими шаблонами AWS SAM (Serverless Application Model)
NFR-09 (Розширюваність)	Додавання агента = реєстрація у config.mjs + стек	Service Router config + agentMapping

Джерело: складено автором

Отже, запропонована архітектура має низку властивостей, істотних для подальшої реалізації. По-перше, суворе дотримання принципу єдиної

відповідальності дозволяє масштабувати навантаження на окремі агенти незалежно; зокрема, зростання кількості ветеринарних звернень не впливає на латентність страхових консультацій. По-друге, декомпозиція на 14 мікросервісів разом із декларативним описом поведінки агентів робить систему розширюваною: додавання нового доменного агента зводиться до реєстрації його у конфігурації Service Router без модифікації існуючих компонентів. По-третє, застосування blackboard як механізму міжагентної комунікації усуває жорсткі залежності між доменними агентами та спрощує їх незалежне тестування. Водночас алгоритмічна частина, а саме правила класифікації намірів, оцінки терміновості та формування діагностичної гіпотези, на цьому етапі описана лише схематично і потребує окремої формалізації, якій присвячено наступний підрозділ.

2.3. Формалізація алгоритмів прийняття рішень та маршрутизації запитів

У цьому підрозділі зафіксовано формальні описи ключових алгоритмів платформи: класифікація намірів і маршрутизація запитів у Service Router (рис. 2.8), RAG-пошук у ветеринарному агенті (рис. 2.9), оцінка терміновості звернення (рис. 2.10) та оцінка сумарного часу обробки. Математична нотація забезпечує однозначний зв’язок між реалізацією (агенти, інструменти, події) і текстом розділу 3.

Класифікацію намірів у Service Router подано у два кроки: спочатку задається тип відображення «вхід → мітка класу» (рівняння (2.1)), далі формулюється правило максимальної апостеріорної ймовірності (MAP, maximum a posteriori) вибору мітки в наближенні до few-shot-класифікації LLM (рівняння (2.2)).

Рівняння (2.1). Тип відображення класифікатора. Нехай M: множина повідомлень користувача, H: множина історій діалогу (у тому числі структурованих представлень контексту сесії), $C = \{c_1, c_2, c_3, c_4\}$: множина дискретних класів намірів, погоджена з бізнес-правилом R-1 (табл. 2.3).

Класифікатор намірів на першому кроці Service Router (рис. 2.8) трактуємо як відображення з $M \times H$ у C: одна й та сама репліка m при різній історії h може відповідати різним класам, тому формально вхід становить пара (m, h), а не лише m.

$$f: M \times H \to C, \tag{2.1}$$

де f: функція класифікації намірів, яка кожній парі $(m, h) \in M \times H$ ставить у відповідність елемент $c \in C$; M: множина допустимих повідомлень користувача; H: множина допустимих історій діалогу (для порожньої історії $h = \emptyset$ у промпт потрапляє лише m); C: множина класів намірів, $C = \{c_1, c_2, c_3, c_4\}$.

Рівняння (2.2). Правило максимальної апостеріорної ймовірності (MAP) для few-shot-класифікації. Клас наміру c трактують як невідому дискретну величину, тоді як системну інструкцію I, набір few-shot прикладів E, поточне повідомлення m і подання історії h розглядають як умовний контекст спостереження. Правило MAP полягає у виборі класу з найбільшою апостеріорною ймовірністю $P(c \mid m, h, I, E)$; це узгоджується з практичною схемою «обрати мітку з найвищою впевненістю» при нормалізованому виході класифікатора (softmax) або температурі генерації близькій до нуля.

$$\hat{c} = \underset{c \in C}{argmax} P(c \mid m,\ h,\ I,\ E), \tag{2.2}$$

де $\hat{c}$: оцінка класу наміру за MAP-правилом, $\hat{c} \in C$; c: змінна перебору по класах, $c \in C$; C: множина класів намірів, та сама, що в рівнянні (2.1), узгоджена з бізнес-правилом R-1 (табл. 2.3); $P(c \mid m, h, I, E)$: апостеріорна ймовірність класу c за умови спостережуваного контексту (m, h) та зафіксованих I, E (у реалізації на нейромережі відповідає нормалізованим скорингам кандидатів на виході класифікатора); m: поточне повідомлення користувача; h: історія діалогу (якщо $h = \emptyset$, у промпті фактично лишаються m, I та E); I: системна

інструкція (політика формату й стилю відповіді класифікатора); $E = \{(e_1, l_1), \dots, (e_k, l_k)\}$: множина анотованих few-shot прикладів (у системі використовується 2-3 приклади на клас).

При $h \neq \emptyset$ історію нормалізують до канонічного подання (хронологічний список реплік або згорнутий JSON-контекст сесії з ChatHistory) і з'єднують із m в єдиний текстовий блок m', що подається у шаблон промпта; правило MAP при цьому не змінюється.

Рівняння (2.3). Функція відповіді (маршрутизація). Після отримання $\hat{c}$ Service Router (рис. 2.8) визначає подальший сценарій обробки: виклик відповідного Bedrock-агента, відповідь загального привітання або повідомлення про недоступність сервісу (R-2 з табл. 2.3). Клас c_4 зарезервовано для нейтрального привітання; інші класи з C ведуть до доменних агентів за відображенням $R(c)$, якщо сервіс доступний. Умова $D(c) = 0$ формалізує вимкнений агент або порушення інваріантів маршруту.

$$\text{Response}(\text{c}, \text{m}, \text{s}) = \begin{Bmatrix} InvokeAgent(R(c), m, s), c \in C \setminus \{c_4\}, \\ DefaultGreeting(), c = c_4, \\ ServiceUnavailable(c), D(c) = 0 \end{Bmatrix}, \qquad (2.3)$$

де $Response$: функція відповіді (маршрутизації), що задає результат кроку після класифікації для трійки (c, m, s);
c: клас наміру, $c \in C$, узгоджений з R-1 (табл. 2.3), на вході маршрутизатора зазвичай $c = \hat{c}$ з рівняння (2.2);
m: поточне повідомлення користувача, те саме, що в рівнянні (2.2) і в доповненні до нього; s: кортеж параметрів сесії, $s = (sessionId, globalSessionId, policyId)$, що передаються в інструменти та агентські виклики; $InvokeAgent$: абстрактна операція виклику доменного агента з ідентифікатором $R(c)$; R: відображення класу наміру на ідентифікатор Bedrock-агента (маршрут у сенсі R-2); C: множина класів намірів, та сама, що в рівнянні (2.1); c_4: клас «загальне привітання» без

залучення предметного агента; $DefaultGreeting$: операція повернення стандартного привітального повідомлення для $c = c_4$; $ServiceUnavailable$: операція сигналізації про недоступність сервісу для класу c; D: індикатор доступності маршруту для класу c (при $D(c) \neq 0$ маршрут дозволений, при $D(c) = 0$ заборонений або недоступний).

У реалізації алгоритму (рис. 2.8) перевірку $D(c)$ виконують до виклику $InvokeAgent$, щоб не запускати агента при недоступному бекенді.

Рис. 2.8 Блок-схема алгоритму обробки запиту Service Router із класифікацією та маршрутизацією

Рівняння (2.4). Векторне кодування фрагмента корпусу. База знань ветеринарного агента проходить індексацію: корпус документів розбивають на текстові фрагменти фіксованої довжини (chunking у Knowledge Base), утворюючи множину $F = \{f_1, \dots, f_L\}$. Кожен f_j подають на вхід моделі вкладень (embedding model); на виході отримують дійсний вектор фіксованої розмірності, який зберігають у векторному сховищі для наближеного найближчого пошуку (ANN, approximate nearest neighbor) за метрикою з рівняння (2.5).

$$e_j = Encoder(f_j) \in R^d, \quad (2.4)$$

де e_j: вектор вкладення (dense embedding) j-го фрагмента корпусу, $e_j \in R^d$;

$Encoder$: відображення «рядок фрагмента → вектор ознак», реалізоване обраною моделлю вкладень у конвеєрі Bedrock Knowledge Base; f_j: j-й текстовий фрагмент із множини $F = \{f_1, \dots, f_L\}$ після сегментації бази знань;

d: розмірність простору вкладень (гіперпараметр моделі; конкретне значення визначається обраним embedding endpoint і збігається з розмірністю вектора запиту e_q у рівнянні (2.5)).

Рівняння (2.5). Косинусна подібність запиту й фрагмента. Запит користувача q подають на той самий embedding endpoint, що й фрагменти F, і отримують вектор $e_q \in R^d$ тієї самої розмірності d, що й e_j у рівнянні (2.4). Косинусна подібність $sim(e_q, e_j) = cos\theta \in [-1; 1]$ інваріантна до масштабу довжини векторів. При L2-нормалізації ($\| e_q \| = \| e_j \| = 1$) скоринг зводиться до скалярного добутку $e_q \cdot e_j$, що спрощує ANN-пошук. Релевантність визначають так:

$$sim(e_q, e_j) = \frac{e_q \cdot e_j}{\| e_q \| \| e_j \|}, \quad (2.5)$$

де $sim(e_q, e_j)$: косинусна подібність між вектором запиту та вектором j-го фрагмента корпусу, $sim(e_q, e_j) = cos\theta$ для кута θ між e_q та e_j; e_q: вектор вкладення запиту, $e_q \in R^d$; e_j: вектор вкладення j-го фрагмента, той самий, що в рівнянні (2.4); $e_q \cdot e_j$: скалярний добуток у R^d, $e_q \cdot e_j = \sum_{i=1}^{d} e_{q,i}\, e_{j,i}$; $\| e_q \|$, $\| e_j \|$: евклідові (L_2-) норми, $\| e_q \| = \sqrt{\sum_{i=1}^{d} e_{q,i}^2}$, $\| e_j \| = \sqrt{\sum_{i=1}^{d} e_{j,i}^2}$.

За спаданням $sim(e_q, e_j)$ фрагменти ранжують і обирають k найрелевантніших для підстановки в промпт.

Рівняння (2.6). Генерація відповіді LLM з RAG і сесійним контекстом. Відібрані за (2.4)-(2.5) фрагменти разом із запитом q утворюють блок доказів (retrieved evidence), який разом із системною інструкцією I_{vet} (межі компетенції, медичні застереження, формат) та структурованим контекстом сесії ctx з ChatHistory (профіль тварини, дані полісу, симптоми) підставляють у шаблон промпта. У Bedrock Agents це крок пошук → доповнення → генерація. За оглядовими оцінками частка галюцинацій у предметних сценаріях зменшується з 15-25% (без RAG) до 3-5% (з RAG) [4, 22]. Формально:

$$a = LLM(q,\ D_k(q),\ I_{vet},\ ctx), \tag{2.5}$$

де a: згенерований текст відповіді асистента (natural-language completion) за заданого промпта; LLM: абстрактне відображення «доповнений промпт → послідовність токенів», реалізоване обраною foundation-моделлю та параметрами інференсу (температура, top-p тощо); q: текстовий запит користувача (або згорнутий діалог), узгоджений за поданням з рівнянням (2.5) для кодування e_q; $D_k(q)$: множина рівно k фрагментів $\{f_{j_1}, \dots, f_{j_k}\} \subseteq F$, відібраних за найбільшими значеннями $sim(e_q,\ e_j)$ з рівняння (2.5) після ранжування; k: гіперпараметр retriever (розмір top-k у Knowledge Base), $k \geq 1$;
I_{vet}: системний промпт ветеринарного агента (політика відповіді та безпеки);
ctx: структурований контекст сесії (записи ChatHistory, поліс, маршрутизація тощо), що передається в промпт поруч із q та $D_k(q)$.

Рис. 2.9 Схема RAG-конвеєра ветеринарного агента: від запиту через семантичний пошук до доповненої генерації

Рівняння (2.7). Оцінка терміновості ветеринарного звернення. Після (2.6) ветеринарний агент узгоджує текст відповіді зі структурованою оцінкою ризику (urgency_level), яку використовує Booking Manager для пріоритизації клініки та рекомендації візиту. Нехай S: множина ідентифікованих із діалогу симптомів, P: профіль тварини, MH: медична історія з ChatHistory. Рівні u_1-u_4 відповідають міткам інструмента RecommendDoctorVisit: emergency, urgent, normal, routine. Алгоритм консервативний: наявність будь-якого симптому з переліку $E_{critical}$ одразу призначає u_1 (узгоджено з R-5). Інші гілки утворюють послідовне дерево рішень (рис. 2.10): умови читають згори донизу, застосовують першу істинну.

$$Urgency(S,P,MH) = \begin{cases} u_1\ (emergency), \exists s \in S : s \in E_critical, \\ u_2\ (urgent), (Duration(S) > 24\,h) \vee HighRiskProfile(P), \\ u_3\ (normal), HasActiveSymptoms(S), \\ u_4\ (routine), in\ all\ other\ cases. \end{cases}, \quad (2.6)$$

де $Urgency$: функція оцінки рівня терміновості, $Urgency(S,P,MH) \in \{u_1, u_2, u_3, u_4\}$; S: множина симптомів і скарг, витягнута з поточного повідомлення та, за наявності даних, з MH і

структурованого контексту сесії; P: профіль тварини (ознаки, що входять до правил $HighRiskProfile$); MH: медична історія сесії, що дозволяє уточнити тривалість перебігу та фоновий ризик при інтерпретації S і P; u_1: найвищий рівень терміновості (emergency), невідкладна допомога; $E_{critical}$: множина критичних симптомів-індикаторів, $E_{critical}$ = {утруднене дихання, судоми, сильна кровотеча, підозра на отруєння, ціаноз}; u_2: рівень urgent (негайний або найближчий огляд без зволікання, якщо прямої загрози життю з переліку $E_{critical}$ не виявлено, але є пролонгований перебіг або високий профільний ризик);

$Duration$: відображення «множина симптомів → оцінена тривалість перебігу» у годинах (поріг $24\ h$ у формулі), узгоджене з часовими мітками в MH та текстом діалогу; $HighRiskProfile$: предикат «підвищений ризик» за віком, масою, хронічними станами та іншими атрибутами P, зафіксованими в політиці агента; u_3: рівень normal (потрібна консультація або візит у плановому порядку при наявності активних, але некритичних симптомів);

$HasActiveSymptoms$: предикат наявності активних (поточних) симптомів у S після перевірок вищих гілок; u_4: рівень routine (плановий або профілактичний сценарій); останній рядок кускового запису (in all other cases) описує решту випадків, коли жодна з попередніх умов не виконується.

Рис. 2.10 Дерево прийняття рішень алгоритму оцінки терміновості ветеринарних звернень

Рівняння (2.8). Час обробки одного обміну «запит-відповідь». Для типового наскрізного сценарію «повідомлення → Service Router → доменний Bedrock-агент → відповідь і події» стінний час (wall-clock latency) представлено як адитивну декомпозицію вздовж критичного шляху для одного обміну: підготовка контексту, класифікація наміру, маршрутизація, виконання агента (RAG і Lambda-інструменти), публікація подій SSE/SNS. Ця оцінка слугує верхньою межею для NFR з підрозділу 1.4 (≤ 10 с на відповідь для типових запитів і ≤ 15 с для мультидисциплінарних). Повний цикл (3-6 обмінів) дає орієнтовний час сесії 1-2 хв для типових і 2-3 хв для мультидисциплінарних запитів. Формально:

$$T_{total} = T_{init} + T_{classify} + T_{route} + T_{agent} + T_{events}, \qquad (2.8)$$

де T_{total}: сумарний час обробки одного звернення вздовж узгодженого ланцюжка, с; T_{init}: фаза ініціалізації (завантаження/узгодження контексту сесії, збереження повідомлення

тощо), орієнтовно $20 - 50$ мс; $T_{classify}$: час інференсу LLM-класифікатора намірів у Service Router (рівняння (2.2)), орієнтовно $0{,}5 - 2$ с; T_{route}: час маршрутизації після отримання класу $\hat{c}$ (рівняння (2.3)), зокрема перевірка $D(c)$ і підготовка виклику агента, орієнтовно $1 - 5$ мс; T_{agent}: час виконання обраного доменного агента (основний внесок: Bedrock Agents, можливі кроки retrieve-generate за (2.4)-(2.6) та Lambda tool use), орієнтовно $3 - 15$ с; T_{events}: час публікації та доставки подій клієнту (SSE/SNS тощо), орієнтовно $100 - 300$ мс.

Орієнтири компонентів узгоджуються з межею ~15 с для однієї відповіді; фактичні виміри в підрозділі 3.3 показують p95 = 9,34 с при піковому навантаженні, що відповідає NFR-01 (p95 < 10 с). При 3-5 обмінах час сесії становить 1-2 хв, тобто на порядок менше за 15-25 хв у моделі AS-IS.

Отже, у підрозділі формалізовано вісім ключових алгоритмів системи: класифікацію намірів (2.2), маршрутизацію запитів до спеціалізованих агентів (2.3), семантичний пошук RAG (2.4)-(2.6), оцінку терміновості ветеринарних звернень за чотирирівневою шкалою (2.7) та адитивну модель часу обробки обміну (2.8). Запропоновані алгоритми узгоджено з бізнес-правилами R-1 - R-5 та нефункціональними вимогами NFR-01, NFR-04, що забезпечує формальне підґрунтя для подальшої програмної реалізації. Подальший вибір технологічного стека, який дозволяє ефективно втілити ці алгоритми у хмарному середовищі, обґрунтовується у наступному підрозділі.

2.4. Обґрунтування вибору технологічного стека (LLM-моделі, фреймворки оркестрації, хмарні бази даних)

Вибір технологічного стека здійснювався за ієрархічним принципом: спочатку обрано хмарну екосистему як стратегічне рішення, а потім обґрунтовано вибір конкретних сервісів у межах цієї екосистеми.

Порівняння трьох основних хмарних платформ (AWS, Microsoft Azure та Google Cloud Platform) за критеріями, релевантними для мультиагентної LLM-системи, виявило переваги AWS за сукупністю факторів. Ключовим аргументом є наявність у AWS керованого сервісу оркестрації агентів (Bedrock Agents) із нативною підтримкою tool use, Knowledge Bases для RAG та декларативним визначенням через CloudFormation [13]; на момент проєктування Azure AI Agent Service перебував на етапі обмеженого попереднього доступу, а Google Vertex AI Agent Builder не підтримував декларативне визначення через інфраструктуру як код (IaC). Модель pay-per-use в AWS Lambda та DynamoDB On-Demand додатково забезпечує масштабування від нуля (NFR-05), що критично для стартового етапу з непередбачуваним навантаженням. Нарешті, лідерська позиція AWS на ринку хмарних послуг гарантує найширшу екосистему документації, спільноти та інтеграцій для обраного стека (Bedrock, Lambda, DynamoDB).

Після обрання AWS як базової екосистеми подальший вибір конкретних сервісів (Bedrock Agents, Lambda, DynamoDB) обумовлений перевагами нативної інтеграції: спільна IAM-модель безпеки, єдиний IaC-інструмент (SAM/CloudFormation), мінімальна мережева латентність між сервісами та відсутність потреби в адаптерах. Нижче наведено порівняльний аналіз альтернатив для кожного рівня стека, починаючи з вибору мовної моделі (табл. 2.10).

Табл. 2.10

Порівняльний аналіз великих мовних моделей (LLM) для системи «ВетЕксперт»

Критерій	**Claude 3.5 Haiku**	**GPT-4o mini**	**Gemini 1.5 Flash**	**Llama 3.1 8B**
Точність у предметних задачах (MMLU, 5-shot) [20]	73-77%	81-82%	78-79%	68-73%
Контекстне вікно (токени)	200 000	128 000	1 000 000	128 000
IFEval (instruction following) [29]	88,7%	85,3%	83,1%	77,2%
Ціна за 1M вхідних / вихідних токенів (USD)	1,00 / 5,00	0,15 / 0,60	0,075 / 0,30	0,22 / 0,22
Доступність через Amazon Bedrock	Так	Ні	Ні	Так (часткова)
Зниження галюцинацій із RAG [4, 22]	До 3-5%	До 5-8%	До 5-7%	До 8-12%

Джерело: складено автором на основі [4, 24, 20, 29]

На момент ухвалення проєктного рішення (2025 р.) обрано Anthropic Claude 3.5 Haiku через збалансоване співвідношення показників: найкращий результат instruction following серед розглянутих моделей (IFEval 88,7%), контекстне вікно 200 тис. токенів, доступність через Amazon Bedrock із нативною підтримкою tool use та нижчий рівень галюцинацій у зв’язці з RAG (до 3-5%) [13, 20, 29]. Відставання за MMLU від GPT-4o mini та Gemini 1.5 Flash компенсується саме нативною інтеграцією з Bedrock Agents, що є критичним для обраної екосистеми: альтернативні моделі довелося б викликати через зовнішні API з додатковою мережевою латентністю, окремою IAM-моделлю та відсутністю декларативного опису через CloudFormation (порівняльні

значення див. у табл. 2.10). Далі порівняно платформи оркестрації (табл. 2.11).

Табл. 2.11

Порівняльний аналіз платформ оркестрації мультиагентних систем

Критерій	**Bedrock Agents**	**LangChain / LangGraph**	**AutoGen**	**Власна реалізація**
Тип рішення	Керований сервіс	Бібліотека (OSS)	Фреймворк (OSS)	Від нуля
Управління інфраструктурою	Не потрібне	Самостійне	Самостійне	Самостійне
Масштабування від нуля (NFR-05)	Так	Ні	Ні	Так (через Lambda)
Модель оплати (NFR-11)	Pay-per-use	Вартість хостингу	Вартість хостингу	Pay-per-use
Інтеграція з Lambda (tool use)	Нативна	Через adapter	Через adapter	Ручна
Підтримка Knowledge Bases (RAG)	Вбудована	Через інтеграції	Обмежена	Ручна

Джерело: складено автором на основі [12, 17, 21]

Обрано Amazon Bedrock Agents через нативну інтеграцію з AWS Lambda, DynamoDB та CloudFormation; за оцінкою Forrester для Amazon Bedrock зазначено до ~70% скорочення часу виведення рішень на ринок порівняно з самостійною збіркою стека [21]. Порівняння хмарних баз даних за критеріями латентності, масштабування та вартості подано у табл. 2.12.

Табл. 2.12

Порівняльний аналіз хмарних баз даних для системи «ВетЕксперт»

Критерій	**DynamoDB**	**Aurora Serverless v2**	**MongoDB Atlas**	**ElastiCache**
Модель даних	Документна / ключ-значення	Реляційна + JSONB	Документна	Ключ-значення
Латентність (P99)	1-5 мс	3-50 мс	2-8 мс	< 1 мс
Масштабування від нуля	Так (on-demand)	Ні (мін. 0,5 ACU)	Так	Ні (мін. 1 вузол)
Вартість за 200K ops/міс (USD)	~0,50	~43,00	~2,00	~15,00
Інтеграція з AWS Lambda	Нативна (SDK v3)	Через RDS Proxy	Через Atlas API	Через VPC endpoint
Гарантія доступності SLA	99,999%	99,99%	99,995%	99,99%

Джерело: складено автором на основі [30]

Слід зазначити, що DynamoDB має суттєві обмеження: відсутність JOIN-запитів, eventual consistency за замовчуванням, максимальний розмір елемента 400 КБ, обмежена кількість вторинних індексів та складність моделювання зв’язків типу N:M. Проте для предметної області «ВетЕксперт» ці обмеження є прийнятними: access patterns системи зводяться до key-value lookup за sessionId або policyId без складних реляційних запитів; розмір даних сесії (історія діалогу та контекст) не перевищує кількох кілобайт, що значно менше за обмеження 400 КБ; зв’язки між сутностями реалізовані на рівні додатку через посилання (policyId у сесії), а не через JOIN-операції; для критичних читань (контекст сесії) використовується strongly consistent read.

Обрано Amazon DynamoDB (on-demand) через найнижчу вартість (~0,50 USD) і масштабування від нуля.

Серверна логіка: Node.js 20.x (ESM) забезпечує cold start середовища виконання від 80 до 200 мс (повний cold start із завантаженням AWS SDK-клієнтів може сягати від 1,5 до 1,8 с; оптимізація через Singleton-патерн розглянута у підрозділі 3.3) і єдину мову з фронтендом [32]. Клієнтський інтерфейс: Nuxt 4 (Vue 3) з Tailwind CSS та Pinia дає базовий бандл фреймворку ~45 КБ (gzipped, без прикладних компонентів) і нативну підтримку SSE [33]. Підсистема подій: SNS + DynamoDB + Lambda Response Streaming (SSE). IaC: AWS SAM [34]. Монорепозиторій: pnpm workspaces та esbuild. Зведений стек із обґрунтуванням вибору кожного рівня подано у табл. 2.13, а взаємозв'язки між компонентами подано на рис. 2.11.

Табл. 2.13

Зведений технологічний стек системи «ВетЕксперт» з обґрунтуванням вибору

Рівень	**Технологія**	**Альтернативи**	**Ключова перевага**
Мовна модель	Claude 3.5 Haiku	GPT-4o mini, Gemini 1.5 Flash, Llama 3.1 8B	Точність 88-92%, нативна інтеграція з Bedrock
Оркестрація	Amazon Bedrock Agents	LangChain, AutoGen, Custom	Керований сервіс, IaC, масштабування від нуля
База даних	DynamoDB (on-demand)	Aurora Serverless, MongoDB Atlas, ElastiCache	Pay-per-request (~0,50 USD/міс), 1-5 мс латентність
Серверна логіка	Lambda (Node.js 20.x, ESM)	Python 3.12	Cold start runtime від 80 до 200 мс, єдина мова з фронтендом
Фронтенд	Nuxt 4 + Tailwind + Pinia	Next.js, Angular	Базовий бандл ~45 КБ (gzipped), Composition API

Рівень	Технологія	Альтернативи	Ключова перевага
Доставка подій	SNS + DynamoDB + SSE	WebSocket, EventBridge + AppSync	Простота, персистентність, нативний EventSource
IaC	AWS SAM	CDK, Terraform	Спеціалізація на serverless, локальне тестування
Монорепозиторій	pnpm workspaces + esbuild	npm, Yarn, webpack	Жорсткі посилання, бандл < 50 КБ

Джерело: складено автором

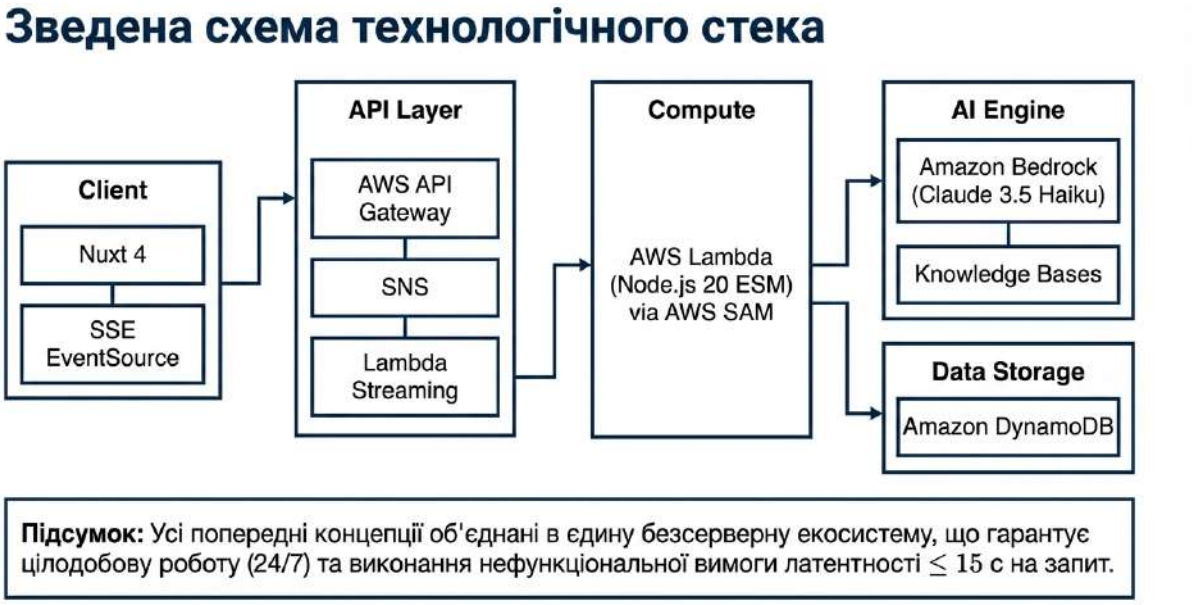


Рис. 2.11 Зведена схема технологічного стека системи «ВетЕксперт» із зазначенням взаємозв’язків між компонентами

Рівняння (2.9). Верифікація вартості експлуатації (NFR-11). Нефункціональна вимога NFR-11 задає орієнтир не більше 100 USD на місяць операційних витрат хмарної інфраструктури при $N_{sess} = 10\,000$ повних діалогових сесій на місяць (наскрізний сценарій «клієнт → Service Router → Bedrock Agents → інструменти Lambda → DynamoDB → події»).

Для прозорого контролю виконання орієнтиру місячну сукупну вартість C_{total} представляють як суму домінуючих складових pay-per-use-моделі AWS: витрати на інференс LLM у Bedrock (C_{LLM}), DynamoDB (C_{DB}), виконання Lambda (C_{λ}), API Gateway (C_{API}), SNS (C_{SNS}) та об'єктне сховище S3 (C_{S3}).

У тому самому записі (2.9) структурне рівняння для C_{total} поєднують із оцінкою домінуючої складової C_{LLM} за токенами; для відтворюваності останньої прийнято такі припущення щодо навантаження: середня довжина сесії становить 5 повідомлень користувача; на одну сесію в середньому припадає 3 виклики Bedrock Agent (класифікатор намірів та один або два доменні агенти); на один виклик орієнтовно 800 вхідних токенів (системна інструкція ~ 300 + контекст ~ 300 + поточне повідомлення ~ 200) та 350 вихідних токенів. Тоді на сесію виходить $t_{in}^{sess} \approx 3 \times 800$, тобто від 2 400 до 2 500 токенів, та $t_{out}^{sess} \approx 3 \times 350 = 1\,050$ токенів; при $N_{sess} = 10\,000$ це дає місячні обсяги порядку 25 млн вхідних та 10,5 млн вихідних токенів, тобто $T_{in}^{M} \approx 25$ та $T_{out}^{M} \approx 10{,}5$ у мільйонах токенів. Вартість C_{LLM} лінійно залежить від цих обсягів і від тарифів постачальника за один мільйон токенів кожного типу (для обраної моделі див. рядок «Ціна за 1M вхідних / вихідних токенів» у табл. 2.10). Зв'язок «тарифи → місячна вартість» і зведення до C_{total} записують так:

$$C_{total} = C_{LLM} + C_{DB} + C_{\lambda} + C_{API} + C_{SNS} + C_{S3}, C_{LLM} =$$
$$= p_{in}\, T_{in}^{M} + p_{out}\, T_{out}^{M} = 1{,}00 \cdot 25 + 5{,}00 \cdot 10{,}5 = 77{,}50\ \frac{USD}{mo}. \qquad (2.9)$$

де C_{total}: сукупна оцінка операційних витрат за календарний місяць у розглянутому сценарії, USD/міс;
C_{LLM}: вартість спожитих токенів foundation-моделі Claude 3.5 Haiku в Amazon Bedrock за місяць (другий рядок (2.9));
C_{DB}: вартість DynamoDB on-demand (авторська оцінка $\approx 0{,}50$ USD/міс при наведеному обсязі операцій читання/запису);

C_{λ}: сукупна вартість викликів AWS Lambda для мікросервісів і action groups, USD/міс;

C_{API}: вартість запитів через API Gateway, USD/міс;

C_{SNS}: вартість публікації подій у SNS, USD/міс;

C_{S3}: вартість зберігання та вибірок з S3 (у тому числі артефакти Knowledge Base / статичний фронтенд за сценарієм навантаження), USD/міс;

p_{in}: ціна за 1 млн вхідних токенів, USD/1M in (для Claude 3.5 Haiku в табл. 2.10: $p_{in} = 1{,}00$); p_{out}: ціна за 1 млн вихідних токенів, USD/1M out ($p_{out} = 5{,}00$);

T_{in}^{M}: сумарний обсяг вхідних токенів за місяць, млн in;

T_{out}^{M}: сумарний обсяг вихідних токенів за місяць, млн out.

За отриманим значенням C_{LLM} з (2.9) та орієнтовними оцінками інших складових з першого рядка (2.9), NFR-11 для базового сценарію 10 000 сесій/міс виконується з запасом; домінуючою складовою залишається C_{LLM}, що типово для мультиагентних LLM-систем із частими викликами інференсу.

Таким чином, C_{LLM} зводиться до перевіряємого добутку «обсяг у мільйонах токенів × ставка за мільйон», що зручно масштабувати при зміні N_{sess} або профілю діалогу (кількість викликів агента на сесію, довжина контексту RAG тощо) без зміни структури рівняння (2.9).

Отже, обрано єдину хмарну екосистему AWS із моделлю pay-per-use, що забезпечує нативну інтеграцію, масштабування від нуля та виконання орієнтиру NFR-11 у базовому сценарії, верифікованому записом (2.9).

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПЛАТФОРМИ «ВЕТЕКСПЕРТ»

3.1. Особливості програмної реалізації серверної логіки та механізму взаємодії агентів

У попередньому розділі було формалізовано бізнес-процеси, спроєктовано мультиагентну архітектуру та обґрунтовано вибір технологічного стеку на базі хмарних сервісів AWS. Цей розділ присвячено безпосередній програмній реалізації запроєктованих рішень: від організації кодової бази та механізмів взаємодії агентів до розгортання інфраструктури та верифікації ефективності системи.

Реалізація серверної частини платформи передбачає координацію 14 незалежних мікросервісів, кожен з яких розгортається як окрема Lambda-функція. За таких умов критичного значення набуває організація кодової бази: необхідно забезпечити повторне використання спільних утиліт, незалежне версіонування компонентів та зручність локальної розробки. Для вирішення цих завдань обрано архітектуру монорепозиторію з менеджером пакетів pnpm, який забезпечує ефективне управління залежностями через механізм символьних посилань та єдине сховище модулів. Структуру каталогів монорепозиторію зведено у табл. 3.1.

Табл. 3.1

Структура каталогів монорепозиторію

Каталог	**Призначення**	**Вміст**
packages/agent-tools/	Спільна бібліотека	Хелпери для DynamoDB, публікації подій, форматування відповідей Bedrock
apps/frontend/	Клієнтський додаток	Nuxt 4, Vue 3, Pinia, Tailwind CSS
apps/services/	Серверні мікросервіси	14 незалежних Lambda-функцій із власними template.yaml
data/	Міграції та насіння БД	Скрипти створення таблиць DynamoDB, тестові дані

Джерело: складено автором

Кожен мікросервіс має уніфіковану структуру: app.mjs виконує роль обробника Lambda, template.yaml є шаблоном AWS SAM, package.json містить залежності. Агенти додатково містять instruction.txt з інструкцією поведінки. Загальну структуру каталогів та залежностей монорепозиторію візуалізовано на рис. 3.1.

Рис. 3.1 Структура каталогів та залежностей монорепозиторію системи «ВетЕксперт»

Service Router реалізує алгоритм обробки як Lambda-функція на Node.js 20.x. Модуль config.mjs реалізує агент-маппінг

Ідентифікатори інжектуються через змінні середовища, тому додавання нового агента потребує лише реєстрації у agentMapping (NFR-09).

Обробник lambdaHandler реалізує семикроковий алгоритм: прийом запиту, ініціалізація контексту, збереження повідомлення, класифікація наміру, маршрутизація, виклик агента Bedrock із sessionState та збереження відповіді.

Ключовим механізмом є передача sessionState при виклику Bedrock Agent: атрибути globalSessionId та policyId автоматично передаються кожному інструменту. Функція invokeIntentionClassifier() при наявності історії формує структурований контекст для контекстно-залежної класифікації. Спільні утиліти, що забезпечують персистентність, комунікацію та протокол Bedrock, винесено до бібліотеки @vetexpert/agent-tools (табл. 3.2).

Табл. 3.2

Компоненти спільної бібліотеки @vetexpert/agent-tools

Модуль	**Категорія**	**Ключові функції**	**Споживачі**
chat-history.mjs	Персистентність	saveChatMessage(), initializeSessionContext(), updateSessionContext(), getChatHistory(), getSessionContext()	service-router, tool-context-save, tool-context-details
event-publisher.mjs	Комунікація	sendEventToPublisher()	tool-policy-details, tool-recommend-doctor-visit, tool-book-vet-clinic
response.mjs	Протокол Bedrock	createAgentResponse(), extractSessionId(), extractParameters()	Усі інструменти агентів (6 Lambda-функцій)

Джерело: складено автором

Метод initializeSessionContext() застосовує ConditionExpression для захисту від гонок при паралельних Lambda-запусках: якщо запис-сентинель із timestamp = 0 уже існує, повертається наявний контекст без перезапису. Публікація подій через event-publisher.mjs виконується неблокуюче, щоб не збільшувати затримку відповіді користувачу.

Інструменти агентів (ReAct-цикл, підрозд. 2.2) реалізовано як окремі Lambda-функції: tool-context-save накопичує множинні значення одного ключа у масиві contextData[contextKey], а tool-recommend-doctor-visit поєднує синхронну відповідь Bedrock із асинхронною SSE-подією.

Event Publisher зберігає події у DynamoDB, звідки SSE Stream доставляє їх клієнту через Lambda Response Streaming [13] із автоматичним перепідключенням на стороні EventSource API. Архітектуру цього механізму подано на рис. 3.2.

Рис. 3.2 Архітектура потокової доставки подій через SSE (опитування DynamoDB кожні 2 с, keep-alive 30 с, максимальна тривалість з'єднання 5 хв)

ШІ-агенти визначено через AWS SAM як AWS::Bedrock::Agent із IAM-роллю та моделлю Claude 3.5 Haiku; поведінку задано інструкціями з instruction.txt (few-shot-промптинг для класифікатора, багатоетапний робочий процес для ветеринарного агента). Як видно з табл. 3.3, частину принципів із підрозд. 2.2 реалізовано штатними засобами AWS SAM, тоді як blackboard-патерн і стійкість до відмов потребували власних рішень на рівні коду.

Табл. 3.3

Відповідність програмних рішень архітектурним принципам і проєктним патернам

Принцип / патерн	**Програмна реалізація**	**Ключові модулі**
Єдина відповідальність мікросервісів	Кожен мікросервіс реалізовано як окремий app.mjs із власним template.yaml	14 каталогів у apps/services/
Незалежне масштабування	Кожна Lambda-функція розгортається як окремий стек CloudFormation	14 файлів template.yaml
Декларативна поведінка агентів	Інструкції у instruction.txt, модель та параметри задані у template.yaml	instruction.txt + AWS::Bedrock::Agent
Слабка зв’язаність	Маппінг агентів через config.mjs, ідентифікатори через env vars	config.mjs + template.yaml Parameters
Подієво-орієнтовані побічні ефекти	sendEventToPublisher() у інструментах, Event Publisher → DynamoDB → SSE	event-publisher.mjs + service-sse-stream
Спільна дошка (blackboard)	ChatHistory[timestamp=0], SaveContext/GetContextDetails	chat-history.mjs + tool-context-save/details
Стійкість до відмов	try-catch у некритичних операціях, fallback-відповіді	app.mjs (service-router)
Singleton ресурси	DynamoDBClient та BedrockClient на рівні модуля	chat-history.mjs + bedrockClient.mjs

Джерело: складено автором

Таким чином, усі 14 мікросервісів реалізовано на єдиному стеку (Node.js 20.x, AWS SAM, pnpm-монорепозиторій) із повторним використанням логіки через бібліотеку @vetexpert/agent-tools, що забезпечує виконання нефункціональних вимог щодо масштабованості, розширюваності та стійкості до відмов. Побудована серверна частина

створює передумови для розробки клієнтського інтерфейсу та його інтеграції з API платформи, що розглядається у наступному підрозділі.

3.2. Розробка клієнтського інтерфейсу та інтеграція системи із зовнішніми API

Клієнтський додаток реалізовано як односторінковий застосунок на фреймворку Nuxt 4 [33] у режимі ssr: false. Відмова від серверного рендерингу обумовлена специфікою платформи: основний сценарій передбачає постійне SSE-з'єднання та тривалу діалогову сесію, для яких рендеринг на сервері не дає приросту продуктивності, зате ускладнює інфраструктуру. Файлова маршрутизація Nuxt дозволила скоротити обсяг конфігураційного коду, адже кожен файл у каталозі pages/ автоматично стає окремим маршрутом.

Кодову базу організовано за принципом розділення відповідальностей. Каталог components/ містить 19 Vue-компонентів рівня представлення, каталог composables/ містить 8 функцій з бізнес-логікою інтерфейсу, stores/ містить 3 Pinia-сховища глобального стану, pages/ містить 3 сторінки (головна, список прийомів, тестова). Модуль utils/ об'єднує допоміжні утиліти (HTTP-клієнт, сервіс чату, маппінг агентів), а каталог types/ містить TypeScript-інтерфейси для типізації повідомлень і SSE-подій, що дозволяє відловлювати розбіжності зі схемою серверного API ще на етапі компіляції. Компоненти згруповано за функціональним призначенням (табл. 3.4).

Табл. 3.4

Компонентні групи клієнтського інтерфейсу

Група	**Компоненти**	**Призначення**
chat/	ChatContainer, ChatHeader, ChatMessages, ChatMessage, ChatComposer, ChatTypingIndicator, AssistantAvatar	Чат-інтерфейс взаємодії з ШІ-агентами
chat/message-types/	TextMessage, ConfirmationMessage	Поліморфні типи повідомлень
pet/	PetProfile, PetAvatar, PetInfo, PetVaccinations, PetUpcoming, PetPolicyCards, PetQuickActions	Профіль тварини та страхового полісу
layout/	AppHeader, AppFooter	Структурні елементи макету
modals/	PetProfileModal	Модальне вікно редагування профілю

Джерело: складено автором

Фрагменти реалізованого клієнтського застосунку з діалоговим вікном, адміністративним переглядом портфеля та керуванням записами на візити подано на рис. 3.3-3.5; ті самі екрани наведено в додатку А з детальним покомпонентним описом.

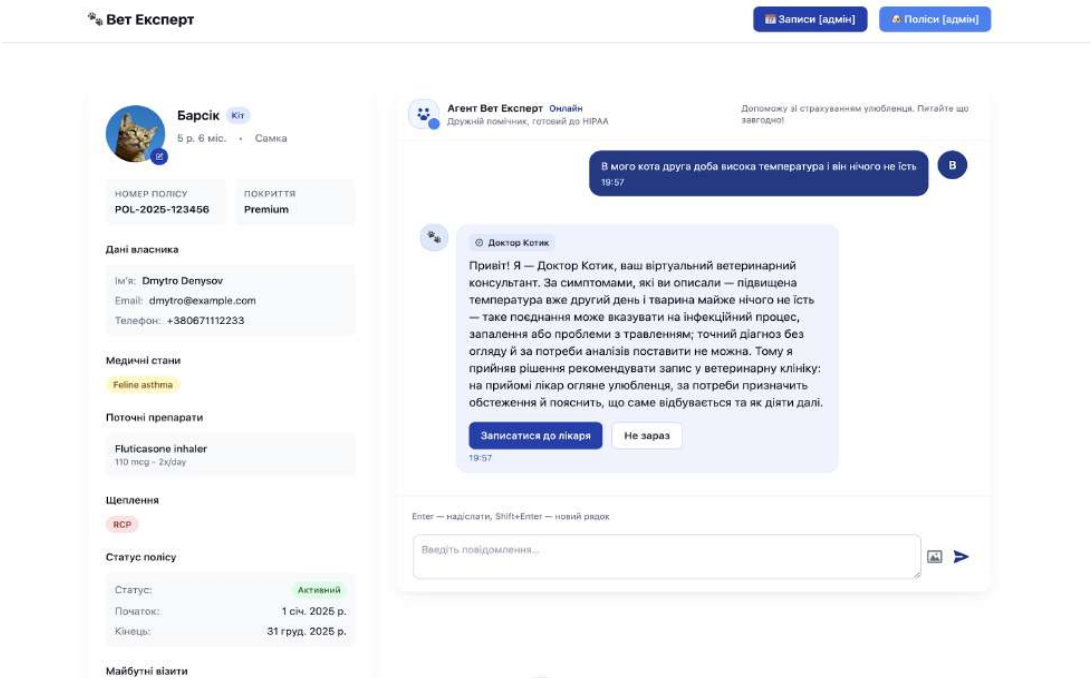


Рис. 3.3 Головний екран платформи «ВетЕксперт»: діалог із мультиагентною системою та контекстна картка улюбленця

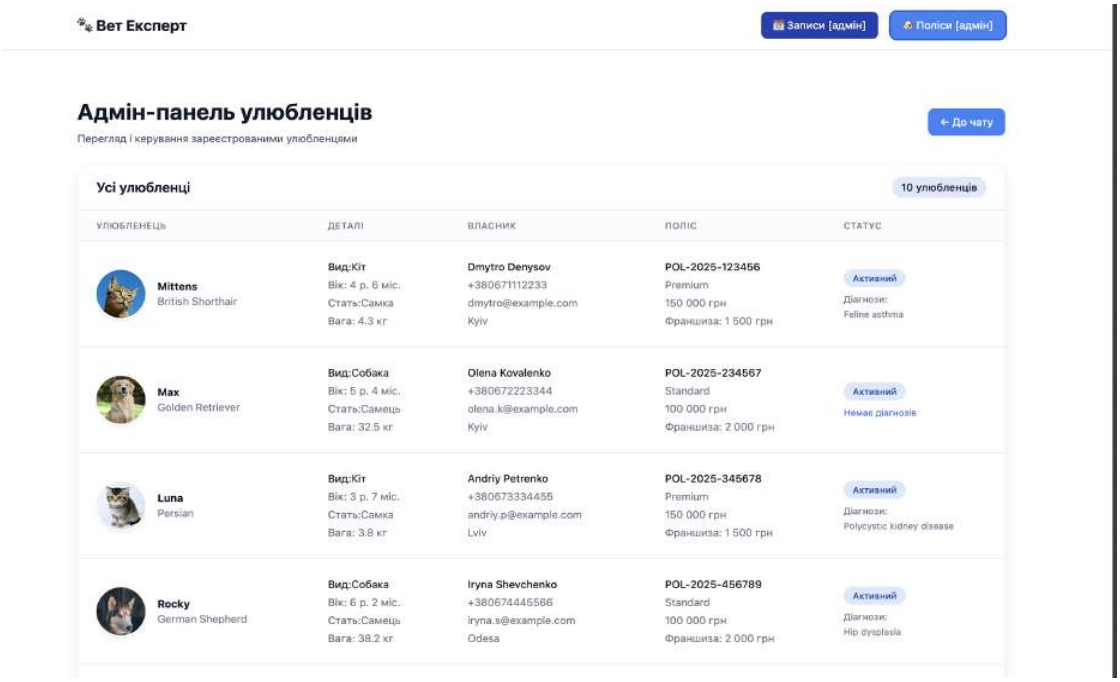


Рис. 3.4 Адміністративна панель управління улюбленцями та страховими полісами

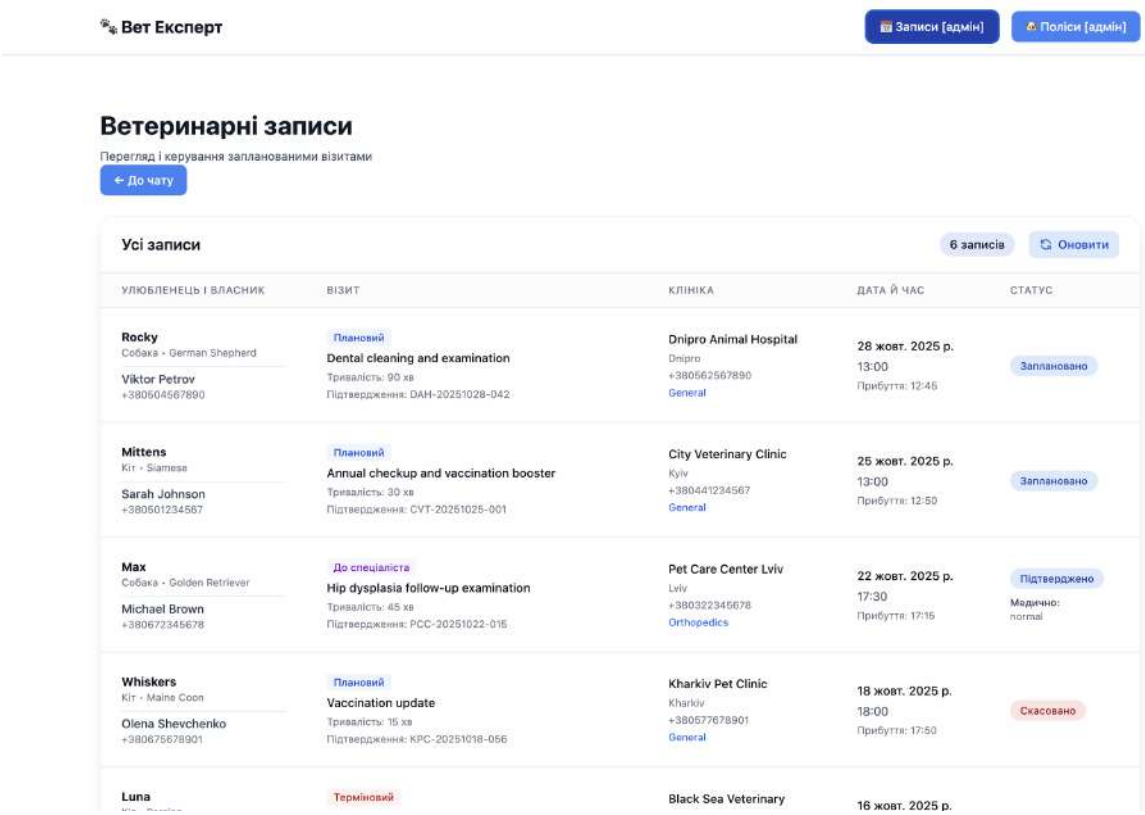


Рис. 3.5 Панель управління ветеринарними записами, створеними мультиагентною системою

Серед перелічених компонентів два визначають гнучкість інтерфейсу. ChatMessage використовує вбудований у Vue механізм динамічного рендерингу <component :is>: залежно від поля type у повідомленні підставляється або TextMessage, який рендерить Markdown-відповідь агента, або ConfirmationMessage, який виводить інтерактивні кнопки підтвердження («Так» / «Ні»). Такий підхід дозволяє розширювати набір типів повідомлень без модифікації батьківського компонента: для цього достатньо додати новий клас у chat/message-types/ та зареєструвати його у маппінгу.

Компонент PetProfile побудовано за патерном прогресивного розкриття: за замовчуванням відображаються лише найкритичніші дані (кличка тварини, номер полісу, найближча подія), а деталізовані блоки (історія вакцинацій, перелік покриттів полісу, швидкі дії) розкриваються за кліком на відповідну картку. Це знижує когнітивне навантаження при

першому завантаженні сторінки й одночасно зберігає доступ до повної інформації в один клац. Результуючу компонентну ієрархію та макет головної сторінки відображено на рис. 3.6.

Рис. 3.6 Компонентна ієрархія та макет головної сторінки платформи «ВетЕксперт»

Для керування глобальним станом обрано Pinia, офіційно рекомендоване для Vue 3 сховище, яке у порівнянні з Vuex потребує меншого обсягу шаблонного коду та природно інтегрується з Composition API. У проєкті використано три сховища: eventQueue (буферизація вхідних SSE-подій), petProfile (дані тварини та полісу) та chat (історія діалогу та стан набору повідомлення). Особливої уваги заслуговує eventQueue: під час активного очікування відповіді агента (прапорець isAgentThinking === true) нові події не застосовуються одразу, а зберігаються у черзі й обробляються після завершення відповіді. Без такої буферизації оновлення профілю або поява інтерактивної кнопки могли б перервати читання поточного повідомлення й дезорієнтувати користувача.

Бізнес-логіка клієнта винесена у вісім composables. Ключова з них, useChat, інкапсулює повний цикл взаємодії з агентом: формування історії повідомлень chatHistory, виклик sendChatMessage(), збереження bedrockSessionId для підтримки контексту між зверненнями та постобробку відповіді (парсинг Markdown у HTML). Мережевий шар зосереджено в chatService.ts, типізованому HTTP-клієнті з таймаутом 30 с, що відповідає верхній межі тривалості відповіді агента за результатами навантажувального тестування (див. підрозділ 3.3). Повний цикл обробки запиту від інтерфейсу до ШІ-агента і назад ілюструє діаграма послідовності на рис. 3.7.

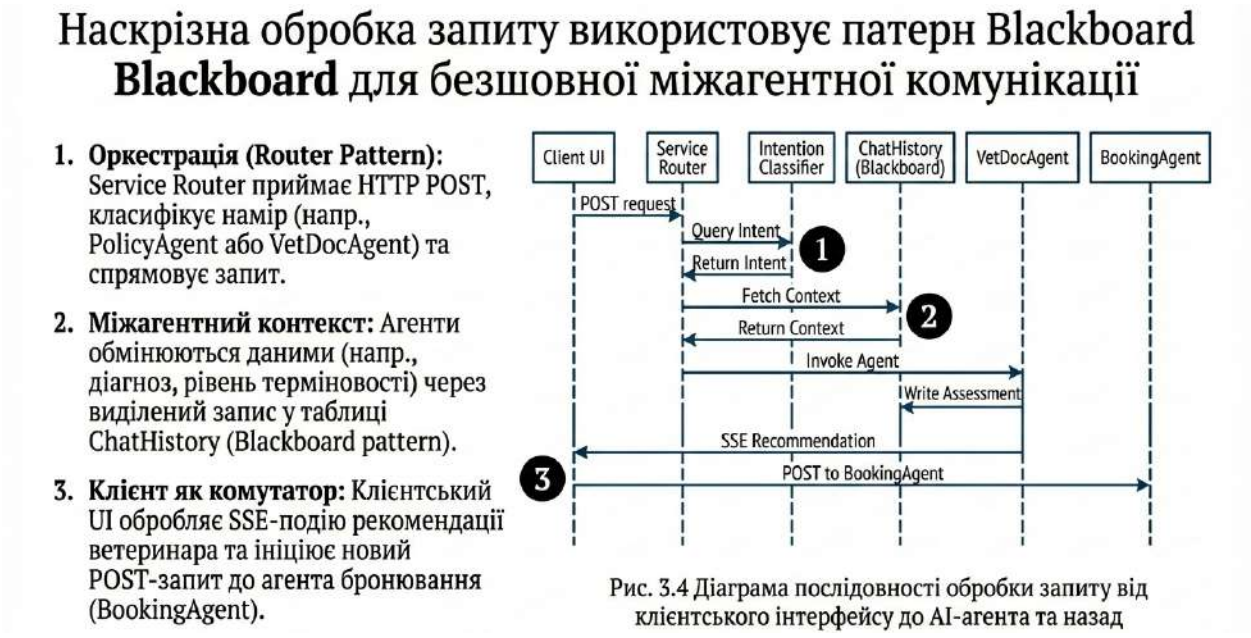


Рис. 3.7 Діаграма послідовності обробки запиту від клієнтського інтерфейсу до ШІ-агента та назад

Головна сторінка встановлює SSE-з’єднання через вбудований у браузер EventSource API. SSE обрано замість WebSocket свідомо: потік даних між сервером і клієнтом є односпрямованим, тож події породжуються виключно серверними агентами, тоді як повідомлення користувача надходять окремим HTTP-каналом. За такого сценарію

протокол SSE є достатнім і водночас простішим у експлуатації: він базується на звичайному HTTP, підтримує автоматичне перепідключення та прозоро проходить через корпоративні проксі. Сторінка обробляє п’ять типів подій: PolicyDetailsRetrieved (оновлення профілю полісу), RecommendDoctorVisit (рекомендація візиту), doctor_visit:confirmed / doctor_visit:declined (реакція користувача) та AppointmentBooked (підтвердження запису).

Окремої уваги заслуговує обробка події RecommendDoctorVisit: вона трансформується у ConfirmationMessage, натискання кнопки «Так» у якому викликає sendMessage() з текстом, адресованим вже іншому агенту, менеджеру бронювання. У такий спосіб клієнтський інтерфейс виступає посередником міжагентної координації: ветеринарний агент і агент бронювання не викликають один одного напряму, а взаємодіють лише через підтверджуючу дію користувача. Це узгоджується з принципом human-in-the-loop для медично значущих рішень: жоден крок, що передбачає зміну планів користувача (запис до лікаря, виклик спеціаліста), не виконується без його явної згоди. Повний перелік інтеграцій клієнтського додатку із серверними компонентами зведено у табл. 3.5.

Табл. 3.5

Інтеграції клієнтського додатку із серверними компонентами

Інтеграція	**Протокол**	**Компонент клієнта**	**Серверний компонент**
Діалог з ШІ-агентами	HTTP POST	chatService.ts → useChat	service-router (API Gateway)
Отримання подій	SSE (EventSource)	pages/index.vue → eventQueueStore	service-sse-stream (Lambda URL)
Список прийомів	HTTP GET	pages/appointments.vue	service-backend (Lambda URL)
Оновлення профілю	SSE → PolicyDetailsRetrieved	handlePolicyUpdated() → petProfileStore	tool-policy-details → event-publisher
Рекомендація лікаря	SSE → ConfirmationMessage	handleRecommendDoctorVisit()	tool-recommend-doctor-visit
Бронювання прийому	SSE → HTTP POST	doctor_visit:confirmed → sendMessage()	service-router → agent-booking-manager

Джерело: складено автором

Варто окремо наголосити, що взаємодію клієнта із зовнішніми API (Amazon Bedrock для виконання агентів, Amazon DynamoDB для персистенції історії діалогу, Amazon SNS та S3 для службових операцій) свідомо опосередковано серверними Lambda-функціями. Прямий виклик

цих сервісів з браузера вимагав би експонування облікових даних або використання Cognito Identity Pools, що ускладнює архітектуру і створює додаткову поверхню атаки. Опосередкування через API Gateway дозволяє централізовано застосовувати авторизацію, обмеження частоти запитів і валідацію вхідних даних, а також приховати внутрішню структуру системи від клієнта. Таким чином, клієнт взаємодіє виключно з публічним шаром платформи (service-router, service-sse-stream, service-backend), а виклики до зовнішніх API інкапсульовані у серверних компонентах.

Підсумовуючи, клієнтський шар побудовано на принципах модульності та чіткого поділу відповідальностей: представлення винесено у Vue-компоненти, бізнес-логіку у composables, а мінімально необхідний глобальний стан у три Pinia-сховища. Технологічні рішення мотивовано природою системи: Nuxt без SSR обрано з огляду на довготривалу SSE-сесію, Pinia замість Vuex завдяки простішій інтеграції з Composition API, SSE замість WebSocket через односпрямований потік подій. Нетривіальними з погляду розробки виявилися буферизація eventQueue, яка запобігає конфлікту оновлень під час генерації відповіді агента, та координація міжагентних сценаріїв через підтвердження користувача, яка дозволила уникнути прямого зв’язування серверних агентів між собою. Реалізовані серверна та клієнтська частини створюють цілісну технічну основу для всебічного тестування платформи, результати якого наведено у наступному підрозділі.

3.3. Тестування платформи: перевірка сценаріїв діалогів, навантажувальне тестування та валідація відповідей

Тестування проведено на чотирьох рівнях: модульне (Jest 29.x [35]), інтеграційне (AWS SAM Local), сценарне (API Gateway) та навантажувальне (Artillery [32]). Піраміду тестування із розподілом за рівнями наведено на рис. 3.8.

Рис. 3.8 Піраміда тестування платформи «ВетЕксперт»

26 тестових сценаріїв виконано успішно (100% pass rate): парсинг JSON (4), валідація message (6), конфігурація агентів (4), ChatHistoryService (9), каскадна валідація policyId (3). Результати сценарного наскрізного тестування зведено у табл. 3.6.

Табл. 3.6

Сценарії наскрізного тестування

Сценарій	**Потік агентів**	**Верифікація**	**Результат**
Ідентифікація полісу	User → PolicyAgent → tool-policy-details → SSE	PolicyDetailsRetrieved за 2-4 с	Passed (30/30)
Ветеринарна консультація	User → VetDocAgent → SaveContext → SSE	Структурована оцінка	Passed (18/20)
Бронювання прийому	User → BookingAgent → find/book-vet-clinic → SSE	AppointmentBooked	Passed

Сценарій	Потік агентів	Верифікація	Результат
Ізоляція відмов	AgentNotFoundException, помилка Bedrock	5 рівнів fallback	Passed

Джерело: складено автором

Два невдалі сценарії ветеринарної консультації (10%) зумовлені варіативністю генерації LLM: в одному випадку агент не виконав уточнювальне запитання перед формуванням оцінки, в іншому випадку структура відповіді не відповідала заданому формату instruction.txt, хоча зміст був коректним. Обидва випадки не стосуються критичних помилок діагностики та пов'язані з недетермінованістю мовної моделі.

Навантажувальне тестування виконано за допомогою Artillery [32] у три фази: Warm-up (лінійне нарощування від 1 до 5 rps протягом 60 с), Normal (стала інтенсивність 20 rps протягом 120 с) та Peak (нарощування від 20 до 50 rps протягом 60 с). За звітом Artillery зафіксовано 3 900 завершених HTTP-сценаріїв; кількість менша за добуток пікових значень на тривалість фаз (5 700), оскільки у фазах Warm-up та Peak інтенсивність зростає лінійно і середня швидкість надходження сценаріїв нижча за цільову пікову. Результати за перцентилями та фазами наведено у табл. 3.7.

Табл. 3.7

Результати навантажувального тестування

Метрика	Warm-up (5 rps)	Normal (20 rps)	Peak (50 rps)
p50	3 240 мс	3 870 мс	5 120 мс
p95	6 180 мс	7 450 мс	9 340 мс
p99	8 920 мс	11 230 мс	14 680 мс
Успішних (2xx)	99,7%	98,9%	97,2%

Джерело: складено автором

Основним «вузьким місцем» є Bedrock (62,5% часу). SSE: доставка 2,1-4,2 с, 100+ з'єднань. Холодний старт Lambda до 1 800 мс; патерн

Singleton (ініціалізація DynamoDBClient та BedrockAgentRuntimeClient на рівні модуля, поза обробником) зменшує повторну ініціалізацію клієнтів при warm invocation до 50-120 мс. Детальну декомпозицію часу відповіді за складовими подано на рис. 3.9.

Рис. 3.9 Декомпозиція часу відповіді Service Router

Точність класифікатора на вибірці з 40 повідомлень становить 95%: PolicyAgent 100%, VetDocAgent 90%, BookingAgent 100%, AgentNotFoundException 90%. 100% з 500 відповідей відповідають BackendChatResponse. Ветеринарний агент (20 сесій): уточнення 95%, оцінка 90%, RecommendDoctorVisit 85%. Зведені результати відповідності нефункціональним вимогам подано у табл. 3.8.

Табл. 3.8

Зведені результати тестування відповідності нефункціональним вимогам

Вимога	**Метрика**	**Цільове**	**Фактичне**	**Статус**
NFR-01	Час відповіді (p95)	< 10 с	7,45 / 9,34 с	Виконано
NFR-02	Одночасних користувачів	50+	100+ SSE, 97,2%	Виконано
NFR-03	Доставка SSE	< 5 с	2,1-4,2 с	Виконано
NFR-07	Ізоляція відмов	Fallback	5 рівнів	Виконано
(додатково)	Точність класифікації	> 90%	95% (38/40)	Виконано

Джерело: складено автором

Слід зазначити обмеження статистичної значущості результатів валідації: при вибірці з 40 повідомлень довірчий інтервал точності класифікації (95% CI) становить 83,1-99,4% (метод Вілсона); при 20 сесіях ветеринарного агента відповідний інтервал становить 68,3-98,8%. Для звуження довірчих інтервалів та підвищення статистичної надійності результатів необхідна вибірка щонайменше 200 повідомлень.

Обмеження: p99 при піку до 14,7 с, варіативність інструментів (85%), холодні старти (до 1,8 с).

Отже, проведене чотирирівневе тестування підтверджує відповідність платформи «ВетЕксперт» усім сформульованим нефункціональним вимогам: p95 часу відповіді 7,45 / 9,34 с (NFR-01), 97,2% успішних запитів при 100+ одночасних SSE-з’єднаннях (NFR-02), доставка подій 2,1-4,2 с (NFR-03), п’ятирівнева ізоляція відмов (NFR-07) та точність класифікації намірів 95% при 100% структурованості відповідей. Водночас окреслено обмеження у p99 на пікових навантаженнях, варіативності

ветеринарного агента та статистичній значущості валідаційної вибірки. Підтверджена технічна спроможність системи створює підґрунтя для переходу до оцінки економічної доцільності впровадження платформи, що виконано у наступному підрозділі.

3.4. Техніко-економічне обґрунтування проєкту та оцінка ефективності впровадження

Загальна трудомісткість реалізації оцінена як $T = 384$ людино-години (≈ 48 робочих днів при тривалості зміни 8 год). Капітальні витрати на створення платформи зводять до величини C_{dev} за моделлю «прямі трудові + overhead»: погодинна ставка помножується на обсяг робіт і збільшується на коефіцієнт накладних (адміністрування, інфраструктура розробки, узгодження).

Рівняння (3.1). Модель вартості розробки. Капітальні витрати визначають як добуток трудомісткості, погодинної ставки та множника $(1 + K_{oh})$:

$$C_{dev} = T \times S_h \times (1 + K_{oh}) = 384 \times 25 \times 1{,}2 = 11\,520\ USD, \qquad (3.1)$$

де C_{dev}: капітальні витрати на розробку платформи, USD;

T: сумарна трудомісткість, людино-год;

S_h: погодинна ставка виконавця, USD/год (за орієнтиром [36] прийнято $S_h = 25$ USD/год);

K_{oh}: коефіцієнт накладних витрат ($K_{oh} = 0{,}2$).

Отримане C_{dev} далі використовують як інвестицію при розрахунку окупності та ROI, а також як основу амортизаційної складової вартості однієї сесії поруч з операційними витратами з табл. 3.9.

Табл. 3.9

Вартість експлуатації хмарної інфраструктури за сценаріями навантаження

Компонент	**1 000 сесій/міс**	**10 000 сесій/міс**	**50 000 сесій/міс**
Claude 3.5 Haiku (LLM)	7,75 USD	77,50 USD	387,50 USD
DynamoDB (on-demand)	0,05 USD	0,50 USD	2,50 USD
Lambda + API GW + SNS + S3	0,14 USD	0,98 USD	4,70 USD
CloudWatch	0,50 USD	2,00 USD	8,00 USD
Разом	8,44 USD	80,98 USD	402,70 USD
На сесію	0,0084 USD	0,0081 USD	0,0081 USD

Джерело: авторські розрахунки за тарифами постачальників (Claude 3.5 Haiku у межах екосистеми AWS [13], [24]) та конфігурацією навантаження, наведеною у тексті

Структуру витрат за компонентами для базового сценарію (10 000 сесій/міс) візуалізовано на рис. 3.10.

Рис. 3.10 Структура витрат на експлуатацію хмарної інфраструктури за компонентами

Частка витрат на LLM у структурі місячних операційних витрат за сценаріями табл. 3.9 (1 000-50 000 сесій/міс) становить 91,8-96,2% (розрахунок як відношення рядка «Claude 3.5 Haiku» до «Разом»). У порівнянні з виділеними обчислювальними ресурсами безсерверні архітектури дозволяють зменшити операційні витрати за рахунок оплати за фактичне споживання та відсутності простою провізіонованих інстансів [14, 15]. Порівняння ключових показників ефективності платформи «ВетЕксперт» із традиційним підходом наведено у табл. 3.10.

Табл. 3.10

Порівняння ключових показників ефективності обслуговування

Показник	**Традиційний підхід**	**Платформа «ВетЕксперт»**	**Покращення**
Час обробки звернення (повний цикл)	25 хв	1-2 хв (3-5 обмінів; час відповіді p95 = 9,3 с)	у 12-25 разів
Очікування клієнта	5-15 хв	0 с (миттєвий доступ)	Ліквідація черги
Доступність	8-12 год/добу	24/7 (99,5%)	у 2-3 рази
Вартість звернення (з амортизацією розробки)	12,50 USD	0,04 USD	у 312 разів
Одночасне обслуговування	1 клієнт/оператор	100+ сесій	Необмежене
Точність класифікації запитів	Залежить від спеціаліста	95%	Стандартизація

Джерело: складено автором

Рівняння (3.2). Повна вартість однієї сесії. Щоб узгодити капітальні витрати з потоковою метрикою «вартість звернення», C_{dev} рівномірно розподіляють на горизонт амортизації $M_{amort} = 36$ міс та на місячну інтенсивність сесій $N_{sess} = 10\,000$; операційну складову $C_{ops}^{sess} \approx 0{,}008$ USD беруть з рядка «На сесію» табл. 3.9:

$$C_{amort}^{sess} = \frac{C_{dev}}{M_{amort} \cdot N_{sess}} = \frac{11\,520}{36 \times 10\,000} \approx 0{,}032\ \frac{USD}{\text{сесія}}, C_{full}^{sess} \approx C_{ops}^{sess} + C_{amort}^{sess} \approx \approx 0{,}04\ \frac{USD}{\text{сесія}}, \quad (3.2)$$

де C_{amort}^{sess}: амортизаційна частка розробки на одну сесію, USD/сесія;

M_{amort}: горизонт амортизації, міс ($M_{amort} = 36$);

N_{sess}: місячна інтенсивність сесій, 1/міс ($N_{sess} = 10\,000$);

C_{ops}^{sess}: операційна вартість сесії за табл. 3.9, USD/сесія;

C_{full}^{sess}: повна оцінка вартості сесії, USD/сесія.

При зміні інтенсивності звернень C_{amort}^{sess} змінюється обернено пропорційно до N_{sess}, тоді як C_{ops}^{sess} визначається тарифами та профілем токенів (див. рівняння (2.9)). Залежність місячного економічного ефекту від кількості сесій для обох сценаріїв відображено на рис. 3.11.

Рис. 3.11 Залежність місячного економічного ефекту від кількості сесій

Економічний ефект оцінено за двома сценаріями: песимістичним ($K_{auto} = 0{,}20$; $K_{sub} = 0{,}05$) та оптимістичним ($K_{auto} = 0{,}30$; $K_{sub} = 0{,}05$). Коефіцієнт K_{auto} відображає частку повністю автоматизованих звернень у перший рік. Значення $K_{sub} = 0{,}05$ прийнято консервативно: у перший рік після впровадження зекономлений робочий час рідко одразу перетворюється на скорочення штату, а реалізується через відкладення найму, скорочення овертаймів та оптимізацію графіків. Такий поріг відповідає нижній межі оцінок реальної монетизації економії в проєктах автоматизації клієнтського сервісу за галузевими звітами [24].

Рівняння (3.3). Базова місячна вартість традиційного обслуговування. Місячну вартість «людського» обслуговування оцінюють як добуток кількості звернень, середнього часу обробки та грошового еквіваленту хвилини роботи оператора. Запис задає верхню орієнтовну межу C_{trad}^{month}, до якої далі застосовують коефіцієнти K_{auto} та K_{sub}:

$$C_{trad}^{month} = N \times T_{avg} \times S_{op} = 10\,000 \times 25 \times 0{,}50 = 125\,000\ \frac{USD}{\text{міс}}, \quad (3.3)$$

де C_{trad}^{month}: базова місячна вартість традиційного обслуговування, USD/міс;

N: кількість звернень за місяць ($N = 10\,000$);

T_{avg}: середній час обробки одного звернення, хв/сесія ($T_{avg} = 25$);

S_{op}: вартість хвилини роботи оператора, USD/хв ($S_{op} = 0{,}50$).

Отримане C_{trad}^{month} узгоджується з порядком величини 12,50 USD/сесія у табл. 3.10 ($C_{trad}^{month}/N = 12{,}50$).

Місячні витрати на експлуатацію платформи становлять $C_{plat}^{month} = C_{ops}^{month} + C_{sup}^{month} = 80{,}98 + 500 = 580{,}98$ USD/міс, де C_{ops}^{month} становить вартість хмарної інфраструктури (табл. 3.9), а $C_{sup}^{month} = 500$ USD відповідає частковій зайнятості DevOps-інженера (≈ 0,25 FTE, або 20 год/міс за ставкою $S_h = 25$ USD/год) на моніторинг, оновлення та усунення інцидентів.

Рівняння (3.4). Місячний економічний ефект (песимістичний сценарій). Добуток $K_{sub}K_{auto}C_{trad}^{month}$ задає грошову оцінку тієї частини витрат кол-центру, яка перетворюється на реальний виграш; віднімання C_{plat}^{month} робить показник нетто:

$$E_{month}^{pess} = K_{sub} \times K_{auto}^{pess} \times C_{trad}^{month} - C_{plat}^{month} =$$
$$= 0{,}05 \times 0{,}20 \times 125\,000 - 580{,}98 \approx 669\ \frac{USD}{\text{міс}}, \quad (3.4)$$

де E_{month}^{pess}: місячний економічний ефект за песимістичним сценарієм, USD/міс;

K_{sub}: коефіцієнт монетизації зекономленого часу ($K_{sub} = 0{,}05$); K_{auto}^{pess}: песимістична частка повністю автоматизованих звернень ($K_{auto}^{pess} = 0{,}20$);

C_{plat}^{month}: сукупні місячні витрати на платформу, USD/міс ($C_{plat}^{month} = 580{,}98$).

Термін окупності: $T_{payback} = C_{dev}/E_{month}^{pess} = 11\,520/669 \approx 17{,}2$ міс. Оскільки $T_{payback} > 12$ міс, ROI за перший рік набуває від'ємного значення, що характерно для платформних інвестицій на старті.

Рівняння (3.5). ROI за перший рік (песимістичний сценарій). ROI визначають як відношення чистого річного ефекту до капітальних витрат:

$$ROI_{Y1}^{pess} = \frac{12\,E_{month}^{pess} - C_{dev}}{C_{dev}} = \frac{12 \times 669 - 11\,520}{11\,520} \approx -0{,}303, \qquad (3.5)$$

тобто близько $-30{,}3\%$ (табл. 3.11). Показник не дисконтує місячні надлишки (на відміну від NPV_3), тому умова $ROI_{Y1} > 0$ збігається з $T_{payback} < 12$ міс. Від'ємне значення означає, що накопичена за 12 міс економія не перекриває C_{dev}; додатне NPV_3 на трирічному горизонті нижче ілюструє довгострокову картину.

Рівняння (3.6). Місячний економічний ефект (оптимістичний сценарій). При зростанні частки автоматизованих звернень до $K_{auto}^{opt} = 0{,}30$ інші величини не змінюються:

$$E_{month}^{opt} = K_{sub} \times K_{auto}^{opt} \times C_{trad}^{month} - C_{plat}^{month} =$$
$$= 0{,}05 \times 0{,}30 \times 125\,000 - 580{,}98 \approx 1\,294\ \frac{USD}{\text{міс}}, \qquad (3.6)$$

Приріст K_{auto} з 0,20 до 0,30 дає нетто-додаток $K_{sub}\Delta K_{auto}\, C_{trad}^{month} = 625$ USD/міс і переводить E_{month} з 669 до 1 294 USD. Термін окупності скорочується до $T_{payback} = 11\,520/1\,294 \approx 8{,}9$ міс < 12 міс, тому ROI за перший рік стає додатним.

Рівняння (3.7). ROI за перший рік (оптимістичний сценарій).

$$ROI_{Y1}^{opt} = \frac{12\,E_{month}^{opt} - C_{dev}}{C_{dev}} = \frac{12 \times 1\,294 - 11\,520}{11\,520} \approx 0{,}348, \qquad (3.7)$$

тобто близько 34,8%. Зміщення K_{auto} на 10 п.п. переводить ROI з $-30{,}3\%$ до $+34{,}8\%$, що демонструє високу чутливість показника до частки автономно оброблених звернень і визначає напрям подальшої роботи над маршрутизацією.

Рівняння (3.8). Чиста приведена вартість на горизонті трьох років. На відміну від ROI у (3.5) та (3.7), NPV_3 дисконтує очікувані річні надлишки E_{year} до моменту прийняття рішення ($t = 0$); для табл. 3.11 обрано річну ставку $r = 0{,}12$:

$$NPV_3 = -C_{dev} + \sum_{t=1}^{3} \frac{E_{year}}{(1+r)^t}, \tag{3.8}$$

де NPV_3: чиста приведена вартість на трирічному горизонті, USD; E_{year}: річний нетто-надлишок, USD/рік ($E_{year} = 12\,E_{month}$ за відповідним сценарієм; прийнято сталим у всіх трьох роках); r: річна ставка дисконтування ($r = 0{,}12$); t: індекс року ($t = 1{,}2{,}3$); коефіцієнт $(1+r)^t$ приводить потік кінця року t до $t = 0$.

Підстановка $E_{year}^{pess} = 8\,028$ USD/рік дає $NPV_3^{pess} \approx 7\,762$ USD; для $E_{year}^{opt} = 15\,528$ USD/рік отримуємо $NPV_3^{opt} \approx 25\,774$ USD (табл. 3.11). Додатне NPV_3 в обох сценаріях підтверджує довгострокову доцільність інвестиції навіть при від’ємному ROI першого року.

Табл. 3.11

Зведені економічні показники проєкту «ВетЕксперт»

Показник	**Песимістичний сценарій**	**Оптимістичний сценарій**
Трудомісткість розробки	384 людино-години (2,4 місяці)	384 людино-години (2,4 місяці)
Вартість розробки (C_{dev})	11 520 USD	11 520 USD

Показник	**Песимістичний сценарій**	**Оптимістичний сценарій**
Місячні витрати на платформу (C_{plat}^{month})	580,98 USD	580,98 USD
Місячна економія (E_{month})	669 USD (K_{auto}=0,20)	1 294 USD (K_{auto}=0,30)
Термін окупності	17,2 місяці	8,9 місяців
ROI за перший рік	$-30{,}3\%$	34,8%
NPV_3 ($r = 12\%$)	7 762 USD	25 774 USD

Джерело: складено автором

Отже, техніко-економічне обґрунтування підтверджує фінансову доцільність проєкту «ВетЕксперт». При $C_{dev} = 11\,520$ USD та місячних витратах на платформу 580,98 USD вартість обробки одного звернення знижується з 12,50 до 0,04 USD. Термін окупності становить 8,9-17,2 місяці, ROI першого року коливається від −30,3% до +34,8%, NPV_3 коливається від 7 762 до 25 774 USD. Додатне NPV_3 в обох сценаріях підтверджує довгострокову ефективність впровадження, а чутливість ROI до K_{auto} задає пріоритетний напрям подальшого розвитку, передусім оптимізацію маршрутизації та розширення покриття типових сценаріїв для зростання частки автономно обслугованих звернень.

ВИСНОВКИ

Мету кваліфікаційної роботи досягнуто: розроблено та програмно реалізовано інтелектуальну хмарну платформу медичного страхування домашніх тварин «ВетЕксперт», що в межах одного діалогу поєднує страхову та ветеринарну експертизу на основі взаємодії спеціалізованих ШІ-агентів. Сім задач дослідження, сформульованих у підрозділі 1.4, виконано в повному обсязі.

Узагальнено роль інтелектуальних систем в автоматизації клієнтського обслуговування та простежено їхню еволюцію від експертних систем до мультиагентних LLM-платформ. Обґрунтовано доцільність мультиагентного підходу для предметної області, у якій звернення клієнта одночасно зачіпає страхову й ветеринарну компетенції, та розглянуто механізми RAG як засіб заземлення відповідей на верифіковані джерела знань [4, 22]. Формалізовано бізнес-процеси у нотаціях AS-IS і TO-BE, визначено варіанти використання та специфіковано нефункціональні вимоги до платформи.

Спроєктовано архітектуру, що узгоджено поєднує безсерверну мікросервісну, мультиагентну та подієво-орієнтовану парадигми. Систему декомпоновано на 14 мікросервісів, серед яких інфраструктурні компоненти, доменні ШІ-агенти Amazon Bedrock, агент-класифікатор намірів і допоміжні Lambda-інструменти. Запропоновано схему контекстно-залежної класифікації клієнтських намірів і маршрутизації запитів між страховим та ветеринарним агентами, чотирирівневий алгоритм оцінки терміновості ветеринарних звернень, а також RAG-конвеєр із міжагентною передачею контексту через blackboard pattern; відповідні рішення формалізовано у восьми алгоритмах підрозділу 2.3. Вибір технологічного стека (Claude 3.5 Haiku, Amazon Bedrock Agents, AWS Lambda, DynamoDB, Nuxt 4 із Vue 3) обґрунтовано порівняльним аналізом альтернатив (табл. 2.11-2.12).

Реалізовано серверну логіку із семикроковим алгоритмом маршрутизації, бібліотеку @vetexpert/agent-tools та клієнтський інтерфейс на Nuxt 4 із шістнадцятьма Vue-компонентами, реактивним управлінням станом і декларативним описом інфраструктури через AWS SAM. Проведено чотирирівневе тестування, а саме модульне (26 сценаріїв, усі пройдено), інтеграційне, сценарне та навантажувальне (до 50 rps, 3 900 запитів), після чого зафіксовано відповідність нефункціональним вимогам: p95 часу однієї відповіді агента дорівнює 9,34 с, система стабільно обслуговує понад 100 одночасних SSE-з'єднань, точність класифікації намірів становить 95% на тестовій вибірці, а чотирирівневий алгоритм оцінки терміновості коректно категоризує невідкладні ветеринарні стани у 90% сценаріїв. Тим самим експериментально підтверджено ключові архітектурно-алгоритмічні положення, сформульовані у підрозділі 1.4.

У техніко-економічному обґрунтуванні оцінено вартість розробки у 11 520 USD, а розрахункова вартість обробки одного звернення з урахуванням амортизації становить 0,04 USD проти 12,50 USD у варіанті з оператором-людиною. Залежно від припущень щодо коефіцієнтів автоматизації та субституції персоналу, термін окупності знаходиться в діапазоні 8,9-17,2 міс., а річний ROI перебуває у діапазоні від −30,3% до +34,8%; NPV_3 за три роки експлуатації при $r = 12\%$ за рівнянням (3.8) становить 7 762 USD за песимістичним і 25 774 USD за оптимістичним сценарієм (табл. 3.11). Це підтверджує економічну доцільність впровадження платформи вже на горизонті першого-другого року експлуатації.

Отримані результати засвідчують практичну придатність мультиагентного підходу до задач страхово-ветеринарного консультування: повний цикл обслуговування скорочується з 15-25 хв до 1-2 хв (3-5 обмінів) при збереженні якості відповідей на рівні, узгодженому з літературними оцінками ефективності RAG-заземлення [4, 22]. Подальший

розвиток платформи доцільно вести у напрямах розширення й галузевої адаптації бази знань RAG, інтеграції мультимодального вводу (фото симптомів, голосові звернення), введення окремого агента обробки страхових випадків та зниження латентності за рахунок кешування LLM-відповідей і оптимізації RAG-конвеєра.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] McKinsey & Company. The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. McKinsey Global Survey, May 2024. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024 (дата звернення: 19.02.2026).

[2] Самодурова Д. А. Інтелектуальні агенти та мультиагентні системи у виробництві. Економічний вісник Донбасу. 2019. № 2(56). С. 179-186. DOI: 10.12958/1817-3772-2019-2(56)-179-186.

[3] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS 2017). 2017. Vol. 30. P. 5998-6008. URL: https://arxiv.org/abs/1706.03762.

[4] Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Küttler H., Lewis M., Yih W., Rocktäschel T., Riedel S., Kiela D. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems (NeurIPS 2020). 2020. Vol. 33. P. 9459-9474. URL: https://arxiv.org/abs/2005.11401.

[5] Бідочко А. Р., Виклюк Я. І. LLMAGENTNET: колаборативна мережа автономних агентів штучного інтелекту для виконання складних завдань. Науковий вісник НЛТУ України. 2025. Т. 35, № 4. С. 101-114. DOI: https://doi.org/10.36930/40350412. URL: https://nv.nltu.edu.ua/index.php/journal/article/view/2779/2781 (дата звернення: 17.04.2026).

[6] Accenture. Technology Vision 2024: Human by Design Technologies Will Reinvent Industries and Redefine Leaders. Accenture Research Report, 2024. URL: https://www.accenture.com/us-en/insights/technology/technology-trends-2024 (дата звернення: 19.02.2026). Також: Accenture. Underwriting Rewritten: How AI and Gen AI Are Reshaping Insurance Underwriting. 2024.

URL: https://www.accenture.com/us-en/insights/insurance/underwriting-rewritten (дата звернення: 19.02.2026).

[7] Федорович І. М. Цифровізація страхового ринку України // Страховий ринок України у світлі євроінтеграції: новітні виклики та тренди : збірник матеріалів VI Міжнародної науково-практичної конференції, м. Київ, 23 березня 2023 р. К. : КНЕУ, 2023. С. 165-167. URL: https://dspace.wunu.edu.ua/items/654c6fa6-0cf4-456c-8314-bb568a33e997 (дата звернення: 17.04.2026).

[8] Deloitte. The Future of Work in Financial Services: AI, Automation, and the Workforce of Tomorrow. Deloitte Insights, 2023. URL: https://www2.deloitte.com/us/en/insights/industry/financial-services/future-of-work-in-financial-services.html (дата звернення: 20.02.2026).

[9] Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p. ISBN: 978-1-492-03402-5.

[10] Fowler M., Lewis J. Microservices: A Definition of This New Architectural Term. martinfowler.com, March 2014. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 20.02.2026).

[11] Wooldridge M. An Introduction to MultiAgent Systems. 2nd ed. Chichester: John Wiley & Sons, 2009. 484 p. ISBN: 978-0-470-51946-2.

[12] Wang L., Ma C., Feng X., Zhang Z., Yang H., Zhang J., Chen Z., Tang J., Chen X., Lin Y., Zhao W. X., Wei Z., Wen J.-R. A Survey on Large Language Model based Autonomous Agents. Frontiers of Computer Science. 2024. Vol. 18. No. 6. Article 186345. DOI: 10.1007/s11704-024-40231-1. URL: https://arxiv.org/abs/2308.11432.

[13] Amazon Web Services. Amazon Bedrock Agents: Build Generative AI Applications with Autonomous Agents. AWS Documentation, 2024. URL: https://docs.aws.amazon.com/bedrock/latest/userguide/agents.html (дата звернення: 20.02.2026). Для потокової відповіді Lambda (response streaming,

SSE) також використано: Amazon Web Services. Invoke your Lambda function using response streaming. AWS Documentation, 2024. URL: https://docs.aws.amazon.com/lambda/latest/dg/configuration-response-streaming.html (дата звернення: 05.03.2026).

[14] Jonas E., Schleier-Smith J., Sreekanti V., Tsai C.-C., Khandelwal A., Pu Q., Shankar V., Carreira J., Krauth K., Yadwadkar N., Gonzalez J., Popa R. A., Stoica I., Patterson D. A. Cloud Programming Simplified: A Berkeley View on Serverless Computing. Technical Report No. UCB/EECS-2019-3. Berkeley: University of California, 2019. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html.

[15] Datadog. The State of Serverless 2024. Datadog Research Report, 2024. URL: https://www.datadoghq.com/state-of-serverless (дата звернення: 20.02.2026).

[16] North American Pet Health Insurance Association (NAPHIA). State of the Industry Report 2024. NAPHIA Annual Report, 2024. URL: https://naphia.org/industry-data/section-1-market-overview (дата звернення: 20.02.2026).

[17] Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. draft. Stanford University, 2024. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 21.02.2026). Для порівняння платформ оркестрації (табл. 2.11) також використано: LangChain. LangGraph: Multi-Actor Applications with LLMs. LangChain Documentation, 2024. URL: https://langchain-ai.github.io/langgraph/ (дата звернення: 05.03.2026); Wu Q., Bansal G., Zhang J. et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. EMNLP 2024. P. 193-203.

[18] Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of

NAACL-HLT 2019. Minneapolis: ACL, 2019. P. 4171-4186. URL: https://arxiv.org/abs/1810.04805.

[19] Brown T. B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS 2020). 2020. Vol. 33. P. 1877-1901. URL: https://arxiv.org/abs/2005.14165.

[20] Hendrycks D., Burns C., Basart S. et al. Measuring Massive Multitask Language Understanding. Proceedings of ICLR 2021. 2021. URL: https://arxiv.org/abs/2009.03300.

[21] Wei J., Wang X., Schuurmans D. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems (NeurIPS 2022). 2022. Vol. 35. P. 24824-24837. URL: https://arxiv.org/abs/2201.11903. Для оцінки економічного ефекту впровадження Amazon Bedrock також використано: Forrester Consulting. The Total Economic Impact of Amazon Bedrock. Commissioned Study, 2024. URL: https://aws.amazon.com/bedrock/forrester-tei/ (дата звернення: 05.03.2026).

[22] Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint arXiv:2312.10997. 2024. URL: https://arxiv.org/abs/2312.10997 (дата звернення: 21.02.2026).

[23] FIPA. FIPA Agent Communication Language Specifications. Foundation for Intelligent Physical Agents, 2002. URL: http://www.fipa.org/repository/aclspecs.html (дата звернення: 21.02.2026).

[24] Talebirad Y., Nadiri A. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. arXiv preprint arXiv:2306.03314. 2023. URL: https://arxiv.org/abs/2306.03314 (дата звернення: 21.02.2026). Для тарифів і характеристик моделей Claude також використано: Anthropic. The Claude Model Family. Anthropic Technical Report, 2024. URL: https://docs.anthropic.com/en/docs/about-claude/models (дата звернення: 05.03.2026).

[25] American Veterinary Medical Association (AVMA). AVMA Pet Ownership and Demographics Sourcebook: 2022-2023 Edition. Schaumburg: AVMA, 2023. 192 p.

[26] Huang L., Yu W., Ma W. et al. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. ACM Computing Surveys. 2024. Vol. 57. No. 2. Article 40. DOI: 10.1145/3703155. URL: https://arxiv.org/abs/2311.05232.

[27] Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education, 2016. 816 p. ISBN: 978-0-13-394303-0.

[28] Dumas M., La Rosa M., Mendling J., Reijers H. A. Fundamentals of Business Process Management. 2nd ed. Berlin: Springer, 2018. 527 p. DOI: 10.1007/978-3-662-56509-4. ISBN: 978-3-662-56508-7.

[29] Zhou J., Lu T., Mishra S. et al. Instruction-Following Evaluation for Large Language Models. Proceedings of the 62nd Annual Meeting of the ACL. 2024. P. 12694-12713.

[30] Swiss Re Institute. Digital Ecosystems in Insurance: Connecting the Dots. Swiss Re Sigma Report No. 5/2024, 2024. URL: https://www.swissre.com/institute/research/sigma-research/sigma-2024-05.html (дата звернення: 25.02.2026). Для порівняння хмарних БД (табл. 2.12) додатково використано: Amazon Web Services. Amazon DynamoDB Developer Guide. AWS Documentation, 2024. URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ (дата звернення: 05.03.2026); Amazon Web Services. Amazon Aurora Serverless v2 User Guide. AWS Documentation, 2024. URL: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-serverless-v2.html (дата звернення: 05.03.2026).

[31] Corkill D. D. Blackboard Systems. AI Expert. 1991. Vol. 6, No. 9. P. 40-47.

[32] Artillery. Artillery Documentation: Load Testing at Scale. 2024. URL: https://www.artillery.io/docs (дата звернення: 15.03.2026). Для середовища виконання Lambda (Node.js) також використано: OpenJS Foundation. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 05.03.2026).

[33] Nuxt. Nuxt 4 Documentation. 2025. URL: https://nuxt.com/docs (дата звернення: 05.03.2026). Для протоколу Server-Sent Events також використано: Hickson I. Server-Sent Events. W3C Recommendation, February 2015. URL: https://www.w3.org/TR/eventsource/ (дата звернення: 05.03.2026).

[34] Amazon Web Services. AWS Serverless Application Model (SAM) Developer Guide. AWS Documentation, 2024. URL: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html (дата звернення: 05.03.2026).

[35] Jest. Jest Documentation: Delightful JavaScript Testing. 2024. URL: https://jestjs.io/docs/getting-started (дата звернення: 15.03.2026).

[36] DOU. Salary statistics for Ukrainian IT specialists. 2025. URL: https://jobs.dou.ua/salaries/ (дата звернення: 25.03.2026).

ДОДАТОК А. ЕКРАННІ ФОРМИ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ ПЛАТФОРМИ «ВЕТЕКСПЕРТ»

А.1. Головний екран діалогу з мультиагентною системою

Основний робочий екран платформи реалізує патерн «двоколонковий робочий простір» (two-column workspace) та об’єднує контекстну картку улюбленця із вікном діалогу з мультиагентною системою. Загальний вигляд екрана наведено на рис. А.1.

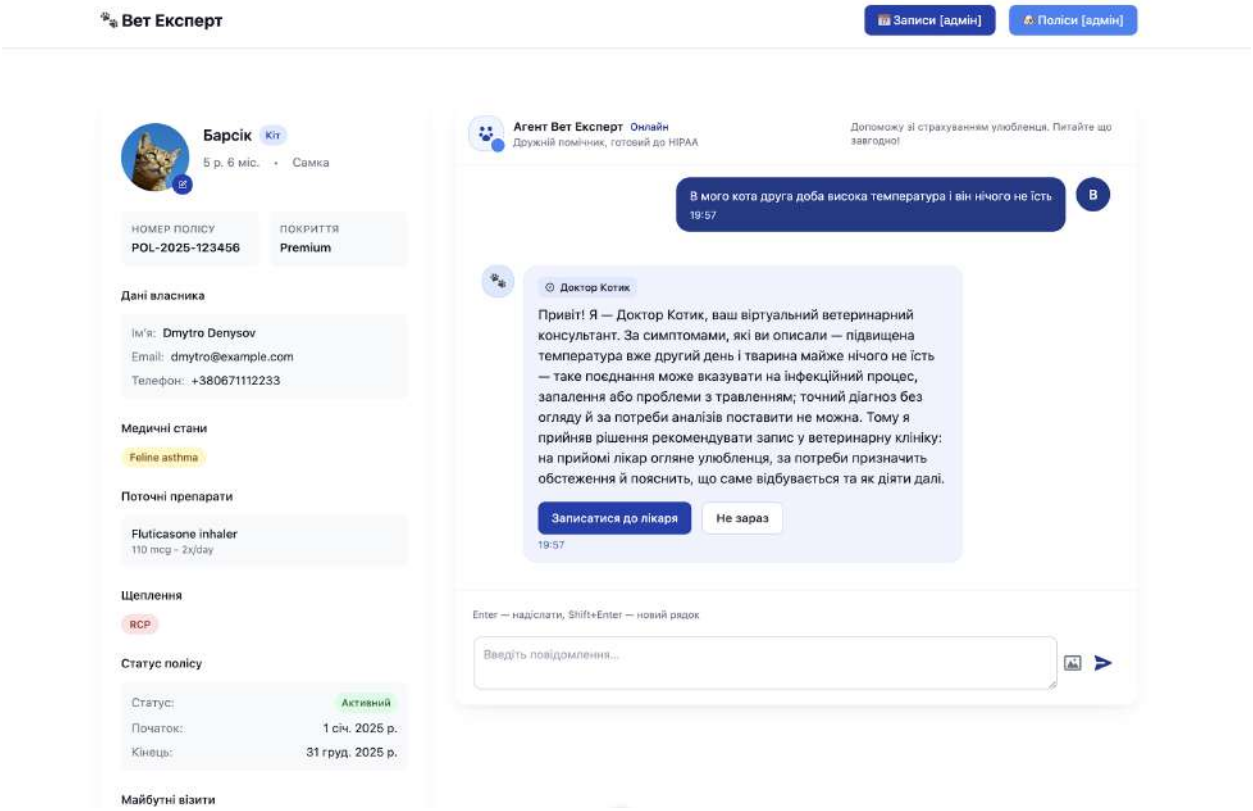


Рис. А.1 Головний екран платформи «ВетЕксперт»: діалог користувача з ветеринарним агентом «Доктор Котик» та контекстна картка улюбленця

Ліва колонка містить статичний контекст, що автоматично передається агентам як частина системного промпту і не потребує повторного введення користувачем. Серед елементів картки улюбленця: фотографія та загальні атрибути (ім’я «Барсік», вид «Кіт», вік «5 р. 6 міс.», стать «Самка»); ідентифікатор страхового полісу (POL-2025-123456) та рівень покриття (Premium); блок «Дані власника» з іменем, email та

телефоном; блок «Медичні стани» з тегом хронічного захворювання (Feline asthma); блок «Поточні препарати» зі схемою дозування (Fluticasone inhaler, 110 mcg, 2x/day); блок «Щеплення» з переліком діючих вакцин (RCP); блок «Статус полісу» з візуальним індикатором активності та термінами дії. Така структура відповідає моделі даних DynamoDB, описаній у підрозділі 2.2 (табл. 2.8), і забезпечує повний контекст для прийняття агентом обґрунтованих рекомендацій без додаткових уточнювальних запитань.

Права колонка реалізує інтерфейс спілкування за принципом месенджера. Верхня панель ідентифікує активного агента («Агент Вет Експерт», статус «Онлайн», підзаголовок «Дружній помічник, готовий до HIPAA»). Повідомлення користувача розташовані праворуч на темно-синьому тлі, тоді як відповіді агента відображаються ліворуч у вигляді картки з заголовком, що містить ім’я субагента («Доктор Котик»), до якого делегувало запит кореневе рішення. Важливим елементом є інтерактивні кнопки під повідомленням агента («Записатися до лікаря», «Не зараз»), які реалізують патерн guided actions: натискання такої кнопки трансформується у структурне повідомлення, адресоване агенту бронювання, що відповідає механізму міжагентної координації через користувача (див. підрозділ 3.2). У нижній частині панелі розташовано поле введення з підказкою щодо клавіш (Enter надсилає повідомлення, Shift+Enter відкриває новий рядок), кнопкою прикріплення зображення та кнопкою відправки. У верхньому правому куті розміщено адміністративні кнопки переходу до панелей «Записи [адмін]» та «Поліси [адмін]».

А.2. Адміністративна панель улюбленців та полісів

Для задач страхового бек-офісу реалізовано окрему адміністративну панель, що надає консолідоване подання всіх зареєстрованих улюбленців та їхніх полісів. Екран наведено на рис. А.2.

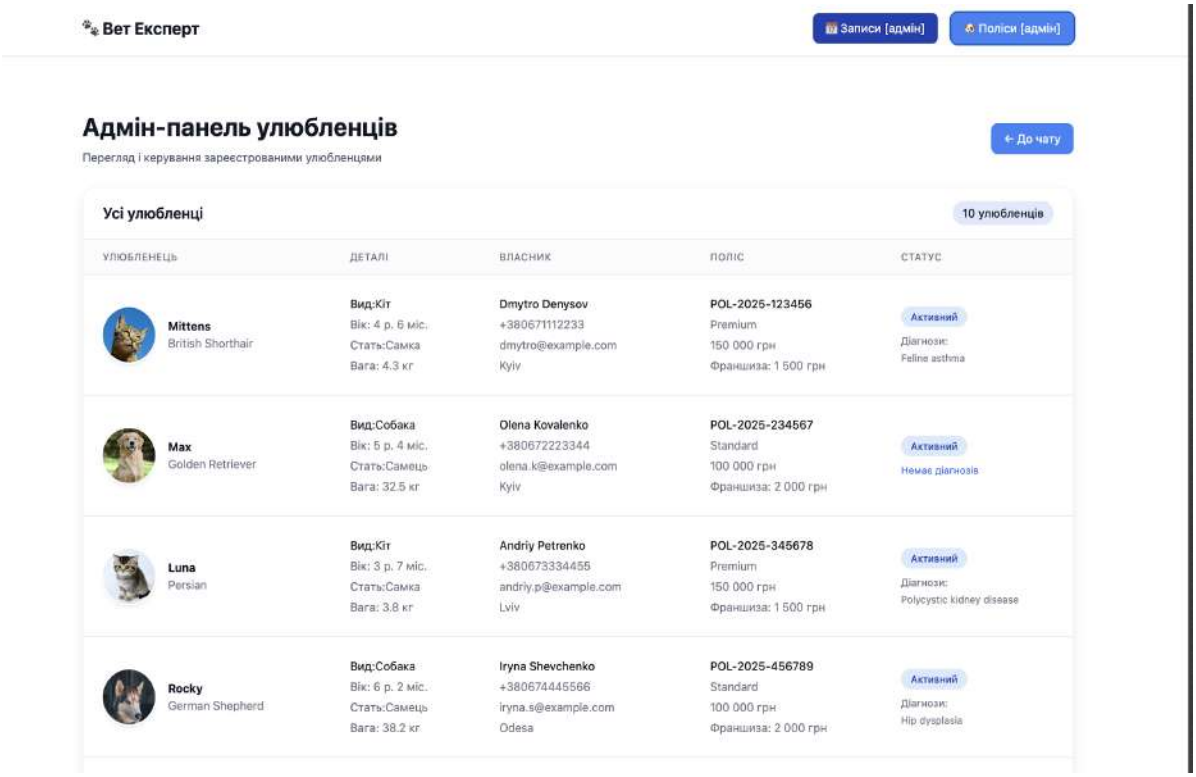


Рис. А.2 Адміністративна панель управління улюбленцями та страховими полісами

Екран побудовано за принципом табличного подання з фіксованим заголовком і групуванням інформації за колонками. Верхня частина містить заголовок «Адмін-панель улюбленців» із підзаголовком «Перегляд і керування зареєстрованими улюбленцями» та кнопку повернення «← До чату», що забезпечує навігацію між основним діалогом та адміністративним контуром. Ліворуч від таблиці розташовано загальний лічильник (10 улюбленців), який дозволяє оператору швидко оцінити обсяг портфеля.

Таблиця поділена на п’ять логічних колонок: (1) у колонці «Улюбленець» наведено мініатюру фотографії, кличку та породу (наприклад, Mittens, British Shorthair); (2) у колонці «Деталі» подано перелік атрибутів у форматі «ключ: значення» (вид, вік, стать, вага); (3) у колонці «Власник» вказано контактні дані (ПІБ, телефон, email, місто); (4) у колонці «Поліс» відображаються номер полісу, рівень покриття, страхова сума та франшиза; (5) у колонці «Статус» подано візуальний індикатор активності

полісу та перелік діагнозів (із явним значенням «Немає діагнозів» у разі відсутності), що використовується агентом для оцінки ризиків. Такий формат дозволяє оператору за один крок отримати повну картину клієнта, що важливо для верифікації звернень і розрахунку страхових виплат.

А.3. Панель ветеринарних записів

Третій екран адміністративного контуру відповідає за управління записами на ветеринарні візити, сформованими агентом бронювання в результаті діалогу з користувачем. Екран наведено на рис. А.3.

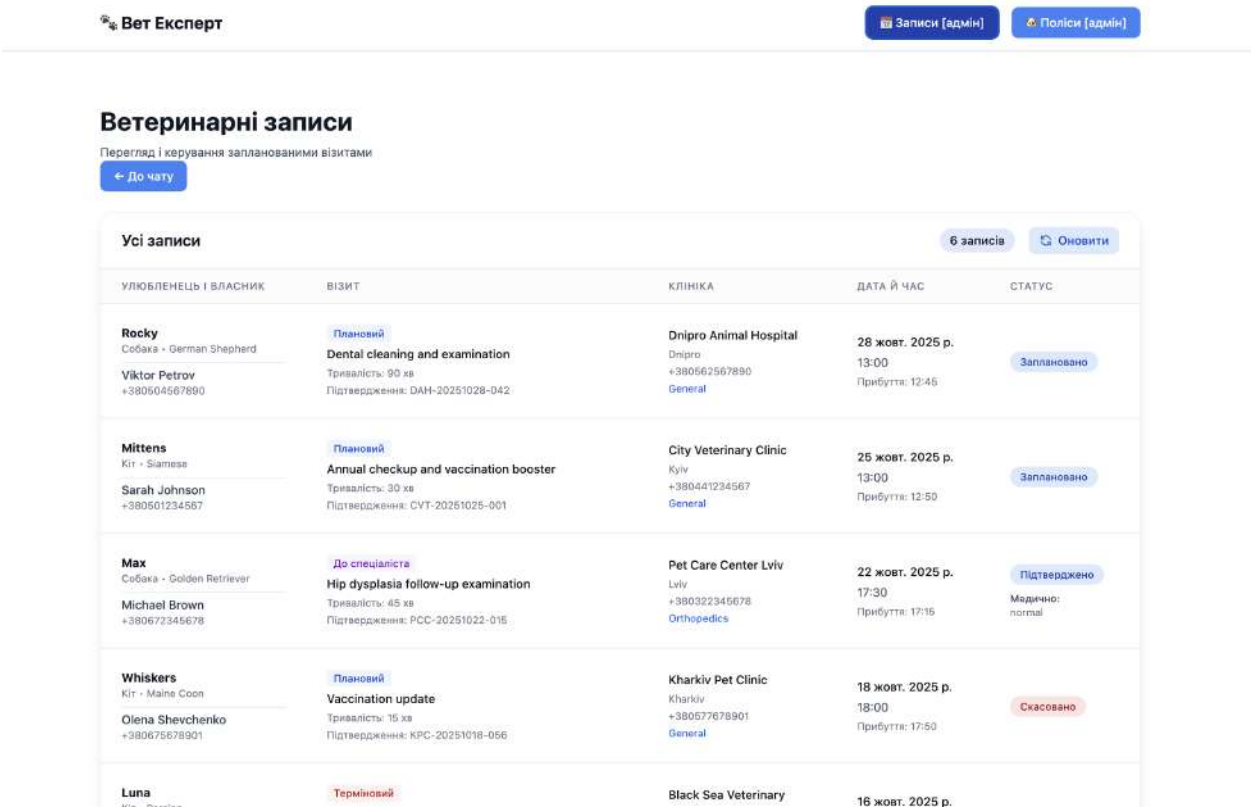


Рис. А.3 Панель управління ветеринарними записами, створеними мультиагентною системою

Структурно екран повторює патерн адміністративної панелі (заголовок «Ветеринарні записи», підзаголовок «Перегляд і керування запланованими візитами», кнопка «← До чату»), але оптимізований для специфіки бронювань. У правому верхньому куті розміщено лічильник

записів (6 записів) та кнопку «Оновити», що повторно викликає API зчитування з DynamoDB без перезавантаження сторінки.

Таблиця складається з п’яти колонок: (1) у колонці «Улюбленець і власник» зібрано об’єднаний блок з інформацією про тварину та її власника; (2) у колонці «Візит» наведено тип звернення (кольорові теги Плановий, До спеціаліста, Терміновий, що відповідають класифікації тріажу з підрозділу 2.5), назву процедури, її тривалість та унікальний код підтвердження у форматі CVT-YYYYMMDD-NNN; (3) у колонці «Клініка» вказано назву закладу, місто, телефон і спеціалізацію (General, Orthopedics); (4) у колонці «Дата й час» подано дату візиту, його початок та рекомендований час прибуття; (5) у колонці «Статус» відображається стан запису (Заплановано, Підтверджено, Скасовано) з відповідним кольоровим маркуванням та (для підтверджених медичних візитів) категорія пріоритетності (normal). Таке подання дозволяє оператору відстежувати повний життєвий цикл запису, тобто від моменту його створення агентом у діалозі до фактичного відвідування клініки, і реалізує принцип прозорості рішень мультиагентної системи для людини-супервайзера.

ДОДАТОК Б. ПОСИЛАННЯ НА РЕПОЗИТОРІЙ ВИХІДНОГО КОДУ

Вихідний код платформи «ВетЕксперт», розробленої в рамках кваліфікаційної роботи, розміщено у відкритому репозиторії на платформі GitHub та доступний за адресою:

URL: https://github.com/ddenysov/purrsurance (дата звернення: 17.04.2026).

Репозиторій містить повний монорепозиторій проєкту зі структурою, описаною у підрозділі 3.1, включаючи вихідний код мікросервісів (apps/), інфраструктурні шаблони AWS SAM (template.yaml), конфігурацію клієнтського застосунку на Nuxt 4 (apps/frontend/) та документацію з розгортання.

ПРЕЗЕНТАЦІЯ

Макротренди автоматизації: загальний бум ШІ та стагнація у крос-доменних нішах.

Глобальний тренд

72% компаній вже впровадили ШІ (McKinsey, 2024). Основний фокус — автоматизація клієнтського сервісу. Економічний ефект доведено.

Проблема ніші

Звернення у ветеринарному страхуванні вимагають одночасних знань медицини та фінансів. Традиційні шаблонні чат-боти тут не працюють, розв'язуючи лише 40-55% стандартних запитів.

Еволюція інтелектуальних систем: від жорстких правил до мультиагентної автономності.

Етап 1: Експертні системи (1980-90-ті)

Формалізовані бази знань. Жорсткі правила логічного виведення.

Проблема: «Вузьке місце знань» та складність оновлення.

Етап 2: Машинне навчання (2000-10-ті)

Статистичний аналіз.

Проблема: Слабке утримання контексту діалогу та обмеження вузької задачі.

Етап 3: Мультиагентні LLM-платформи (Сучасність)

Трансформерні моделі (GPT-4, Claude).

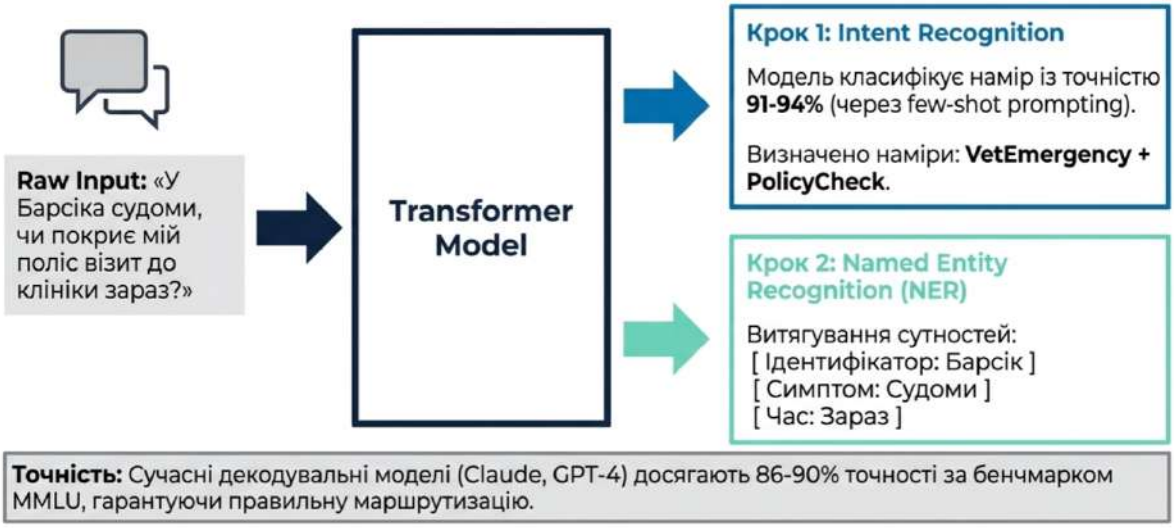
Здатність розуміти наміри, утримувати контекст відкритого домену та делегувати задачі автономним агентам.

Архітектурна матриця: вибір базису для ШІ-сервісів

Критерій	Монолітна	Мікросервісна	Мультиагентна
Масштабованість	Обмежена	Горизонтальна	Динамічна
Автономність компонентів	Відсутня	Часткова	Повна
Робота з неструктурованими запитами	Слабка	Залежить від коду	Нативна через LLM
Стійкість до відмов	Єдина точка відмови	Ізоляція сервісів	Ізоляція + перенаправлення

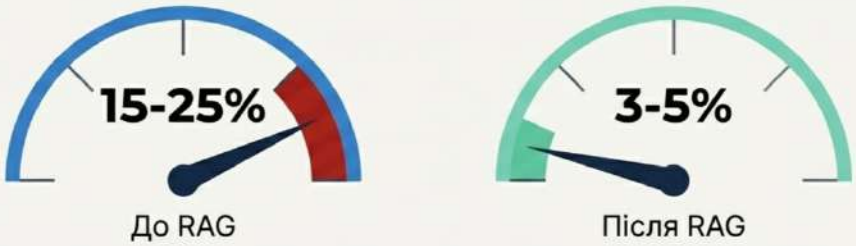
Мультиагентна архітектура забезпечує динамічне масштабування та повну ізоляцію складних ШІ-задач, що є безальтернативним для систем обслуговування.

Механізми обробки мови: розпізнавання наміру та витягування сутностей (NER).



Точність: Сучасні декодувальні моделі (Claude, GPT-4) досягають 86-90% точності за бенчмарком MMLU, гарантуючи правильну маршрутизацію.

Заземлення знань через RAG (Retrieval-Augmented Generation): подолання проблеми галюцинацій.



Проблема:
Базові LLM схильні до галюцинацій (генерації правдоподібних, але хибних фактів) у 15-25% випадків. Це неприпустимо у медицині.

Рішення:
RAG-конвеєр відсікає «фантазії» моделі. Модель генерує відповідь виключно на основі знайдених у векторній базі верифікованих документів (поліси, ветеринарні протоколи).

Результат:
Зниження помилок до 3-5%. Досягається Recall 85-93% та Faithfulness 90-96%.

Функціональні ролі ШІ-агентів платформи.

Страховий менеджер (Policy Agent)

Зона відповідальності: Інформування про тарифи, розрахунок покриття, перевірка франшизи, збір даних для страхового випадку (Правило R-6).

Ветеринарний лікар (Vet Doc Agent)

Зона відповідальності: Збір анамнезу, оцінка симптомів, чотирирівневий алгоритм визначення терміновості (тріаж). Обов'язкове медичне застереження (Правило R-5).

Менеджер записів (Booking Agent)

Зона відповідальності: Пошук релевантних клінік на основі страховки та терміновості, бронювання часу, фіксація запису в БД.

AS-IS (Традиційний процес)
Час обробки (Повний цикл): 15–45 хв.
Передачі між спеціалістами (Handoffs): 2–3
Одночасні сесії: 1 клієнт/оператор
Вартість звернення: ~12,50 USD
TO-BE (Мультиагентна ШІ-платформа)
Час обробки (Повний цикл): 1–3 хв. (У 8–25 разів швидше)
Передачі між спеціалістами (Handoffs): 0 (Автоматична оркестрація)
Одночасні сесії: 100+ (Горизонтальне масштабування)
Вартість звернення: 0,04 USD (З урахуванням амортизації)
Трансформація ручного лінійного процесу в паралельну подієво-орієнтовану хмарну архітектуру.

Системні інваріанти та бізнес-правила прийняття рішень
R-1 Класифікація
Кожен запит проходить обов'язкову класифікацію намірів через оркестратор перед маршрутизацією.
R-3 Заземлення (RAG)
Усі фактичні медичні твердження повинні мати підтвердження у вигляді процитованого фрагмента з верифікованої бази знань.
R-5 Медичне застереження
Ветеринарна консультація автоматично супроводжується юридичним застереженням, що не замінює очного візиту.
R-7 Збереження контексту
Контекст сесії (поліс, симптоми) безперервно зберігається та передається між агентами для цілісності діалогу.

Обґрунтування стека: Оркестрація та Бази даних

Оркестрація - Bedrock Agents

Критерій	Bedrock	LangChain	AutoGen
Managed Service	✓ Керований сервіс AWS	✗ Self-managed	✗ Self-managed
Нативна інтеграція Lambda	✓ Так	✗ Ні	✗ Ні
Підтримка Knowledge Bases	✓ Вбудована	🛠 Вимагає конфігурації	🛠 Вимагає конфігурації

- Обрано керований сервіс AWS Bedrock Agents.
- Нативна інтеграція з Lambda (tool use).
- Вбудована підтримка Knowledge Bases.
- Здатність масштабування 'від нуля' (scale-to-zero).

База даних - DynamoDB

Критерій	DynamoDB	Aurora	MongoDB
Модель даних	NoSQL (K/V)	SQL (Relational)	NoSQL (Document)
Латентність (Latency)	Низька (1-5 мс)	Середня (10-50 мс)	Середня (5-20 мс)
Вартість (Базове навантаження)	~$0.50/міс (On-Demand)	~$20/міс	~$10/міс

- Обрано Amazon DynamoDB (On-Demand).
- Надзвичайно низька латентність (1-5 мс).
- Вартість ~$0.50/міс для базового навантаження.
- Модель ідеально підходить для збереження ChatHistory без складних JOIN-запитів.

Обґрунтування стека: Вибір базової LLM-моделі

Критерій	Claude 3.5 Haiku	GPT-4o mini	Gemini 1.5 Flash	Llama 3.1 8B
IFEval (Instruction Following)	88.7%	85.3%	83.1%	77.2%
Контекстне вікно (токени)	200,000	128,000	1,000,000	128,000
Нативна інтеграція AWS Bedrock	Так	Ні	Ні	Часткова

IFEval 88.7%: Здатність чітко слідувати системним інструкціям, що є критичним для точного роутингу.

200k Токенів: Дозволяє утримувати тривалі сесії зі складною історією діалогу та контекстом RAG.

Нативна Інтеграція: Розгортання агентів через Infrastructure as Code без додаткової мережної затримки.

Трирівнева безсерверна архітектура

Key Insight: Архітектура базується на принципах єдиної відповідальності та незалежного масштабування (14 окремих мікросервісів AWS Lambda).

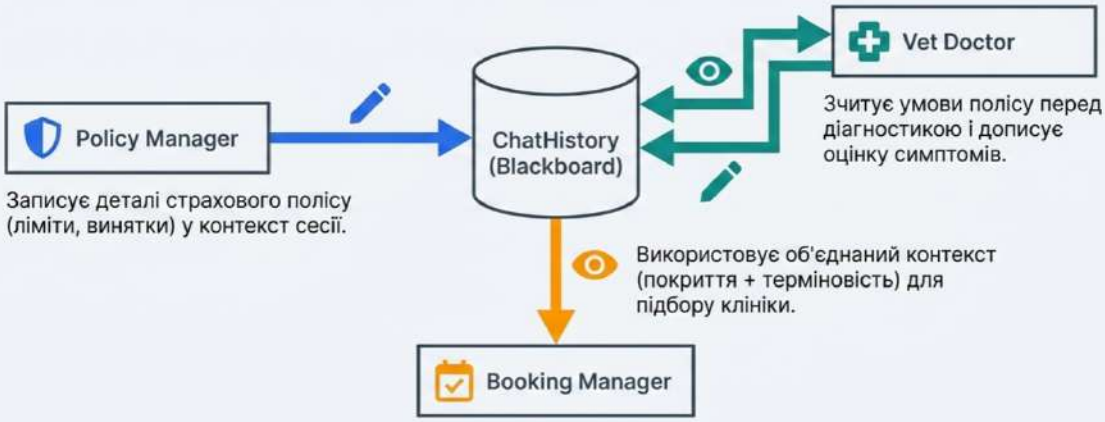
Розподіл інструментів та спеціалізація агентів

Інструменти (Action Groups)	Intention Classifier	Policy Manager	Vet Doctor	Booking Manager
GetPolicyDetails		✓		
Knowledge Base (RAG)			✓	
Find/Book Vet Clinic				✓
Save/Get Context		✓	✓	✓

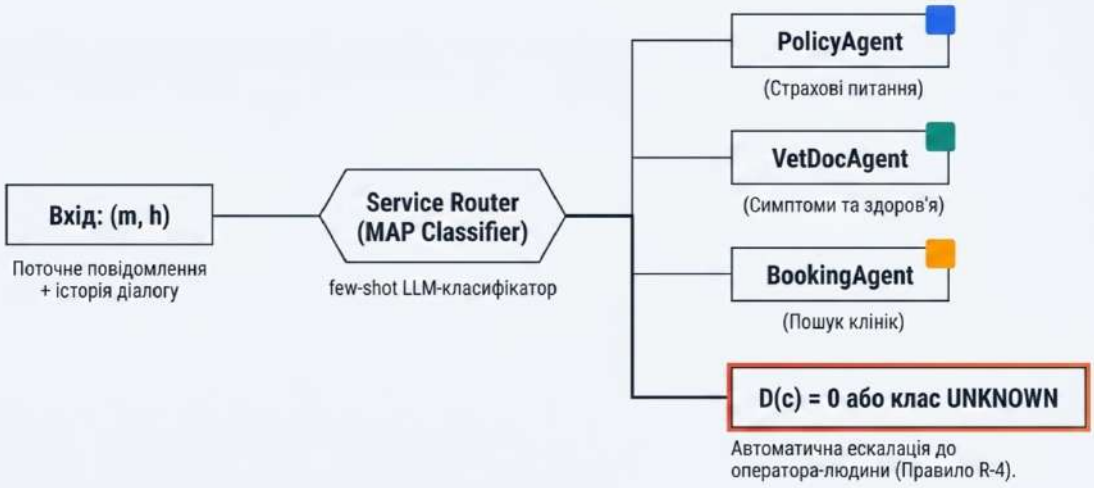
Агенти обмежені конкретними Action Groups. Цей підхід унеможливлює витік даних (data bleed) та блокує спроби маніпуляцій промптами (prompt injection) поза межами компетенції агента.

Патерн «Blackboard»: Асинхронний обмін контекстом

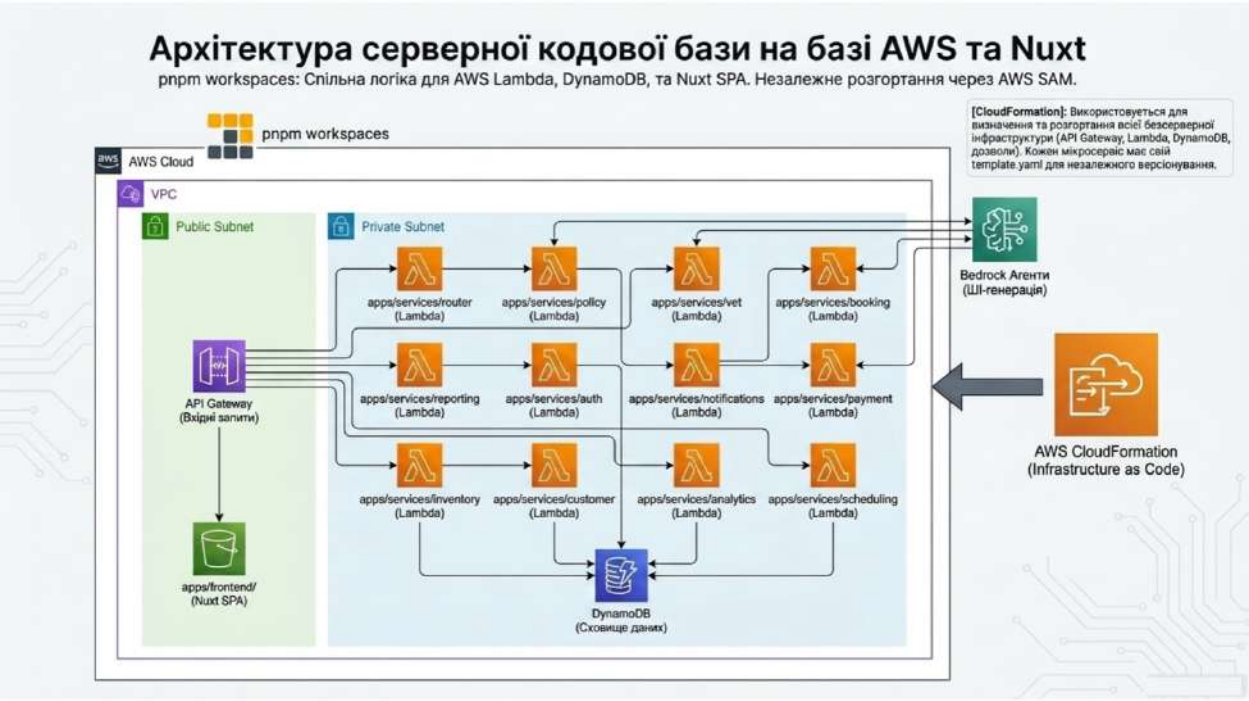
Доменні агенти НЕ викликають один одного напряму. Вони комунікують через єдине спільне сховище (Blackboard).



Алгоритм класифікації намірів та маршрутизації



Технічна архітектура RAG: Інтеграція з AWS Bedrock
Amazon Bedrock
Knowledge Bases
Claude 3.5 Haiku (LLM)
VetDocAgent (Симптоми та здоров'я)
Service Router (MAP Classifier)
API Gateway (Gateway)
Користувацький запит
Згенерована відповідь
Ветеринарні документи (PDF, DOCX)
Amazon S3 Vector Store
Сховище даних
Використання RAG на базі AWS Bedrock забезпечує заземлення на перевірені медичні знання, знижуючи ризик галюцинацій.


Архітектура серверної кодової бази на базі AWS та Nuxt
pnpm workspaces: Спільна логіка для AWS Lambda, DynamoDB, та Nuxt SPA. Незалежне розгортання через AWS SAM.
pnpm workspaces
[CloudFormation]: Використовується для визначення та розгортання всієї безсерверної інфраструктури (API Gateway, Lambda, DynamoDB, дозволи). Кожен мікросервіс має свій template.yaml для незалежного версіонування.
AWS Cloud
VPC
Public Subnet
Private Subnet
Bedrock Агенти (ШІ-генерація)
apps/services/router (Lambda)
apps/services/policy (Lambda)
apps/services/vet (Lambda)
apps/services/booking (Lambda)
API Gateway (Вхідні запити)
apps/services/reporting (Lambda)
apps/services/auth (Lambda)
apps/services/notifications (Lambda)
apps/services/payment (Lambda)
AWS CloudFormation (Infrastructure as Code)
apps/services/inventory (Lambda)
apps/services/customer (Lambda)
apps/services/analytics (Lambda)
apps/services/scheduling (Lambda)
apps/frontend/ (Nuxt SPA)
DynamoDB (Сховище даних)

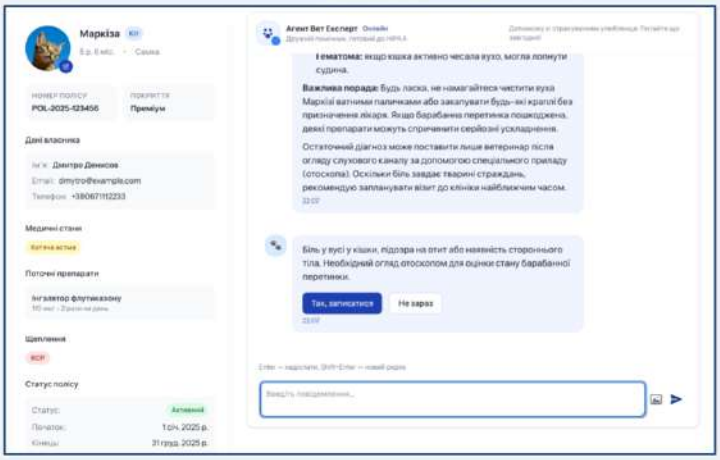
Синтез: Життєвий цикл запиту
КЛІЄНТСЬКА ЧАСТИНА (FRONTEND)
СЕРВЕРНА ЧАСТИНА (BACKEND) ТА AI
ДАНИ ТА СТРІМІНГ
Дія в Nuxt UI
API Gateway
Service Router
Bedrock Agent
Lambda Tool Execution
DynamoDB (Blackboard update)
Nuxt UI Render
Pinia State Update
SSE Stream

Структура Vue додатка: Компоненти, Маршрутизація та Стан
main.js / main.ts (Точка входу)
App.vue (Кореневий компонент)
Vue Router (Маршрутизатор)
routes: [...]
Pinia сховище (Стан)
stores: { user, agents, bookings }
Сторінки (Views)
HomeView.vue
AgentsView.vue
AboutView.vue
PoliciesView.vue
Сервіси / API (Дані)
api.js (HTTP запити)
auth.js
agents.js
bookings.js
Компоненти
Header.vue
NavBar.vue
Footer.vue
UserAvatar.vue
DataCard.vue
...

Клієнтський інтерфейс: Двоколонковий робочий простір

Прогресивне розкриття інформації: усі необхідні дані перед очима без перевантаження.

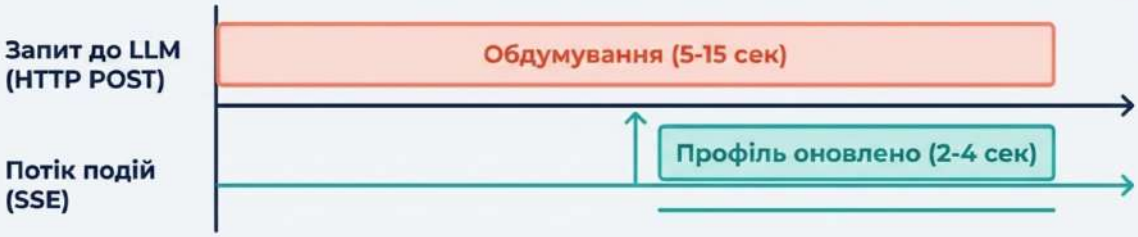
Nuxt 4 (ssr: false) | Vue 3 | Pinia



- **Ліва панель:** Автоматичне підтягування контексту (дані тварини, поліс). Агенти «бачать» це автоматично.
- **Права панель:** Діалогове вікно. Динамічний рендеринг компонентів (Markdown, інтерактивні кнопки).
- **Стан:** 19 Vue-компонентів зі сховищем Pinia для безшовної роботи в режимі SPA.

Асинхронний UX та Server-Sent Events (SSE)

Замість блокування інтерфейсу під час генерації відповіді ШІ, платформа використовує події, надіслані сервером (SSE).



Інструменти (Lambda) → Event Publisher → DynamoDB → SSE Stream

Бізнес-цінність: Користувач **миттєво** бачить, що система працює, отримуючи **візуальні індикатори** (оновлення профілю, кнопки дій) без очікування повної відповіді.

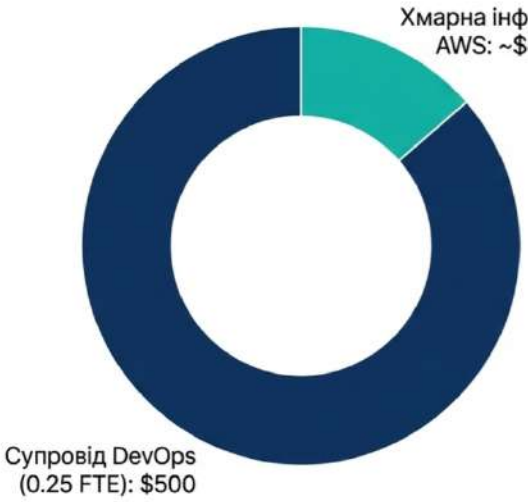
Принцип Human-in-the-Loop: Безпека медичних рішень

Жодна критична дія (медична чи адміністративна) не виконується без прямої згоди користувача.

Ветеринарний агент і агент бронювання не взаємодіють напряму. Клієнтський інтерфейс виступає посередником. Подія RecommendDoctorVisit генерує інтерактивну кнопку. Лише натискання «Так» (doctor_visit:confirmed) активує наступного агента.

Короткий огляд: Економічний потенціал платформи

Операційні витрати (OpEx): Serverless архітектура



92-96% витрат на хмарну інфраструктуру йде безпосередньо на токени Claude 3.5 Haiku.

Бази даних (DynamoDB) та обчислення (Lambda) коштують менше 5%.

Ми платимо виключно за згенеровану цінність.

Апробація дослідження: Участь у фахових конференціях

VII Міжнародну науково-технічну конференцію «Сучасний стан та перспективи розвитку IoT»

з поданням тез на тему «Сучасні інформаційні «Сучасні інформаційні технології в Україні і світі».

XI Міжнародна науково-практична конференція «Розбудова інноваційних економіки, менеджменту та освіти в умовах нової соціальної реальності»

з поданням тез на тему «IT-інновації, програмна інженерія та штучний інтелект у новій соціальній реальності.».

Висновок

Головний результат
повністю розроблено хмарну мультиагентну ШІ-платформу «ВетЕксперт».

Архітектура
14 мікросервісів в AWS, RAG-конвеєр для бази знань, спільний контекст між агентами (патерн blackboard).

Технічні показники
точність розпізнавання намірів — 95%; час відповіді (p95) — 9,34 с; понад 100 одночасних сесій без збоїв.

Економіка
скорочення часу обслуговування з 15-25 хв до 1-2 хв; зниження вартості звернення з $12,50 до $0,04.

Окупність
від 8,9 до 17,2 місяців (NPV за 3 роки — до $25 774).

Подальший розвиток
аналіз фотографій симптомів, розпізнавання голосу, новий агент для врегулювання збитків.