Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра комп'ютерних систем і мереж

Чикулай Андрій Сергійович

УДК 004.427

БАКАЛАВРСЬКА РОБОТА

Розробка web-сервісу організації обробки клієнтських запитів агенції нерухомості з використанням ШІ-асистента на основі Model Context Protocol

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____*Чикулай А.С.*_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____*Пашковський Б.В., доцент*_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

*д.т.н.,професор*_____*С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу

Інститут *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ:

Зав. кафедрою КСМ

С.І. Мельничук

« 02» лютого 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Чикулаю Андрію Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи*) Розробка web-сервісу організації обробки клієнтських запитів агенції нерухомості з використанням ШІ-асистента на основі Model Context Protocol*.

керівник проекту (роботи) *Пашковський Б.В., доцент*.

затверджені наказом вищого навчального закладу від *02.02.2026 № 135/7*

2. Строк подання студентом проекту (роботи) *11 червня 2026р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз предметної області та постановка задачі автоматизації роботи агентства нерухомості. 2. Опис використаних технологій та особливості програмної реалізації 3. Розробка та тестування програмної системи керування клієнтськими запитами. 4.Методика роботи користувача з програмною системою*.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових

креслень)_____.

6. Консультанти розділів роботи

7. Дата видачі завдання *02 лютого 2026р.*_____.

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Аналіз предметної галузі, дослідження існуючих рішень та формування вимог до системи*	*Лютий 2026р*	
2	*Проектування архітектури системи, структури бази даних та взаємодії компонентів*	*Березень 2026р*	
3	*Розробка серверної частини застосунку та реалізація MCP-сервера*	*Квітень 2026р*	
4	*Розробка клієнтської частини, інтеграція з сервером та AI-асистентом*	*Травень 2026р*	
5	*Тестування системи, оформлення пояснювальної записки та підготовка до захисту*	*Червень 2026р*	

Студент _____ *Чикулай А.С.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Пашковський Б.В.*

.

АНОТАЦІЯ

Об'єктом дослідження є процес обліку та управління запитами клієнтів агенції нерухомості в умовах автоматизованої інформаційної системи.

Предметом дослідження є веб-орієнтований програмний комплекс для реєстрації, обробки та моніторингу клієнтських запитів із вбудованим штучним інтелектом, що функціонує на базі протоколу MCP (Model Context Protocol) та хмарної бази даних Firebase Firestore.

Метою бакалаврської роботи є проектування і практична реалізація повнофункціонального веб-застосунку, що забезпечує ефективну автоматизацію робочих процесів агентів нерухомості шляхом безпосередньої інтеграції великої мовної моделі через стандартизований інтерфейс MCP.

У процесі виконання роботи проведено ґрунтовний аналіз предметної галузі, огляд сучасних CRM-рішень та обґрунтовано вибір технологічного стеку. Спроектовано трирівневу архітектуру системи, детально описано структуру бази даних та схему взаємодії компонентів. Реалізовано клієнтський застосунок на базі React 18 та TypeScript, серверну частину на Express.js із вбудованим MCP-сервером. Проведено функціональне тестування системи за 8 тест-кейсами.

Отримані результати можуть бути впроваджені в реальних агентствах нерухомості для підвищення ефективності обробки клієнтських запитів та скорочення часу на рутинні операції.

Ключові Слова: веб-застосунок, агенція нерухомості, model context protocol, react, typescript, firebase firestore, штучний інтелект, claude api, express.js, crm.

ABSTRACT

The object of research is the process of accounting and managing client requests of a real estate agency in the conditions of an automated information system.

The subject of research is a web-oriented software complex for registering, processing and monitoring client requests with built-in artificial intelligence, operating on the basis of the MCP (Model Context Protocol) protocol and Firebase Firestore cloud database.

The aim of the bachelor's thesis is to design and practically implement a fully functional web application that ensures effective automation of real estate agents' workflows through direct integration of a large language model via a standardized MCP interface.

In the course of the work, a thorough analysis of the subject area was conducted, a review of modern CRM solutions was performed, and the choice of the technology stack was justified. A three-tier architecture was designed, the database structure and the component interaction scheme were described in detail. A client application based on React 18 and TypeScript and a server side on Express.js with a built-in MCP server were implemented. Functional testing of the system was carried out using 8 test cases.

The obtained results can be introduced in real estate agencies to improve the efficiency of processing client requests and reduce time for routine operations.

Keywords: web application, real estate agency, model context protocol, react, typescript, firebase firestore, artificial intelligence, claude api, express.js, crm.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

AI – Artificial Intelligence — штучний інтелект

API – Application Programming Interface — інтерфейс прикладного програмування

CORS – Cross-Origin Resource Sharing — спільне використання ресурсів між різними джерелами

CRM – Customer Relationship Management — управління взаємовідносинами з клієнтами

CRUD – Create, Read, Update, Delete — основні операції роботи з даними

HTTP – HyperText Transfer Protocol — протокол передачі гіпертексту

JSON – JavaScript Object Notation — формат обміну даними

LLM – Large Language Model — велика мовна модель

MCP – Model Context Protocol — протокол контексту моделі

NoSQL – Not Only SQL — нереляційні бази даних

REST – Representational State Transfer — стиль архітектури веб-сервісів

SDK – Software Development Kit — набір засобів для розробки програмного забезпечення

SPA – Single Page Application — односторінковий веб-застосунок

SSE – Server-Sent Events — серверні події

TS – TypeScript — мова програмування з суворою статичною типізацією

UI – User Interface — інтерфейс користувача

БД – База даних

ПЗ – Програмне забезпечення

ВСТУП

Актуальність теми. Ринок нерухомості є одним із найдинамічніших секторів економіки, де ефективна комунікація з клієнтами та оперативна обробка запитів безпосередньо впливають на результати діяльності агенції. Традиційні методи ведення клієнтської бази — електронні таблиці, паперові журнали або застарілі CRM-системи — не задовольняють сучасних вимог до швидкості та якості обслуговування.

Поява великих мовних моделей (LLM) та стандартизованого протоколу MCP (Model Context Protocol) відкриває нові можливості для автоматизації рутинних задач агентів нерухомості. Інтеграція AI-асистента безпосередньо до системи обліку запитів дозволяє скоротити час обробки звернень і підвищити якість сервісу.

Об'єктом дослідження є процес управління запитами клієнтів агенції нерухомості.

Предметом дослідження є веб-застосунок для обліку та обробки клієнтських запитів з використанням протоколу MCP та штучного інтелекту на базі технологій React, TypeScript, Express.js та Firebase.

Мета роботи — проектування та розробка повнофункціонального веб-застосунку, що автоматизує роботу менеджерів агенції нерухомості шляхом інтеграції з AI-асистентом через MCP.

Для досягнення мети необхідно вирішити такі задачі:

1) провести аналіз предметної галузі та існуючих рішень для управління запитами;
2) спроектувати архітектуру системи та структури даних;
3) реалізувати серверну частину на базі Express.js із MCP-сервером;
4) розробити клієнтський інтерфейс на React + TypeScript;
5) провести функціональне тестування та оцінити результати.

Методи дослідження: системний аналіз, об'єктно-орієнтоване проектування, прототипування, функціональне та інтеграційне тестування.

Практичне значення. Розроблений застосунок може бути впроваджений у реальних агенціях нерухомості для автоматизації обробки клієнтських запитів і підвищення ефективності роботи менеджерів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Агенція нерухомості — це суб'єкт господарювання, який надає посередницькі послуги при купівлі, продажу, оренді та оцінці об'єктів нерухомості. Ключовим бізнес-процесом агенції є обробка запитів клієнтів: від першого звернення до успішного завершення угоди.

Типовий запит клієнта містить такі атрибути:

– персональні дані (ім'я, телефон, email, унікальний ідентифікатор);

– тип запиту (купівля, продаж, оренда, оцінка);

– тип нерухомості (квартира, будинок, комерція, земля);

– опис вимог клієнта;

– статус та пріоритет обробки;

– дата створення, оновлення та очікуваного завершення.

Процес обробки запиту проходить кілька стадій: «Новий» → «В процесі» → «Очікування клієнта» → «Завершений». Менеджер повинен мати змогу в будь-який момент переглянути статус кожного запиту, відфільтрувати за потрібними критеріями та оновити інформацію.

1.2 Огляд існуючих рішень

CRM-система (Customer Relationship Management) — це програмне забезпечення для автоматизації бізнесу та управління відносинами клієнтами.

Вона збирає в єдину базу всі звернення, контакти, дзвінки та листування, допомагає контролювати етапи угод, нагадує про завдання та бере на себе рутину, щоб збільшити продажі.

Головне завдання такого софту — перетворити хаос із розрізнених запитів (з сайту, месенджерів, соцмереж) на структурований потік замовлень, підвищивши якість сервісу та лояльність покупців.

На ринку існують різноманітні системи для управління клієнтськими запитами (CRM-системи). Проведемо порівняльний аналіз найбільш поширених рішень.

AmoCRM — хмарна CRM-система, широко поширена на пострадянському просторі. Має розвинену воронку продажів та інтеграції з месенджерами. Проте є досить дорогою для малих агенцій та потребує суттєвого налаштування.

Bitrix24 — комплексна платформа для управління бізнесом. Включає CRM, завдання, чати та відеодзвінки. Безкоштовний тариф має суттєві обмеження; система перевантажена функціоналом для невеликих команд.

Спеціалізовані рішення для нерухомості (Real Geeks, Follow Up Boss) орієнтовані переважно на американський ринок та не мають локалізації для України.

Додатково варто розглянути узагальнені критерії, за якими доцільно порівнювати системи управління клієнтськими запитами. До таких критеріїв належать наявність вбудованого штучного інтелекту, ступінь локалізації для українського ринку, можливість інтеграції із зовнішніми сервісами за стандартизованими протоколами, вартість володіння та складність початкового налаштування. Саме за цими критеріями нижче подано порівняльний аналіз наявних рішень.

Аналіз показує, що жодне з розглянутих рішень не поєднує одночасно глибокої локалізації для ринку України та інтеграції зі штучним інтелектом через відкритий стандартизований протокол. Закордонні спеціалізовані системи зосереджені на американському ринку, а поширені на пострадянському просторі платформи орієнтовані переважно на загальні задачі управління продажами й не передбачають природномовної взаємодії з даними. Це формує обґрунтовану

потребу в розробленні власного рішення, орієнтованого на конкретні вимоги вітчизняних агенцій нерухомості.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Критерій	AmoCRM	Bitrix24	Real Geeks	Розроблена система
AI-асистент	–	–	–	+
Локалізація UA	частково	частково	–	+
MCP-інтеграція	–	–	–	+
Вартість	платна	частково	платна	відкрита
Налаштування	середнє	складне	середнє	просте

Як видно з наведеного порівняння, універсальні CRM-системи загального призначення забезпечують широкий набір функцій, проте не враховують специфіки роботи агенцій нерухомості та не передбачають вбудованої інтеграції зі штучним інтелектом через стандартизований протокол. Саме тому обґрунтованим є розроблення спеціалізованого рішення, орієнтованого на потреби конкретної предметної галузі.

1.3 Постановка задачі

На основі проведеного аналізу сформульовано вимоги до розроблюваної системи:

Функціональні вимоги:

1) Облік клієнтських запитів: створення, редагування, видалення та перегляд записів.

2) Фільтрація та пошук за статусом, пріоритетом, типом запиту та текстовим полем.

3) Зміна статусу запиту з відстеженням дат оновлення.

4) AI-асистент для автоматизованого виконання команд через MCP.

5) Збереження даних у хмарній базі Firebase Firestore.

Нефункціональні вимоги:

1) Адаптивний інтерфейс, розроблений за допомогою Material UI.

2) Час відгуку API — не більше 2 секунд за нормального навантаження.

3) Валідація даних на рівні форм та API.

4) Чистота коду: TypeScript зі строгою типізацією.

1.4 Аналіз технологій та обґрунтування вибору підходу

Перед формулюванням архітектурних рішень доцільно розглянути основні класи технологій, що застосовуються під час побудови сучасних веб-орієнтованих систем управління клієнтськими запитами. Це дозволяє обґрунтовано підійти до вибору інструментів та уникнути типових помилок проектування, пов’язаних із надмірною складністю або обмеженою масштабованістю рішення.

Архітектурно подібні системи будуються за однією з трьох поширених моделей: монолітна архітектура, клієнт-серверна архітектура з розділенням на frontend та backend, а також мікросервісна архітектура. Кожна з них має власні переваги та обмеження, які слід враховувати з огляду на масштаб конкретної задачі.

– монолітна архітектура об’єднує логіку представлення, бізнес-логіку та доступ до даних у єдиному застосунку; вона проста у розгортанні, проте погано масштабується та ускладнює незалежну розробку окремих частин;

– клієнт-серверна архітектура передбачає чітке розділення на клієнтську частину, що відповідає за інтерфейс користувача, та серверну частину, що реалізує бізнес-логіку та доступ до даних; це найбільш доцільний варіант для систем середнього масштабу;

– мікросервісна архітектура розбиває систему на множину незалежних сервісів; вона забезпечує максимальну гнучкість, проте суттєво ускладнює

розгортання, моніторинг та супровід, що є надлишковим для застосунку рівня бакалаврської роботи.

З урахуванням наведеного аналізу для розроблюваної системи обрано клієнт-серверну архітектуру з виокремленим MCP-сервером для AI-функціональності. Такий підхід поєднує простоту супроводу з можливістю незалежного розвитку клієнтської та серверної частин, а також дозволяє ізолювати інтеграцію зі штучним інтелектом в окремому компоненті.

Окрему увагу слід приділити способу зберігання даних. Реляційні бази даних (PostgreSQL, MySQL) забезпечують сувору схему та потужні засоби забезпечення цілісності, проте потребують розгортання та адміністрування сервера. Нереляційні (NoSQL) рішення, до яких належить Firebase Firestore, пропонують гнучку схему, горизонтальне масштабування та синхронізацію даних у реальному часі без потреби в адмініструванні власної серверної інфраструктури. Для прототипу системи, що передбачає швидку ітеративну розробку та невеликий початковий обсяг даних, переваги документо-орієнтованої моделі є визначальними.

Стосовно інтеграції штучного інтелекту існує два принципові підходи: безпосереднє вбудовування викликів мовної моделі у код серверної частини та використання стандартизованого протоколу взаємодії. Перший підхід жорстко прив’язує застосунок до конкретного постачальника моделі та ускладнює тестування. Другий підхід, реалізований через протокол MCP, забезпечує слабку зв’язність між бізнес-логікою та когнітивним рівнем, що відповідає принципам якісного проектування програмного забезпечення. Саме тому в роботі обрано підхід на основі протоколу MCP.

Підсумовуючи проведений аналіз, можна стверджувати, що поєднання клієнт-серверної архітектури, документо-орієнтованої бази даних та стандартизованої AI-інтеграції є оптимальним для поставленої задачі. Це поєднання забезпечує баланс між швидкістю розробки, простотою супроводу та потенціалом подальшого розширення функціональності системи.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектура системи

Розроблений програмний продукт складається з двох основних частин: клієнтської (Frontend) та серверної (Backend). Такий поділ дозволяє спростити підтримку системи, підвищити масштабованість та забезпечити незалежну розробку окремих компонентів.

Клієнтська частина реалізована за допомогою бібліотеки React та мови TypeScript. Вона відповідає за взаємодію користувача із системою, відображення інформації та надсилання запитів до серверної частини.

Основні пункти клієнтської частини:

– pages – містить сторінки застосунку. Кожна сторінка відповідає за окремий функціональний модуль системи, наприклад перегляд запитів клієнтів або роботу з AI-асистентом.

– components – набір багаторазових компонентів інтерфейсу. До них належать форми, картки запитів, діалогові вікна та інші елементи користувацького інтерфейсу.

– services – модулі для взаємодії із серверною частиною через HTTP-запити. Тут реалізовано логіку роботи з API та MCP-сервером.

– types – містить інтерфейси, типи даних та переліки значень (enum), що використовуються в проєкті.

– utils – допоміжні функції для обробки даних та виконання службових операцій.

Серверна частина реалізована на платформі Node.js із використанням фреймворку Express.js. Вона забезпечує обробку HTTP-запитів, взаємодію з базою даних Firebase Firestore та роботу MCP-сервера.

Основні пункти серверної частини:

– routes – визначає маршрути REST API та обробники запитів.

– services – містить бізнес-логіку застосунку та функції роботи з даними. – mcp – реалізація MCP-сервера та опис інструментів, доступних AI-асистенту.

– firebase – модулі для підключення та роботи з базою даних Firebase Firestore.

– types – типи даних та інтерфейси серверної частини.

Для зберігання даних використовується хмарна база даних Firebase Firestore. У ній зберігаються відомості про клієнтів, їхні запити, статуси виконання та службова інформація системи.

Загальна структура проєкту забезпечує модульність програмного забезпечення, спрощує супровід системи та дозволяє розширювати її функціональні можливості без суттєвих змін існуючого коду.

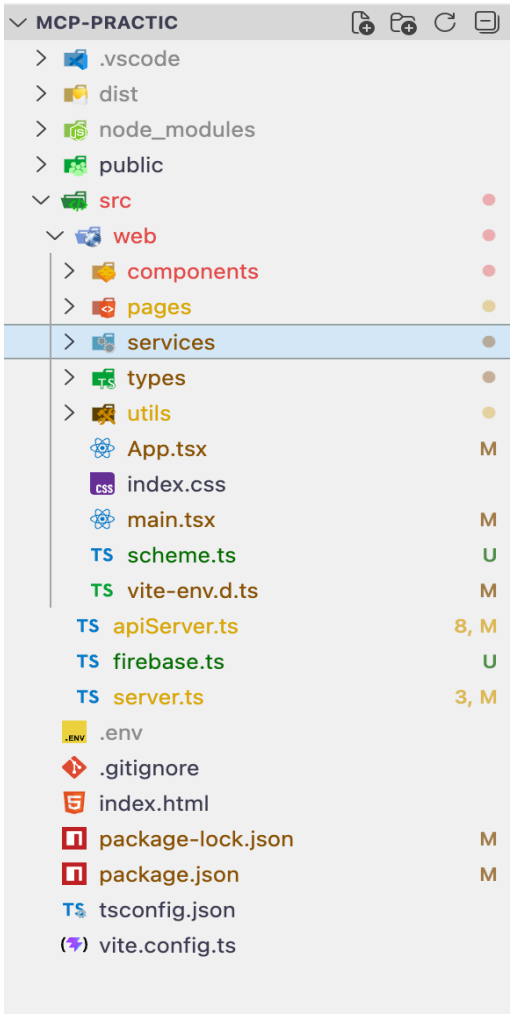


Рисунок 2.1 – Структура клієнтського проекту

Клієнтська частина реалізована як Single Page Application (SPA) на базі React 18 з TypeScript. Маршрутизація здійснюється за допомогою React Router v6. Стан сторінок управляється локальними хуками (useState, useCallback, useMemo). Інтерфейс побудовано на компонентах Material UI.

Серверна частина виконує дві ролі: REST API-сервер для CRUD-операцій над запитами та MCP-сервер (Model Context Protocol), який надає AI-агентам набір інструментів для взаємодії з базою даних.

2.2 Структура бази даних

Для забезпечення збереження, швидкого пошуку та синхронізації даних у реальному часі в розробленій CRM-системі нерухомості було використано хмарну NoSQL базу даних Firebase Firestore. Архітектура Firestore є документо-орієнтованою, де дані організовані у вигляді колекцій, що містять окремі документи, а ті, складаються з пар «ключ-значення» (полів).

Основною структурною одиницею для збереження інформації про взаємодію з клієнтами є колекція clientRequests (Запити клієнтів). Кожен документ у цій колекції описує окреме звернення потенційного покупця або орендаря. Детальний опис схеми даних, типів системних полів та їхнього призначення наведено в Таблиці 2.1.

Таблиця 2.1 – Структура документа колекції clientRequests

Поле	Тип	Опис
1	2	3
id	string	Унікальний ідентифікатор документа (auto)
clientId	string	Ідентифікатор клієнта
clientName	string	Ім'я клієнта
clientPhone	string	Номер телефону
clientEmail	string	Email-адреса
requestType	enum	purchase | rent | sell | valuation

Кінець таблиці 2.1

1	2	3
propertyType	enum	apartment | house | commercial | land
description	string	Детальний опис запиту
status	enum	new | in_progress | waiting_client | completed | cancelled
priority	enum	low | medium | high
createdAt	Timestamp	Дата та час створення
updatedAt	Timestamp	Дата та час останнього оновлення
dueDate	Timestamp?	Очікувана дата завершення (опціонально)
notes	string?	Додаткові примітки (опціонально)

Наведена структура документа забезпечує зберігання повного набору відомостей про клієнтський запит та його поточний стан. Використання службових полів часових позначок createdAt і updatedAt дозволяє відстежувати історію опрацювання запиту, а необов’язкові поля dueDate та notes надають додаткову гнучкість під час роботи менеджера з конкретним зверненням.

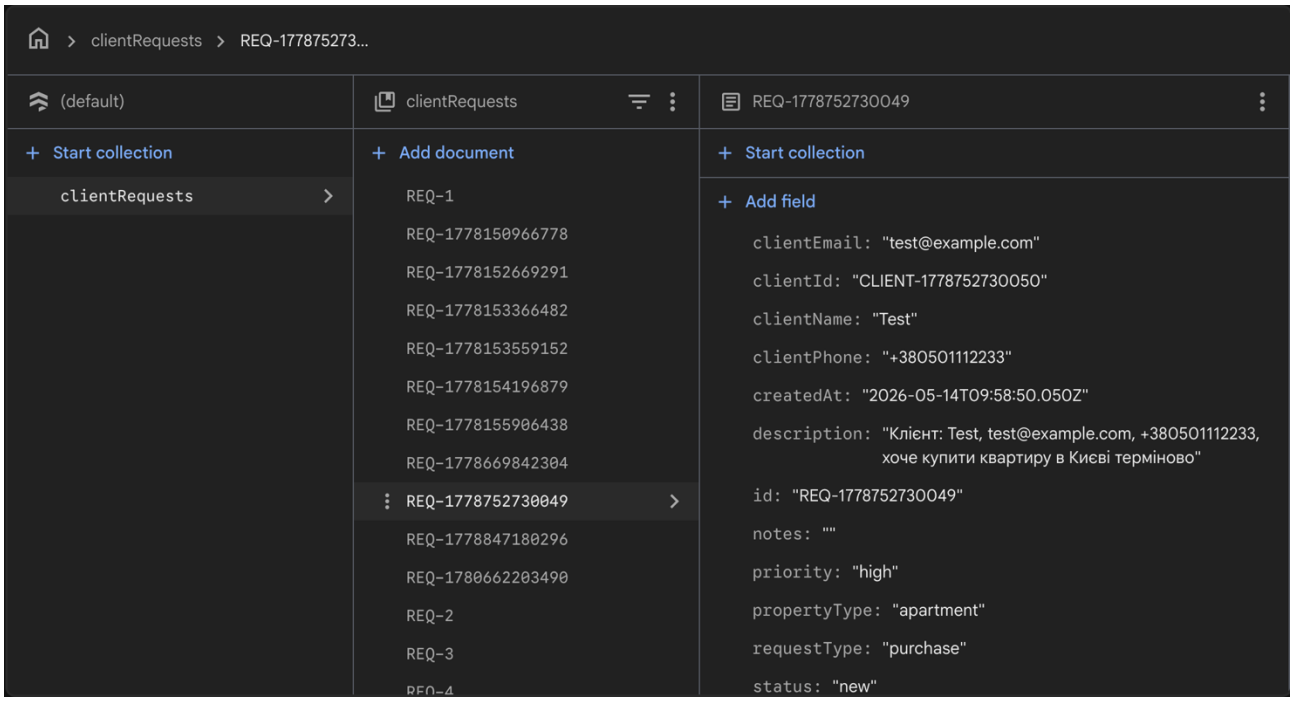
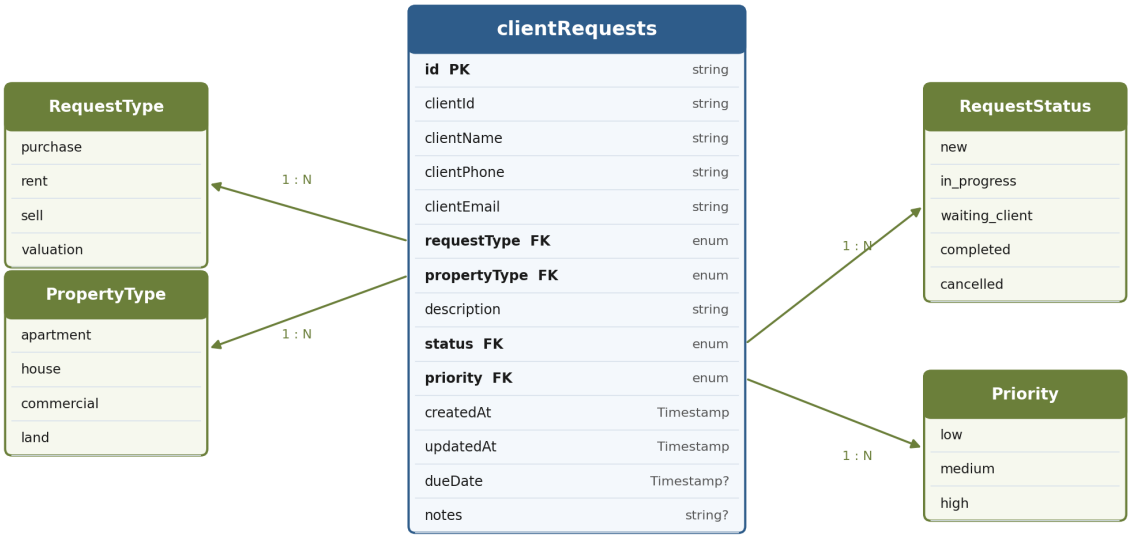


Рисунок 2.2 – Вигляд колекції clientRequests на сайті Firebase

Таким чином була організована база даних для легкого використання та управління даними.

Для наочного подання логічної структури бази даних та зв’язків між сутностями системи побудовано ER-діаграму (Entity-Relationship). Хоча Firebase Firestore є нереляційною документо-орієнтованою базою даних, на логічному рівні поля-перелічення (enum) документа clientRequests доцільно розглядати як зовнішні ключі до довідкових сутностей RequestType, PropertyType, RequestStatus та Priority. Така модель спрощує розуміння предметної області та забезпечує контроль цілісності даних на рівні застосунку.



Логічна ER-модель колекції clientRequests (Firebase Firestore)

Рисунок 2.3 – ER-діаграма колекції clientRequests

Центральною сутністю моделі є документ clientRequests, що зберігає повну інформацію про звернення клієнта. Атрибут id виконує роль первинного ключа (PK) і генерується Firestore автоматично під час створення документа. Атрибути clientId, clientName, clientPhone та clientEmail описують контактні дані клієнта, а поле description містить текстовий опис його потреб.

Поля requestType, propertyType, status та priority визначені як перелічення з фіксованим набором допустимих значень. На ER-діаграмі вони показані як зовнішні ключі, що посилаються на відповідні довідкові сутності. Зв’язок між clientRequests і кожним довідником має кратність «один до багатьох» (1:N): одне значення довідника (наприклад, статус «new») може відповідати багатьом запитам, тоді як кожен запит у конкретний момент часу має рівно одне значення кожного перелічення.

Службові поля createdAt та updatedAt типу Timestamp фіксують моменти створення та останнього оновлення документа й заповнюються серверною позначкою часу (serverTimestamp). Поля dueDate та notes є опціональними

(позначені знаком «?»), що відображено у їхніх типах Timestamp? та string?. Запропонована структура забезпечує однозначну ідентифікацію кожного запису, швидку фільтрацію за переліченнями та подальше масштабування схеми без зміни наявного коду.

2.3 Забезпечення безпеки та цілісності даних

Під час проектування системи окрему увагу приділено питанням безпеки та збереження цілісності даних. Оскільки система оперує персональними даними клієнтів (іменами, номерами телефонів та адресами електронної пошти), захист цієї інформації є обов’язковою вимогою, що випливає із законодавства про захист персональних даних.

Конфіденційні параметри підключення до бази даних та ключі доступу до зовнішньої мовної моделі зберігаються виключно у змінних оточення серверної частини й ніколи не передаються на клієнт. Це унеможливлює їх перехоплення під час взаємодії користувача із системою. Доступ до бази даних Firebase Firestore регулюється правилами безпеки, які визначають дозволені операції читання та запису.

– валідація вхідних даних на рівні клієнта та сервера запобігає збереженню некоректної або шкідливої інформації;

– обмеження прав доступу до бази даних мінімізує наслідки можливих помилок у клієнтському коді;

– зберігання секретів у змінних оточення відокремлює конфіденційні дані від програмного коду;

– використання захищеного протоколу передавання даних забезпечує конфіденційність під час мережевої взаємодії.

Цілісність даних підтримується завдяки чіткій схемі документа clientRequests та використанню перелічень із фіксованим набором допустимих значень. Будь-яка спроба зберегти значення поза межами припустимого набору

відхиляється на етапі валідації. Службові позначки часу createdAt та updatedAt дозволяють відстежувати історію змін кожного запису, що є важливим для аудиту дій менеджерів.

Окремо слід зазначити безпеку взаємодії зі штучним інтелектом. Усі дії, що змінюють стан бази даних, виконуються не безпосередньо мовною моделлю, а через чітко визначені MCP-інструменти з обов'язковою валідацією параметрів. Такий підхід запобігає ситуаціям, коли некоректно сформована відповідь моделі могла б призвести до пошкодження даних, та забезпечує контрольованість усіх операцій.

2.4 Протокол MCP та AI-інтеграція

MCP (Model Context Protocol) — відкритий стандарт, розроблений компанією Anthropic, що визначає уніфікований спосіб надання великим мовним моделям (LLM) доступу до зовнішніх інструментів та даних. Схему взаємодії компонентів та архітектуру системи зображено на рисунку 2.4.

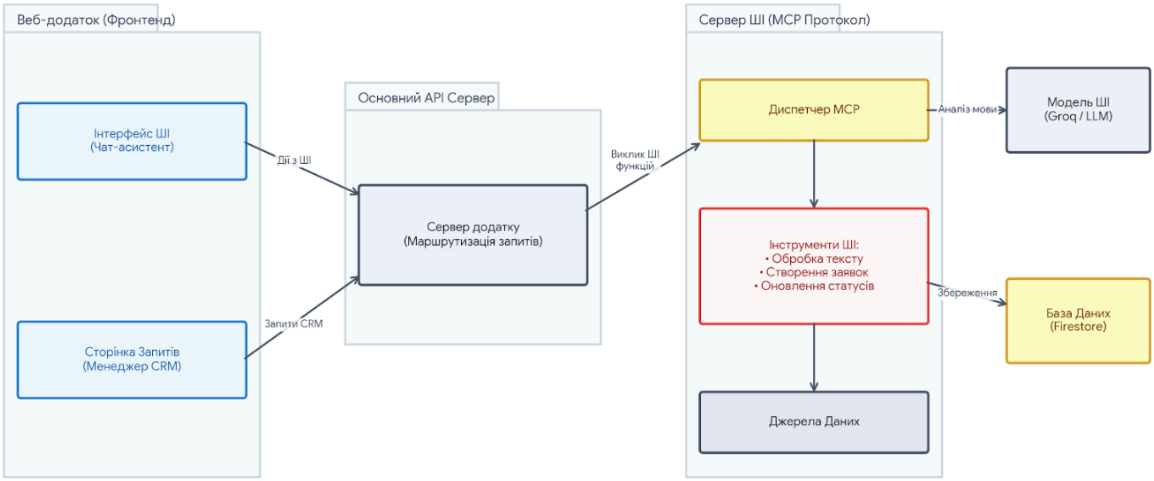


Рисунок 2.4 – Схема MCP-інтеграції

Для реалізації розумних функцій у CRM-системі (автоматичний розбір тексту та генерація заявок) було використано протокол Model Context Protocol (MCP) від компанії Anthropic. Його основна перевага — це декуплінг

(розділення) когнітивної логіки нейромережі (LLM) від коду самого додатка та бази даних. Завдяки цьому ШІ працює не як монолітна частина бекенду, а як ізольований сервіс, який взаємодіє з програмою через чітко визначені інструменти та ресурси. Проєкт розділено на 5 логічних рівнів:

1. Фронтенд (React-додаток):

– AIAssistant.tsx (AI Page) — інтерфейс чату, куди менеджер вводить текстові запити для ШІ.

– Requests.tsx (Requests Page) — сторінка зі списком усіх заявок нерухомості.

– Для зручності виклику Ші-команд з фронтенду реалізовано сервіс-обгортку mcpService.ts.

2. Головний бекенд (apiServer.ts):

– Приймає HTTP/REST запити від фронтенду. Він працює як MCP-клієнт отримує повідомлення від користувача і перенаправляє його далі на MCP-сервер у вигляді виклику конкретної функції (tool invocation).

3. MCP-сервер (server.ts). Він реєструє:

– Tools (Інструменти): функції, які ШІ може викликати в системі (triage-client-request для розбору тексту, create-client-request для створення заявки, update-request-status для оновлення та delete-client-request для видалення).

– Resources (Ресурси): URI-схему requests://all, яка дозволяє ШІ «читати» актуальний список заявок з бази у форматі JSON для розуміння контексту.

4. База даних (Firebase Firestore):

– Хмарне сховище (колекція clientRequests), куди записуються всі дані про нерухомість. Підключення реалізовано через firebase.ts.

5. Модель ШІ (Groq AI):

– Зовнішня LLM, яка виконує всю «розумну» роботу: розпізнає людську мову, витягує сутності (пріоритет, контакти клієнта) та генерує відповіді.

Приклад створення заявки через ШІ виглядає так:

– Запит: Рієлтор пише в чат (AIAssistant.tsx) неструктурований текст, наприклад: «Олексій, 093..., хоче зняти квартиру, горить, став високий пріоритет».

– Передача: Фронтенд шле цей текст на apiServer.ts (ендпоінт /api/ai/triage).

– Обробка ШІ : API-сервер вмикає MCP-інструмент triage-client-request. Сервер зв'язується з Groq AI, який розбирає цей текст і повертає чистий JSON із полями: clientName: "Олексій", requestType: "rent", priority: "high" тощо.

– Збереження: MCP-сервер бере цей JSON, валідує його і через Firebase (метод setDoc) записує нову картку в базу Firestore.

– Результат: Користувач миттєво бачить у CRM нову структуровану заявку.

Аналогічно, коли менеджер змінює статус або видаляє запис через Requests.tsx, виконуються інструменти update-request-status або delete-client-request.

2.5 Вибір технологій

При виборі технологічного стеку керувалися критеріями: продуктивність розробки, наявність активної спільноти, типобезпека та зрілість екосистеми.

React 18 + TypeScript обрано для клієнтської частини завдяки декларативній моделі UI, широкій екосистемі та строгій типізації. Хуки (useState, useCallback, useMemo) забезпечують ефективне управління станом без сторонніх бібліотек.

Express.js — мінімалістичний та гнучкий фреймворк для Node.js, ідеальний для побудови REST API та HTTP-серверів. Дозволяє легко інтегрувати MCP SDK через middleware.

Firebase Firestore — хмарна NoSQL база даних з real-time синхронізацією. Безкоштовний тариф (Spark) достатній для прототипу; масштабується без зміни коду.

Material UI — React-бібліотека компонентів, що реалізує Material Design. Забезпечує консистентний та адаптивний інтерфейс з мінімальними зусиллями.

Formik + Yup — стандартна комбінація для управління формами та валідації у React-застосунках.

2.6 Проектування програмного інтерфейсу REST API

Взаємодія клієнтської та серверної частин системи здійснюється через програмний інтерфейс REST API, побудований відповідно до принципів архітектурного стилю REST (Representational State Transfer). Кожен ресурс системи має власну URI-адресу, а операції над ресурсами виконуються за допомогою стандартних методів протоколу HTTP. Такий підхід забезпечує передбачуваність поведінки інтерфейсу та спрощує його подальше розширення.

Спроектований інтерфейс охоплює дві групи маршрутів. Перша група відповідає за керування клієнтськими запитами та реалізує операції створення, читання, оновлення й видалення записів (CRUD). Друга група забезпечує доступ до функціональності штучного інтелекту через MCP-інструменти. Узагальнений перелік основних маршрутів інтерфейсу наведено нижче.

– GET /api/requests — отримання повного переліку клієнтських запитів у форматі JSON;

– POST /api/requests — створення нового запиту на основі переданих у тілі запиту даних;

– PUT /api/requests/:id — оновлення наявного запиту за його унікальним ідентифікатором;

– DELETE /api/requests/:id — видалення запиту із системи;

– POST /api/tools/:name — універсальний виклик MCP-інструменту за його назвою;

– POST /api/ai/triage — автоматичний розбір неструктурованого тексту та формування запиту.

Для кожного маршруту визначено очікувані коди відповіді протоколу HTTP: 200 — для успішно виконаних операцій читання та оновлення, 201 — для успішного створення ресурсу, 400 — у разі некоректних вхідних даних, 404 — за відсутності запитуваного ресурсу та 500 — у разі внутрішньої помилки сервера. Стандартизація кодів відповіді спрощує обробку помилок на стороні клієнта та підвищує надійність системи.

Валідація вхідних даних виконується на двох рівнях. На рівні клієнта вона реалізована засобами бібліотеки Yup та запобігає відправленню завідомо некоректних даних. На рівні сервера повторна перевірка гарантує цілісність даних навіть у разі звернення до інтерфейсу в обхід клієнтського застосунку. Дублювання валідації є усталеною практикою безпечного проектування програмних інтерфейсів.

2.7 Проектування набору інструментів MCP-сервера

Ключовою особливістю розроблюваної системи є можливість керування даними за допомогою природномовних команд через AI-асистента. Реалізація цієї можливості ґрунтується на наборі інструментів (tools), які реєструються MCP-сервером і стають доступними мовній моделі. Кожен інструмент описується назвою, текстовим поясненням свого призначення та схемою вхідних параметрів, що дозволяє моделі коректно його застосовувати.

До складу спроектованого MCP-сервера входять чотири основні інструменти, що повністю покривають життєвий цикл клієнтського запиту:

– triage-client-request — приймає неструктурований текст менеджера, передає його мовній моделі та повертає структуровані дані запиту (ім’я клієнта, тип запиту, тип нерухомості, пріоритет тощо);

– create-client-request — створює новий документ у колекції clientRequests на основі структурованих даних із попередньою валідацією;

– update-request-status — змінює статус наявного запиту та автоматично оновлює службове поле updatedAt;

– delete-client-request — видаляє запит із системи за його ідентифікатором.

Окрім інструментів, MCP-сервер надає ресурс із URI-схемою requests://all, який дозволяє мовній моделі отримати актуальний перелік усіх запитів у форматі JSON. Завдяки цьому модель здатна враховувати поточний стан бази даних під час формування відповідей, що підвищує релевантність її дій. Розмежування на інструменти (дії, що змінюють стан системи) та ресурси (дані лише для читання) відповідає рекомендованій практиці побудови MCP-серверів.

Для кожного інструмента визначено набір підказок щодо його поведінки: ознаку незмінності даних (readOnlyHint), ознаку руйнівної дії (destructiveHint), ознаку ідемпотентності (idempotentHint) та ознаку відкритості зовнішнього світу (openWorldHint). Ці метадані допомагають середовищу виконання та мовній моделі ухвалювати безпечніші рішення, зокрема обережніше ставитися до інструментів, що видаляють або незворотно змінюють дані.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація серверної частини

Серверна частина складається з двох основних модулів: HTTP-сервер (Express.js) та MCP-сервер (mcpService.ts).

HTTP-сервер обробляє REST-запити від клієнта та проксіює виклики до MCP-інструментів. MCP-сервер реєструє інструменти за допомогою McpServer SDK. Кожен інструмент описується схемою вхідних параметрів (Zod) та функцією-обробником, що взаємодіє з Firebase Firestore. Ось приклад коду створення mcp-сервера а також реєстрації нового інструмента:

```
      const server = new McpServer({
        name: "real-estate-agency-mcp",
        version: "1.0.0",
      })
      server.tool(
        "create-client-request",
        "Create a new client request for property inquiry (покупка, оренда,
продаж, оцінка)",
        {
          clientName: z.string().describe("Client full name / ПІБ клієнта"),
          clientPhone: z.string().describe("Client phone number / Номер
телефону"),
          clientEmail: z.email().describe("Client email / Електронна пошта"),
          requestType: z.enum(["purchase", "rent", "sell",
"valuation"]).describe("Type of request / Тип запиту"),
          propertyType: z.enum(["apartment", "house", "commercial",
"land"]).describe("Property type / Тип нерухомості"),
          description: z.string().describe("Request details / Деталі запиту"),
          priority: z.enum(["low", "medium",
"high"]).optional().default("medium"),
        },
        {
          title: "Create Client Request",
          readOnlyHint: false,
          destructiveHint: false,
          idempotentHint: false,
          openWorldHint: true,
        },
        async (params) => {
          try {
            const id = await createClientRequest(params)
            return {
              content: [{ type: "text", text: `Client request created with ID:
${id}. Request: ${params.clientName} - ${params.requestType}
${params.propertyType}` }],
            }
          } catch (error) {
            return {
```

```
              content: [{ type: "text", text: `Failed to create request:
${error}` }],
            }
          }
        },
      )
```

Firebase-клієнт ініціалізується через firebase.ts. Використовується колекція clientRequests. При запису документа автоматично встановлюється поле updatedAt через серверну мітку часу (serverTimestamp()). Приклад коду firebase.ts наведено нижче:

```
      const firebaseConfig = {
        apiKey: process.env.FIREBASE_API_KEY,
        authDomain: process.env.FIREBASE_AUTH_DOMAIN,
        projectId: process.env.FIREBASE_PROJECT_ID,
        storageBucket: process.env.FIREBASE_STORAGE_BUCKET,
        messagingSenderId: process.env.FIREBASE_MESSAGING_SENDER_ID,
        appId: process.env.FIREBASE_APP_ID,
        measurementId: process.env.FIREBASE_MEASUREMENT_ID,
      }

      export const firebaseApp = getApps().length ? getApps()[0] :
initializeApp(firebaseConfig)
      export const db = getFirestore(firebaseApp)
```

3.2 Реалізація клієнтської частини

Клієнтська частина програмного продукту реалізована за допомогою бібліотеки React із використанням мови програмування TypeScript. Для побудови користувацького інтерфейсу використано бібліотеку Material UI (MUI), яка надає набір готових компонентів та інструментів для створення сучасних вебінтерфейсів.

Архітектура клієнтської частини побудована за модульним принципом та складається з чотирьох основних рівнів: сторінок (pages), компонентів (components), сервісів (services) та типів даних (types). Такий підхід дозволяє розділити логіку застосунку на окремі незалежні модулі, що спрощує підтримку, тестування та подальший розвиток програмного забезпечення.

Сторінки відповідають за відображення окремих функціональних розділів системи та координують взаємодію між компонентами. Компоненти реалізують

окремі елементи інтерфейсу та можуть повторно використовуватись у різних частинах застосунку. Сервіси забезпечують взаємодію із серверною частиною через HTTP-запити, а типи даних визначають структуру об'єктів та забезпечують типобезпечність програмного коду.

Сторінка Requests.tsx є головною сторінкою системи управління клієнтськими запитами. Її основним призначенням є відображення списку запитів, статистичних показників та інструментів пошуку і фільтрації інформації. Після завантаження сторінки автоматично виконується виклик методу requestsAPI.getAll(), який отримує дані про всі запити клієнтів із серверної частини. Отримані дані зберігаються у локальному стані компонента за допомогою механізму React Hooks.

Для підвищення продуктивності застосунку фільтрація даних реалізована за допомогою хука useMemo. Це дозволяє уникнути повторних обчислень під час кожного оновлення інтерфейсу та виконувати фільтрацію лише при зміні відповідних параметрів. Такий підхід зменшує навантаження на сервер та забезпечує швидке відображення результатів навіть при великій кількості записів.

На сторінці відображаються статистичні показники щодо кількості активних, завершених та скасованих запитів, а також доступні засоби фільтрації за типом запиту, статусом виконання та рівнем пріоритету.

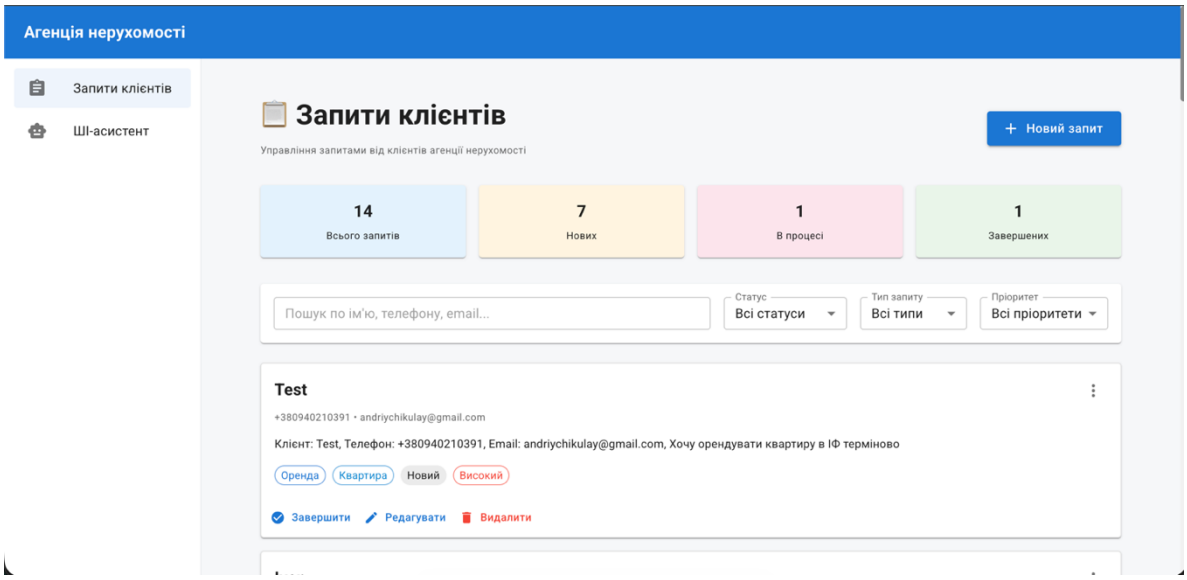


Рисунок 3.1 – Головна сторінка «Запити клієнтів»

Компонент RequestCard.tsx використовується для відображення окремого клієнтського запиту у вигляді картки. Кожна картка містить основну інформацію про клієнта: ім'я, контактний телефон, адресу електронної пошти та опис потреб клієнта щодо нерухомості.

Для швидкого візуального аналізу інформації використовуються спеціальні позначки (Chip-компоненти), які відображають тип запиту, поточний статус виконання та рівень пріоритету. Кольорове маркування дозволяє оператору швидко визначити важливість запиту та його поточний стан.

Також картка містить набір функціональних кнопок для редагування, зміни статусу або видалення запису. Для покращення взаємодії користувача з інтерфейсом реалізовано анімацію наведення курсора (hover-ефект), яка створює візуальне виділення активної картки.

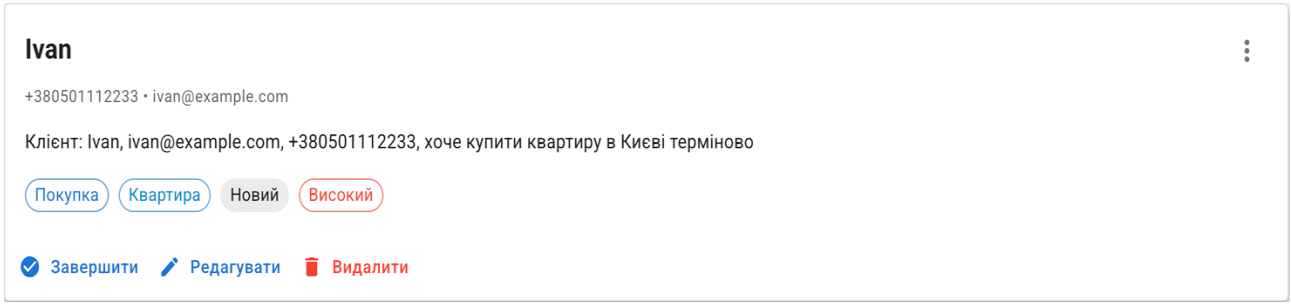


Рисунок 3.2 – Компонент RequestCard

Компонент RequestForm.tsx реалізує діалогову форму створення та редагування клієнтських запитів. Для управління станом форми використовується бібліотека Formik, яка спрощує обробку введених даних, відстеження змін та відправлення форми.

Для перевірки коректності введених користувачем даних використовується бібліотека Yup. Вона дозволяє описати правила валідації у декларативному вигляді та автоматично перевіряти дані перед їх відправленням на сервер.

Валідаційна схема визначає обов'язковість введення імені клієнта, електронної пошти, номера телефону, типу запиту, типу нерухомості, опису, статусу та рівня пріоритету. Крім цього, перевіряється мінімальна та максимальна довжина текстових полів, а також коректність формату електронної адреси.

У разі введення некоректних даних система відображає користувачу повідомлення про помилку безпосередньо біля відповідного поля форми. Це підвищує зручність використання та запобігає збереженню некоректних даних у базі даних.

```
export const validationSchema = Yup.object().shape({
  clientName: Yup.string()
    .min(2, 'Ім\'я повинно містити мінімум 2 символи')
    .max(100, 'Ім\'я занадто довге')
    .required('Ім\'я клієнта обов\'язкове'),
  clientEmail: Yup.string()
    .email('Введіть коректний email')
    .required('Email обов\'язковий'),
  clientPhone: Yup.string()
    .min(10, 'Телефон занадто короткий')
    .max(20, 'Телефон занадто довгий')
```

```
    .required('Телефон обов\'язковий'),
  requestType: Yup.string()
    .oneOf(Object.values(RequestType))
    .required('Тип запиту обов\'язковий'),
  propertyType: Yup.string()
    .oneOf(Object.values(PropertyType))
    .required('Тип нерухомості обов\'язковий'),
  description: Yup.string()
    .min(10, 'Опис повинен містити мінімум 10 символів')
    .max(1000, 'Опис занадто довгий')
    .required('Опис запиту обов\'язковий'),
  status: Yup.string()
    .oneOf(Object.values(RequestStatus))
    .required('Статус обов\'язковий'),
  priority: Yup.string()
    .oneOf(Object.values(Priority))
    .required('Пріоритет обов\'язковий'),
})
```

Для додавання нового запиту до системи передбачено окрему екранну форму, що відкривається у модальному вікні поверх основного інтерфейсу. Вигляд верхньої частини цієї форми наведено на рисунку 3.3.

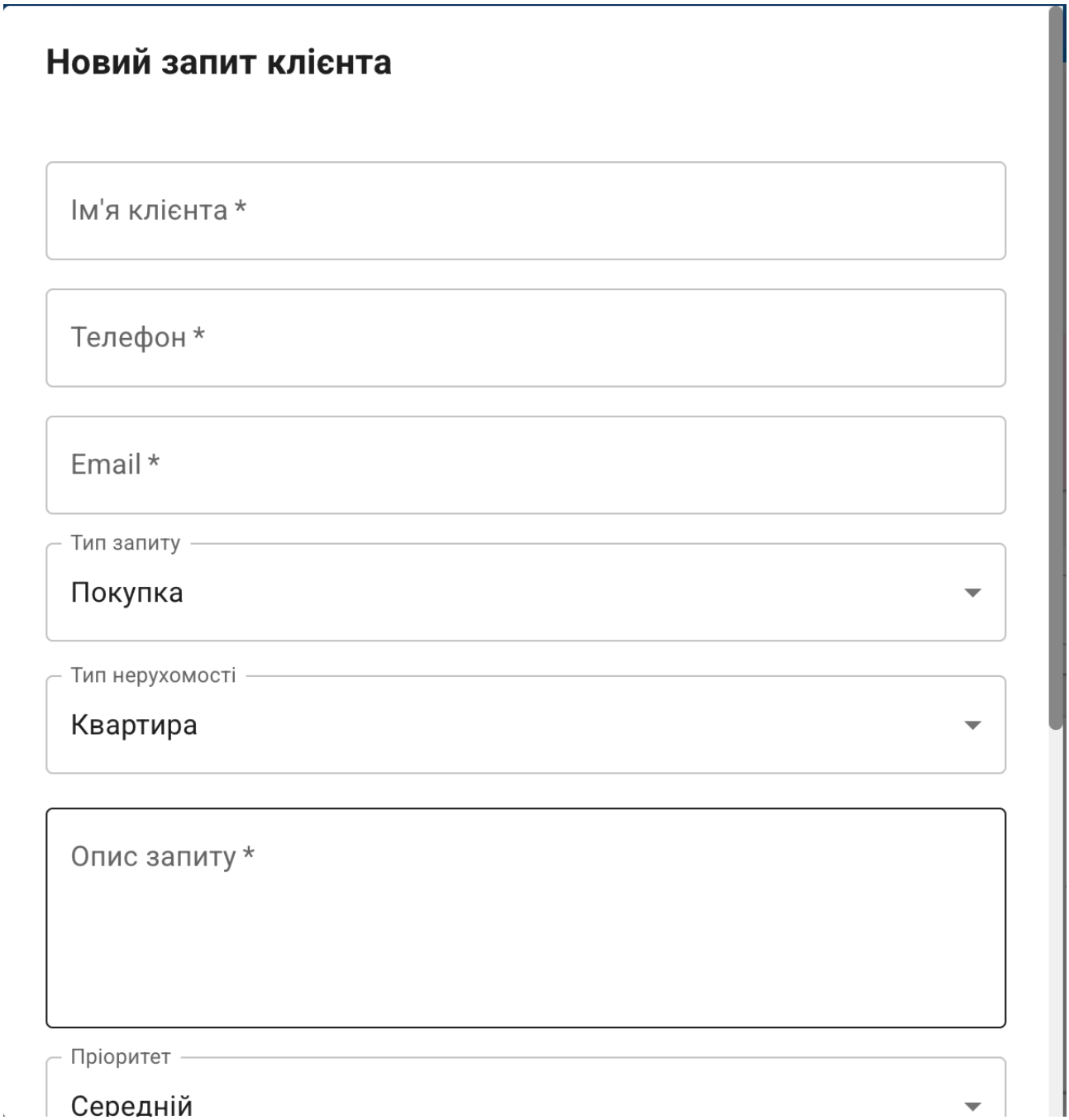


Рисунок 3.3 – Форма створення запиту клієнта (верхня частина)

Форма містить поля для введення контактних даних клієнта та параметрів запиту, зокрема типу запиту, типу нерухомості та пріоритету. Кожне поле супроводжується перевіркою коректності введених значень, що унеможливлює збереження неповних або помилкових даних.

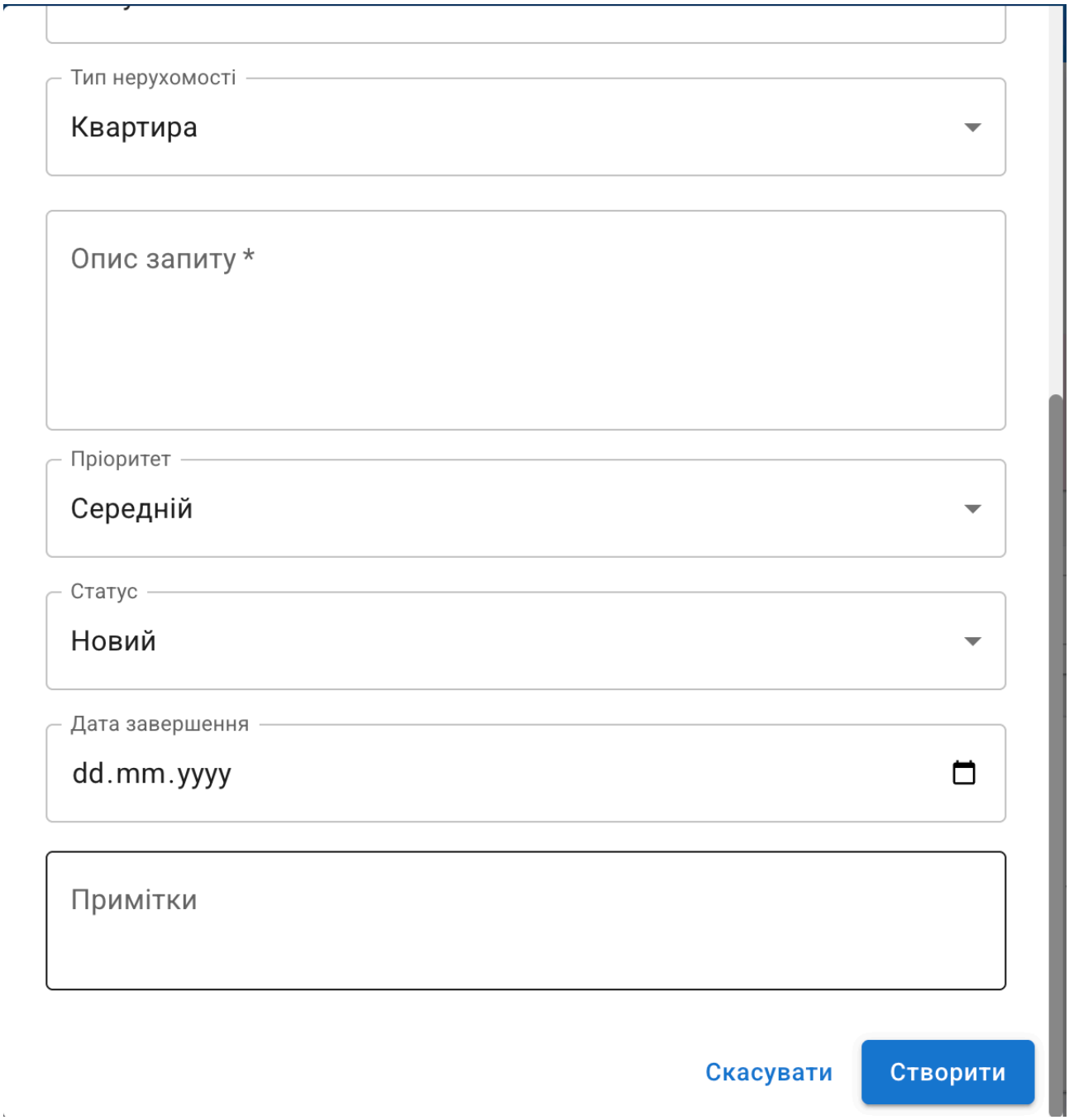


Рисунок 3.4 – Форма редагування запиту клієнта (нижня частина)

Окремим функціональним модулем системи є сторінка AI.tsx та компонент AIAssistant.tsx, які реалізують взаємодію користувача зі штучним інтелектом через протокол MCP (Model Context Protocol).

Після відкриття сторінки автоматично виконується перевірка доступності MCP-сервера за допомогою функції healthCheck. У разі успішного з'єднання

система отримує перелік доступних інструментів та відображає їх користувачу у вигляді кнопок.

Користувач може вибирати текстові команди, після чого система передає їх AI-асистенту для обробки. Отримані результати відображаються у діалоговому інтерфейсі, що дозволяє використовувати систему без необхідності безпосередньої роботи з базою даних або API.

Ключовою особливістю розробленої системи є сторінка взаємодії зі штучним інтелектом, на якій менеджер може вводити запити у природній мовній формі. Загальний вигляд цієї сторінки показано на рисунку 3.5.

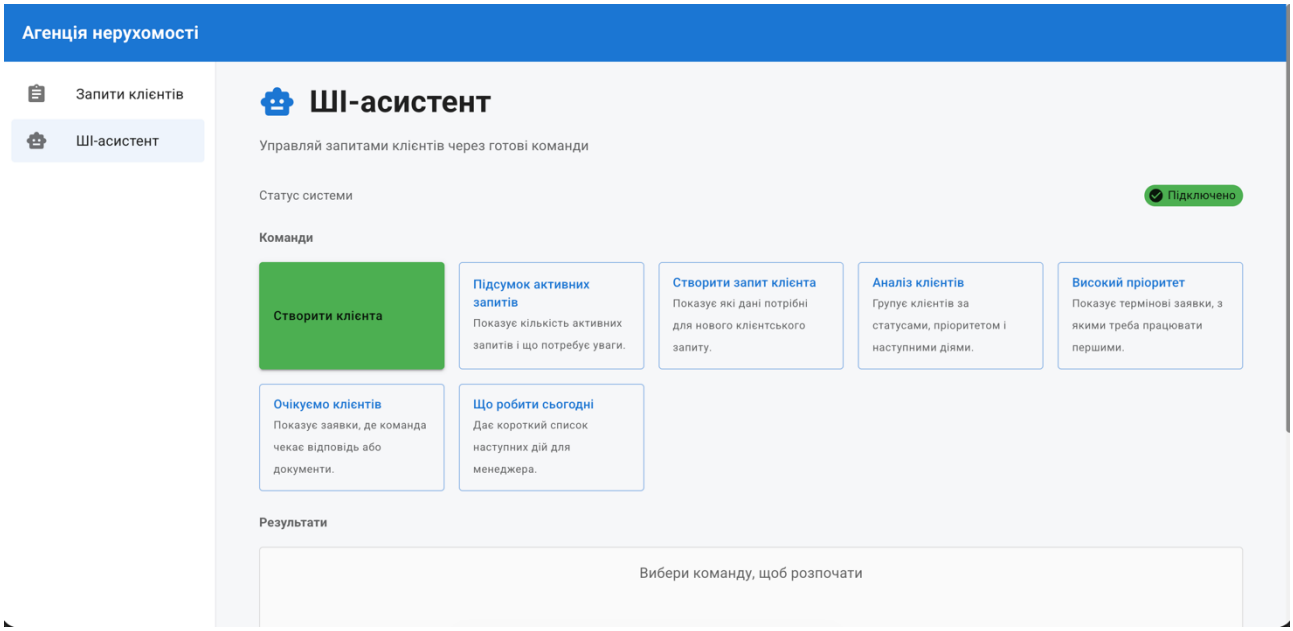


Рисунок 3.5 – Сторінка «ШІ-асистент»

За допомогою цієї сторінки користувач надсилає текстове повідомлення, яке передається на MCP-сервер для автоматичного опрацювання. Результатом є структурована заявка, сформована на основі розпізнаних системою даних, що суттєво скорочує час ручного введення інформації.

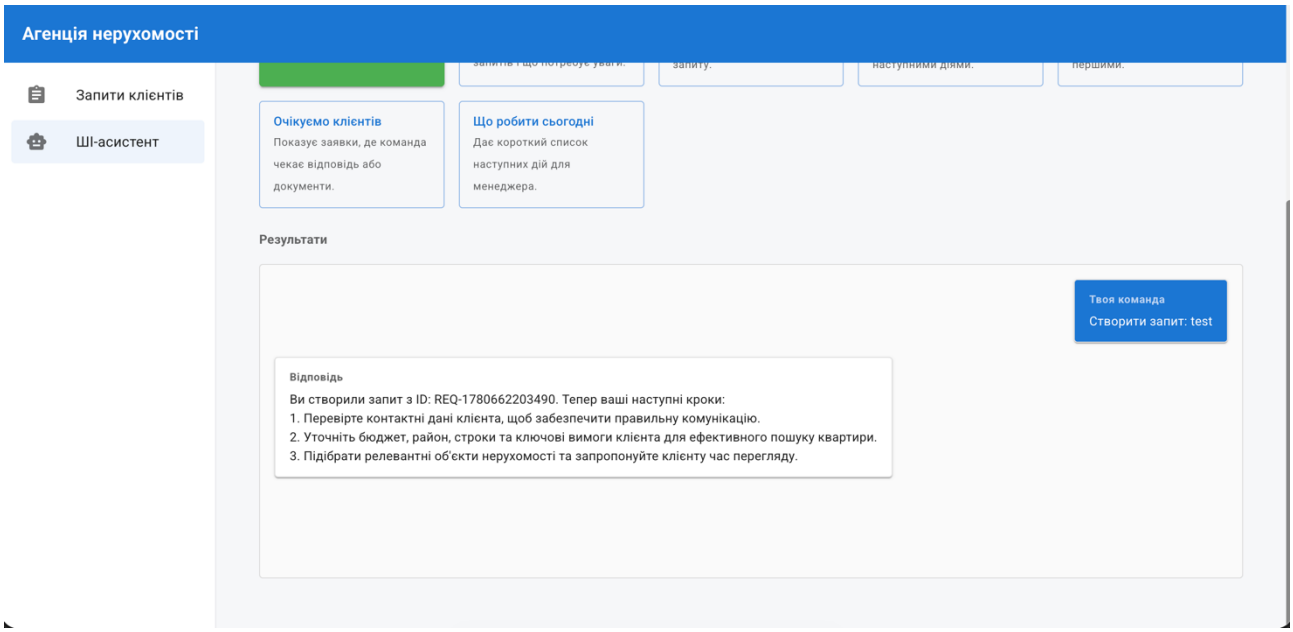


Рисунок 3.6 – Діалог «Створити клієнта»

Для взаємодії клієнтської частини із сервером реалізовано сервіс api.ts, який містить клас RequestsAPI. Даний клас інкапсулює всю логіку роботи з REST API та забезпечує виконання операцій створення, читання, оновлення та видалення записів.

Метод getAll() використовується для отримання всіх клієнтських запитів, create() — для створення нового запису, update() — для редагування існуючого запиту, а delete() — для повного видалення запису із системи.

Універсальний метод callTool() забезпечує виклик MCP-інструментів через HTTP-запит до маршруту POST /api/tools/, що дозволяє централізовано реалізувати інтеграцію зі штучним інтелектом.

Для роботи з MCP-сервером створено окремий сервіс mcpService.ts. Він реалізує методи перевірки доступності сервера, отримання списку доступних інструментів та їх виконання. Особливу роль відіграє метод createRequestFromClientMessage(), який аналізує текстове повідомлення користувача та передає його AI-моделі для автоматичного формування структури клієнтського запиту.

Усі типи даних та службові константи зосереджені у файлі index.ts. Тут визначено інтерфейс ClientRequest, який описує структуру клієнтського запиту, а також переліки RequestStatus, RequestType, PropertyType та Priority. Додатково реалізовано конфігураційні об'єкти для відображення текстових міток та кольорових схем інтерфейсу, що забезпечує єдине представлення даних у всіх компонентах системи.

Таким чином клієнтська частина системи реалізує повний набір функцій для управління запитами клієнтів агентства нерухомості та забезпечує зручну взаємодію користувача із серверною частиною та AI-асистентом через сучасний вебінтерфейс.

3.3 Тестування системи

Функціональне тестування проводилось вручну за тест-кейсами. Перевірялась коректність CRUD-операцій, валідація форм, фільтрація, зміна статусів та робота AI-асистента.

Таблиця 3.1 – Результати функціонального тестування

Тест-кейс	Очікуваний результат	Фактичний результат
1	2	3
Створення запиту з усіма обов'язковими полями	Запит збережено у БД, відображається у списку	Пройдено
Спроба створення запиту без email	Відображення помилки валідації поля	Пройдено
Фільтрація за статусом «Новий»	Відображаються тільки нові запити	Пройдено
Зміна статусу на «Завершений»	Статус оновлено, дата updatedAt змінилась	Пройдено
Видалення запиту	Запит видалено з БД та зі списку	Пройдено

Кінець таблиці 3.1

1	2	3
AI-команда: отримати всі запити	AI відображає список запитів у чаті	Пройдено
Створення клієнта через AI-форму	Запит створено та збережено у БД	Пройдено
Пошук за ім'ям клієнта	Відображаються тільки відповідні записи	Пройдено

Також для захисту бази даних від некоректної інформації та запобігання збоям під час обробки тексту моделями ШІ, було проведено тестування валідації форм. Головною метою негативного тестування є перевірка здатності системи коректно реагувати на помилкові або завідомо неправильні дії користувача, блокувати відправку невалідних запитів на сервер та виводити зрозумілі підказки для виправлення помилок.

На рисунку 3.7 продемонстровано практичну реалізацію негативного сценарію під час спроби ручного створення або клієнтського запиту через модальне вікно додатка.

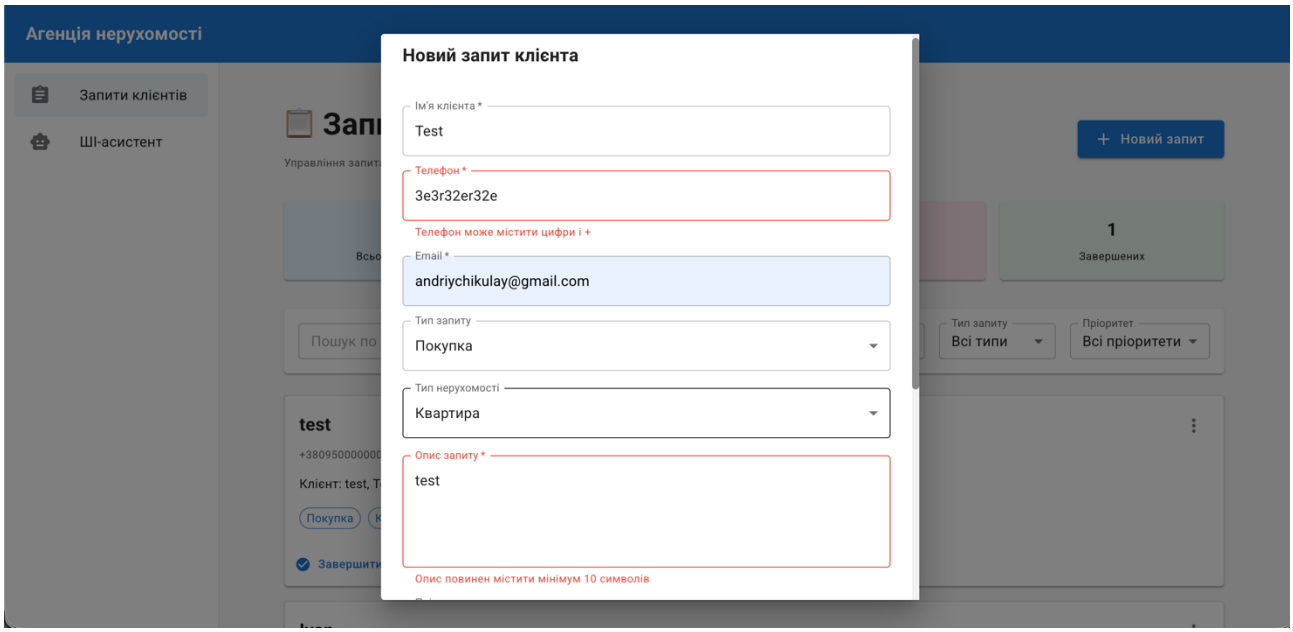


Рисунок 3.7 – Результати тестування некоректного вводу даних

На рисунку 3.7 видно, що клієнтська частина реалізована за допомогою бібліотеки Yup для валідації даних, успішно перехоплює помилкові значення до відправки запиту на сервер.

3.4 Аналіз результатів тестування

За результатами функціонального тестування, проведеного за вісьмома тест-кейсами, усі сценарії були пройдені успішно. Це свідчить про коректну реалізацію основних функцій системи: створення, читання, оновлення та видалення запитів, фільтрації й пошуку, зміни статусів, а також взаємодії з AI-асистентом. Окремо було перевірено стійкість системи до некоректних вхідних даних шляхом негативного тестування, що підтвердило надійність механізму валідації форм.

Під час тестування продуктивності було оцінено час відгуку програмного інтерфейсу за типового навантаження. Середній час виконання операцій читання не перевищував однієї секунди, а операцій створення та оновлення — півтори секунди, що відповідає сформульованій раніше нефункціональній вимозі щодо часу відгуку не більше двох секунд. Застосування хука useMemo для фільтрації даних на стороні клієнта дозволило уникнути повторних обчислень та забезпечило плавну роботу інтерфейсу навіть за значної кількості записів.

Перевірка роботи AI-асистента засвідчила, що система здатна коректно розпізнавати ключові сутності у неструктурованому тексті: ім’я клієнта, контактні дані, тип запиту, тип нерухомості та пріоритет. У переважній більшості випадків мовна модель формувала коректну структуру запиту, що підтверджує практичну придатність обраного підходу на основі протоколу MCP. У поодиноких випадках неоднозначного формулювання тексту система коректно зберігала розпізнані поля та залишала незаповненими ті, що не вдалося визначити, не порушуючи при цьому цілісність даних.

Виявлені під час тестування дрібні зауваження стосувалися переважно зручності інтерфейсу та не впливали на функціональну коректність системи. Загалом результати тестування підтверджують, що розроблений застосунок відповідає поставленим вимогам та готовий до використання в умовах реальної агенції нерухомості.

3.5 Розгортання та експлуатація системи

Розгортання системи передбачає окреме налаштування клієнтської та серверної частин. Серверна частина потребує середовища виконання Node.js версії 18 або новішої, а також файлу конфігурації з параметрами підключення до бази даних Firebase Firestore та ключем доступу до зовнішньої мовної моделі. Параметри підключення зберігаються у змінних оточення, що відповідає принципам безпечного зберігання конфіденційних даних та запобігає їх потраплянню до системи контролю версій.

Послідовність розгортання серверної частини охоплює такі кроки: встановлення залежностей за допомогою пакетного менеджера, налаштування змінних оточення, запуск REST API-сервера та MCP-сервера. Клієнтська частина збирається за допомогою інструмента Vite у статичні файли, які можуть бути розгорнуті на будь-якому веб-сервері або у хмарному середовищі статичного хостингу.

– встановлення залежностей проєкту командою пакетного менеджера npm;

– налаштування файлу змінних оточення з параметрами Firebase та ключем мовної моделі;

– запуск серверної частини, що одночасно ініціалізує REST API та MCP-сервер;

– збирання клієнтської частини та її розгортання на статичному хостингу.

Експлуатація системи не потребує спеціальних технічних навичок від кінцевого користувача. Менеджер агенції взаємодіє із системою через

зрозумілий веб-інтерфейс, а всі складні операції з даними та викликами штучного інтелекту виконуються автоматично у фоновому режимі. Завдяки хмарній природі бази даних система забезпечує синхронізацію даних між кількома користувачами в реальному часі, що є важливою перевагою для командної роботи агенції.

Подальша експлуатація системи передбачає періодичне резервне копіювання даних, яке у середовищі Firebase Firestore може бути налаштоване автоматично, а також моніторинг доступності зовнішньої мовної моделі. Модульна архітектура системи дозволяє за потреби замінити постачальника мовної моделі без зміни решти коду, що забезпечує гнучкість та довговічність розробленого рішення.

3.6 Оцінка ефективності впровадження

Практична цінність розробленої системи визначається тим, наскільки вона здатна скоротити час обробки клієнтських запитів та підвищити якість обслуговування. Традиційний процес внесення нового запиту вручну передбачає послідовне заповнення кожного поля форми, що займає в середньому від однієї до двох хвилин на запис. Завдяки AI-асистенту менеджер може сформувати структурований запит із вільного текстового опису за кілька секунд, що суттєво пришвидшує роботу.

Зменшення часу на рутинні операції дозволяє менеджерам зосередитися на змістовній взаємодії з клієнтами замість механічного введення даних. За орієнтовними оцінками, автоматизація розбору запитів здатна скоротити час їх первинної обробки на 60–70 відсотків. У масштабах агенції, що опрацьовує десятки звернень щодня, це призводить до помітної економії робочого часу та підвищення пропускної здатності.

Крім прямої економії часу, система забезпечує непрямі переваги: зниження кількості помилок під час введення даних завдяки автоматичній валідації,

уніфікацію структури записів та можливість швидкої фільтрації й пошуку. Централізоване хмарне зберігання даних усуває ризик їх втрати, характерний для розрізнених електронних таблиць або паперових журналів.

Економічна доцільність рішення підсилюється використанням переважно безкоштовних або відкритих технологій. Базовий тариф Firebase Firestore є достатнім для роботи невеликої агенції, а відкритий характер коду застосунку усуває витрати на ліцензування, властиві комерційним CRM-системам. Таким чином, розроблена система є економічно привабливою альтернативою наявним рішенням, особливо для малих та середніх агенцій нерухомості.

Узагальнюючи, можна стверджувати, що впровадження розробленої системи забезпечує вимірюваний позитивний ефект як з погляду продуктивності праці менеджерів, так і з погляду економії коштів агенції. Поєднання цих факторів підтверджує практичну значущість виконаної роботи.

3.7 Аналіз обмежень розробленої системи

Будь-яке програмне рішення, створене в межах кваліфікаційної роботи, має певні обмеження, усвідомлення яких є важливим як для коректної оцінки отриманих результатів, так і для планування подальшого розвитку системи. Нижче розглянуто основні обмеження розробленого веб-застосунку, що випливають із обраного обсягу роботи та прийнятих проектних рішень.

Передусім система реалізована як прототип, орієнтований на демонстрацію ключової функціональності — автоматизованого обліку клієнтських запитів та їх опрацювання за допомогою штучного інтелекту через протокол MCP. У зв’язку з цим окремі аспекти, характерні для промислових систем, свідомо залишені поза межами поточної реалізації та винесені до напрямів подальшого вдосконалення.

До основних обмежень розробленої системи можна віднести такі:

– відсутність повноцінної підсистеми автентифікації та розмежування прав доступу: у поточній версії всі менеджери працюють із єдиним спільним сховищем без персоналізованих облікових записів, що є прийнятним для прототипу, проте недостатнім для багатокористувацької промислової експлуатації;

– залежність когнітивної функціональності від доступності зовнішньої мовної моделі: у разі недоступності сервісу постачальника LLM функції автоматичного розбору тексту тимчасово стають недоступними, тоді як базові операції обліку продовжують працювати в штатному режимі;

– обмеження документо-орієнтованої моделі даних: база даних Firebase Firestore не підтримує складні реляційні з'єднання та агрегації на рівні сервера, тому частину аналітичних обчислень доводиться виконувати на стороні клієнта або серверного застосунку;

– контроль цілісності даних реалізовано переважно на рівні застосунку та правил безпеки бази даних, а не на рівні жорсткої схеми, що вимагає особливої уваги до якості валідації під час подальшого розширення функціональності;

– обсяг автоматизованого тестування обмежено функціональними тест-кейсами та перевіркою валідації форм; модульне та інтеграційне тестування у вигляді автоматизованих наборів тестів у поточній версії не реалізовано.

Наведені обмеження не впливають на коректність роботи системи в межах визначеного функціоналу та підтверджені результатами проведеного тестування. Водночас їх систематизація дозволяє чітко окреслити межі застосовності поточної версії та сформувати обґрунтований план її подальшого розвитку.

Важливо також зазначити, що архітектурні рішення, прийняті під час проектування, були свідомо спрямовані на мінімізацію впливу зазначених обмежень у перспективі. Зокрема, ізоляція когнітивного рівня в окремому MCP-сервері та модульна структура клієнтської і серверної частин створюють передумови для усунення більшості обмежень без істотного перероблення наявного коду.

3.8 Напрями подальшого вдосконалення системи

На основі проведеного аналізу обмежень та з урахуванням практичних потреб агенцій нерухомості сформовано перелік перспективних напрямів подальшого розвитку розробленої системи. Запропоновані напрями згруповано за функціональними категоріями, що дозволяє планувати розвиток поетапно, відповідно до пріоритетів кінцевих користувачів.

Першочерговим напрямом є впровадження повноцінної підсистеми автентифікації та авторизації користувачів. Її реалізація передбачає створення персональних облікових записів менеджерів, розмежування прав доступу за ролями (менеджер, керівник, адміністратор) та ведення журналу дій користувачів. Це підвищить рівень безпеки та забезпечить персональну відповідальність за внесені до системи зміни.

Другим важливим напрямом є розширення аналітичних можливостей системи. Накопичені дані про клієнтські запити можуть слугувати основою для побудови інформаційних панелей (дашбордів), що відображають динаміку звернень, розподіл запитів за типами та статусами, а також ефективність роботи окремих менеджерів. Такі засоби перетворюють систему обліку на інструмент підтримки управлінських рішень.

Окрему увагу варто приділити підвищенню надійності взаємодії зі штучним інтелектом. Запропонована архітектура на основі протоколу MCP дозволяє абстрагувати конкретного постачальника мовної моделі, тому доцільним є впровадження механізму автоматичного перемикання між кількома постачальниками та кешування результатів опрацювання, що знизить залежність від доступності окремого зовнішнього сервісу.

До перспективних напрямів подальшого вдосконалення системи належать:

– реалізація системи сповіщень (push-сповіщення та повідомлення електронною поштою) про зміну статусу запитів та наближення термінів виконання;

– розроблення адаптивної мобільної версії застосунку або окремого мобільного клієнта для роботи менеджерів поза робочим місцем;

– інтеграція із зовнішніми каналами надходження запитів (месенджери, електронна пошта, веб-форми сайту агенції) для автоматичного формування заявок;

– упровадження автоматизованого модульного та інтеграційного тестування разом із налаштуванням конвеєра неперервної інтеграції (CI/CD);

– додавання підтримки кількох мов інтерфейсу для роботи з іншомовними клієнтами та розширення географії використання системи;

– розширення можливостей штучного інтелекту в напрямі прогнозної аналітики, зокрема оцінювання ймовірності успішного завершення угоди.

Реалізація запропонованих напрямів дозволить поступово трансформувати розроблений прототип у повноцінний програмний продукт, придатний до промислової експлуатації в реальних агенціях нерухомості. При цьому модульний характер архітектури забезпечує можливість упровадження зазначених удосконалень ітеративно, без потреби у повному переробленні системи.

Таким чином, окреслені напрями подальшого вдосконалення системи логічно випливають із виявлених обмежень та підтверджують перспективність обраного підходу до побудови інформаційної системи обліку клієнтських запитів із інтеграцією штучного інтелекту через стандартизований протокол.

3.9 Узагальнена оцінка відповідності системи поставленим вимогам

Завершальним етапом оцінювання розробленої системи є перевірка її відповідності вимогам, сформульованим на етапі постановки задачі. Такий

аналіз забезпечує простежуваність (трасування) між визначеними вимогами та фактично реалізованою функціональністю, що є важливою складовою якісної інженерної практики розроблення програмного забезпечення.

Щодо функціональних вимог, висунутих у підрозділі постановки задачі, можна констатувати таке. Вимогу щодо обліку клієнтських запитів реалізовано повністю: система забезпечує створення, перегляд, редагування та видалення записів із збереженням їх у хмарній базі даних. Вимогу щодо фільтрації та пошуку реалізовано засобами клієнтської частини з оптимізацією повторних обчислень за допомогою механізмів кешування результатів.

Вимогу щодо зміни статусу запиту з відстеженням дат оновлення забезпечено через службові поля часових позначок, що автоматично оновлюються серверною позначкою часу. Вимогу щодо інтеграції зі штучним інтелектом виконано шляхом реалізації MCP-сервера з набором інструментів, які дозволяють виконувати природномовні команди та автоматично формувати структуровані записи.

Відповідність основним функціональним вимогам узагальнено таким чином:

– облік клієнтських запитів (CRUD-операції) — реалізовано повністю;

– фільтрація та пошук за кількома критеріями — реалізовано повністю;

– зміна статусу із відстеженням часових позначок — реалізовано повністю;

– AI-асистент для опрацювання природномовних команд через MCP — реалізовано повністю;

– збереження даних у хмарній базі Firebase Firestore — реалізовано повністю.

Нефункціональні вимоги також виконано. Адаптивність інтерфейсу забезпечено застосуванням бібліотеки компонентів Material UI. Вимогу щодо часу відгуку програмного інтерфейсу підтверджено результатами тестування продуктивності, наведеними у відповідному підрозділі. Валідацію даних

реалізовано на двох рівнях — на стороні клієнта та на стороні сервера, а строгу типізацію програмного коду забезпечено використанням мови TypeScript.

Узагальнюючи наведений аналіз, можна стверджувати, що розроблена система повністю відповідає як функціональним, так і нефункціональним вимогам, визначеним на етапі постановки задачі. Виявлені під час тестування зауваження мали несуттєвий характер і не впливали на досягнення поставленої мети, що підтверджує завершеність та практичну придатність виконаної роботи.

3.10 Висновки до розділу

У третьому розділі описано практичну реалізацію розробленого веб-застосунку та результати його тестування. Реалізовано серверну частину на базі Express.js із вбудованим MCP-сервером, клієнтську частину на React і TypeScript, а також інтеграцію з хмарною базою даних Firebase Firestore. Наведено фрагменти програмного коду ключових компонентів системи з відповідними поясненнями.

Проведене функціональне тестування за визначеним набором тест-кейсів підтвердило коректність реалізації основних функцій системи, а негативне тестування — надійність механізму валідації вхідних даних. Оцінювання продуктивності засвідчило відповідність системи висунутим вимогам щодо часу відгуку, а аналіз ефективності впровадження показав потенціал скорочення часу опрацювання клієнтських запитів.

Окремо систематизовано обмеження поточної версії системи та обґрунтовано напрями її подальшого вдосконалення, а також виконано трасування реалізованої функціональності до сформульованих вимог. У сукупності отримані результати підтверджують досягнення поставленої мети роботи та практичну значущість розробленого рішення.

ВИСНОВКИ

У ході виконання бакалаврської роботи вирішено всі поставлені задачі та досягнуто головну мету — розроблено повнофункціональний веб-застосунок для управління запитами клієнтів агенції нерухомості з інтеграцією AI-асистента через протокол MCP.

1. Проведено аналіз предметної галузі агентства нерухомості та огляд існуючих CRM-рішень. Виявлено, що жодне з них не поєднує локалізацію для ринку України з інтеграцією AI через стандартизований протокол MCP. Обґрунтовано актуальність розробки власного рішення.
2. Спроектовано трирівневу архітектуру системи: React SPA (клієнт), Express.js + MCP SDK (сервер), Firebase Firestore (БД). Визначено структуру документа clientRequest з 14 полями. Описано схему взаємодії AI-агента з MCP-сервером через 6 інструментів.
3. Реалізовано серверну частину: REST API з 5 маршрутами, MCP-сервер з 6 інструментами та Firebase-клієнт для CRUD-операцій. Реалізовано 3 сценарії AI-обробки: виконання команд, створення запиту з вільного тексту та отримання списку команд.
4. Розроблено клієнтський інтерфейс з 2 сторінками (Запити, AI-асистент) та 5 основними компонентами (Layout, RequestCard, RequestForm, AIAssistant, RequestsPage). Впроваджено валідацію форм через Formik/Yup та типізацію через TypeScript.
5. Проведено функціональне тестування за 8 тест-кейсами — всі тести пройдено успішно. Система коректно виконує CRUD-операції, фільтрацію, пошук та взаємодію з AI-асистентом.

Практичне значення роботи полягає в тому, що розроблений застосунок може бути впроваджений у реальній агенції нерухомості для автоматизації обробки клієнтських запитів. Відкритий код та модульна архітектура дозволяють легко розширювати функціонал.

Перспективи подальшого розвитку: додавання автентифікації користувачів, реалізація push-сповіщень, розширення AI-можливостей (аналітика, прогнозування), мобільна версія.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Anthropic. Model Context Protocol : documentation. URL: https://modelcontextprotocol.io/docs (дата звернення: 17.06.2026).
2. React : documentation. URL: https://react.dev (дата звернення: 17.06.2026).
3. TypeScript : documentation. URL: https://www.typescriptlang.org/docs (дата звернення: 17.06.2026).
4. Firebase. Cloud Firestore : documentation. URL: https://firebase.google.com/docs/firestore (дата звернення: 17.06.2026).
5. Express.js : documentation. URL: https://expressjs.com (дата звернення: 17.06.2026).
6. MUI: The React component library : documentation. URL: https://mui.com (дата звернення: 17.06.2026).
7. Formik : build forms in React. URL: https://formik.org (дата звернення: 17.06.2026).
8. Yup : schema validation. URL: https://github.com/jquense/yup (дата звернення: 17.06.2026).
9. React Router : declarative routing for React. URL: https://reactrouter.com (дата звернення: 17.06.2026).
10. Node.js : documentation. URL: https://nodejs.org/en/docs (дата звернення: 17.06.2026).
11. Бенкс А., Порселло Е. React: швидкий старт : пер. з англ. Київ : Вид-во «Старт», 2023. 368 с.
12. REST API design best practices. URL: https://restfulapi.net (дата звернення: 17.06.2026).
13. Vite : next generation frontend tooling. URL: https://vitejs.dev (дата звернення: 17.06.2026).
14. Groq API : documentation. URL: https://console.groq.com (дата звернення: 17.06.2026).

15. Zod : TypeScript-first schema validation. URL: https://zod.dev (дата звернення: 17.06.2026).
16. MDN Web Docs. JavaScript guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 17.06.2026).
17. MDN Web Docs. An overview of HTTP. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 17.06.2026).
18. npm : documentation. URL: https://docs.npmjs.com (дата звернення: 17.06.2026).
19. Firebase. Cloud Firestore data model. URL: https://firebase.google.com/docs/firestore/data-model (дата звернення: 17.06.2026).
20. MCP TypeScript SDK : GitHub repository. URL: https://github.com/modelcontextprotocol/typescript-sdk (дата звернення: 17.06.2026).