Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ CRM-СИСТЕМИ ОПТИМІЗАЦІЇ ОБСЛУГОВУВАННЯ ЛІФТОВОГО ОБЛАДНАННЯ

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

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІм-21
Мацько Юрій Андрійович

(підпис)

Керівник:
к.т.н., доцент
Христинець Наталія Анатоліївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
«_____« грудня _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Гринюк Сергій Васильович

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т.ТЕРЛЕЦЬКИЙ
«_____« _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Мацько Юрія Андрійовича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Дослідження CRM-системи оптимізації обслуговування ліфтового обладнання*

Керівник роботи *к.т.н., доцент Христинець Наталія Анатоліївна*

затверджені наказом закладу вищої освіти від «17» червня 2025 року № 290/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 09.12.2025р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз проблеми оптимізації управління технічним обслуговуванням ліфтового обладнання
Проєктування архітектури інформаційно-керуючої системи «ELSECRM»
Програмна реалізація та дослідження функціональних можливостей системи
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми оптимізації упр. тех. обслуговуванням ліфтового обладнання	*Христинець Н. А., доцент*		
Проєктування архітектури CRM-системи «ELSECRM»	*Христинець Н. А., доцент*		
Програмна реалізація та дослідження системи	*Христинець Н. А., доцент*		
Нормоконтроль	*Багнюк Н. В., доцент*		
Гарант ОП	*Гринюк С. В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О. І., ст.викладач*		

7. Дата видачі завдання 18.06.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми*	До 01.08.2025 р.	
2.	*Аналіз проблеми оптимізації управління технічним обслуговуванням ліфтового обладнання та постановка завдань дослідження*	До 20.08.2025 р.	
3.	*Проєктування архітектури інформаційно-керуючої системи «ELSECRM»*	До 25.09.2025 р.	
4.	*Програмна реалізація та дослідження функціональних можливостей системи*	До 20.10.2025 р.	
5.	*Висновки та пропозиції*	До 25.10.2025 р	
6.	*Формування списку використаних джерел*	До 27.10.2025 р.	
7.	*Формування додатків*	До 30.10.2025 р.	
8.	*Оформлення ілюстративного матеріалу*	До 05.11.2025 р.	
9.	*Представлення остаточного варіанту кваліфікаційної роботи керівникові*	До 11.11.2025 р.	
10.	*Нормоконтроль*	До 29.11.2025 р.	
11.	*Інструментальна перевірка на академічний плагіат*	До 02.12.2025 р.	
12.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	До 09.12.2025 р.	

Здобувач вищої освіти _____ Мацько Ю. А.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Христинець Н. А.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Мацько Ю. А. IoT-орієнтована CRM-система для управління технічним обслуговуванням ліфтового обладнання. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел, додатків.

Перший розділ присвячено аналізу проблеми управління технічним обслуговуванням ліфтів. Тут розглядаються ключові бізнес-процеси сервісних служб , проводиться огляд існуючих методів та порівняльний аналіз CRM-систем. Також в цьому розділі обґрунтовано роль технологій Інтернету речей (IoT) у задачах моніторингу.

В другому розділі здійснено проєктування архітектури системи «ELSECRM». Обґрунтовано вибір технологічного стеку (Python, FastAPI, Vue.js), розроблено загальну архітектуру та модель взаємодії компонентів, спроєктовано структуру бази даних та рольову модель доступу RBAC.

Третій розділ присвячено програмній реалізації компонентів системи: серверної частини на FastAPI, клієнтської на Vue.js та механізму real-time моніторингу на базі MQTT та WebSocket. Також проведено експериментальне дослідження продуктивності системи.

Ключові слова: CRM, IoT, технічне обслуговування, ліфт, FastAPI, Vue.js, MQTT, WebSocket, моніторинг в реальному часі.

ANNOTATION

Matsko Y. IoT-oriented CRM-system for elevator maintenance management. Manuscript.

Qualifying work of a Master's of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

Qualification work consists of an introduction, three sections, conclusions, a references, appendices.

The first section is devoted to the analysis of the elevator maintenance management problem. It reviews the key business processes of service companies, overviews existing methods , and provides a comparative analysis of CRM systems. This section also substantiates the role of Internet of Things (IoT) technologies in monitoring tasks.

In the second section, the architecture of the «ELSECRM» system is designed. The choice of the technology stack (Python, FastAPI, Vue.js) is substantiated, the general architecture and component interaction model are developed, the database structure and the role-based access control RBAC model are designed.

The third section is devoted to the software implementation of the system components: the backend on FastAPI , the frontend on Vue.js, and the real-time monitoring mechanism based on MQTT and WebSocket. An experimental study of the system's performance is also conducted.

Keywords: CRM, IoT, maintenance, elevator, FastAPI, Vue.js, MQTT, WebSocket, real-time monitoring.

ЗМІСТ

ВСТУП

Сучасний етап розвитку промисловості та сфери послуг в Україні характеризується глибинною цифровою трансформацією, що передбачає перехід від традиційних методів управління до інтегрованих інформаційних систем [1, 2]. Особливої гостроти ця проблема набуває у галузі технічного обслуговування критичної інфраструктури, до якої беззаперечно належить ліфтове господарство. Надійна та безпечна експлуатація ліфтів є невід'ємною складовою функціонування житлового фонду, комерційних та адміністративних будівель, що безпосередньо впливає на якість життя та безпеку громадян. Значна частина ліфтового фонду в Україні експлуатується понад нормативний термін, що підвищує ризики виникнення раптових несправностей та висуває надзвичайно жорсткі вимоги до ефективності, оперативності та якості роботи сервісних служб.

Актуальність теми дослідження зумовлена тим, що більшість вітчизняних сервісних компаній досі спираються на застарілі, переважно паперові, методи управління. Такий підхід створює низку системних проблем, що негативно впливають на всі аспекти діяльності. По-перше, це низька швидкість реагування на аварійні виклики через неефективну комунікацію; диспетчер змушений у телефонному режимі з'ясовувати місцезнаходження та завантаженість персоналу, що призводить до значних затримок та неоптимального розподілу ресурсів. По-друге, це ризик втрати важливих технічних даних через недосконалість паперового документообігу, що ускладнює аналіз історії ремонтів та може призвести до повторних поломок. По-третє, це повна відсутність прозорості як для керівництва, яке не має об'єктивних даних для прийняття управлінських рішень, так і для клієнтів, які не можуть відстежити статус своїх заявок. Аналіз ринку програмного забезпечення показує відсутність доступних спеціалізованих рішень, адаптованих до українських реалій. У цих умовах, розробка та дослідження сучасної інформаційної системи, що поєднує функціонал CRM-системи та технології Інтернету речей, є надзвичайно важливим та своєчасним завданням. Впровадження такої системи дозволить перейти від

реактивної моделі обслуговування «ремонт після відмови» до проактивної «запобігання відмові», що сприятиме підвищенню безпеки, надійності та економічної ефективності у критично важливій галузі.

Кваліфікаційну роботу виконано на кафедрі комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету в межах науково-дослідної роботи кафедри за напрямом «Розробка інтелектуальних інформаційно-керуючих систем та технологій».

Метою кваліфікаційної роботи є підвищення ефективності процесів управління технічним обслуговуванням ліфтового обладнання шляхом розробки та дослідження архітектури спеціалізованої IoT-орієнтованої CRM-системи.

Для досягнення поставленої мети необхідно було вирішити наступні завдання:

– провести глибокий аналіз предметної області та ключових бізнес-процесів сервісних служб для виявлення вузьких місць та потреб в автоматизації;

– виконати огляд та порівняльний аналіз існуючих програмних засобів управління технічним обслуговуванням з метою виявлення їх переваг, недоліків та визначення функціональних прогалин;

– спроєктувати комплексну архітектуру інформаційно-керуючої системи, що поєднує функціонал CRM та засоби IoT-моніторингу;

– розробити програмний прототип системи, що реалізує ключові функціональні модулі;

– провести експериментальне дослідження продуктивності розробленого прототипу для визначення оптимальних параметрів його функціонування;

– провести комплексну оцінку потенційного підвищення ефективності робочих процесів при використанні розробленої системи.

Об'єктом дослідження є процеси управління технічним обслуговуванням ліфтового обладнання. Предметом дослідження є архітектура, функціональні можливості та показники ефективності IoT-орієнтованої CRM-системи. У процесі роботи було використано комплекс загальнонаукових та спеціальних методів дослідження. Методи системного аналізу було застосовано для

декомпозиції складної предметної області на керовані компоненти та для вивчення взаємозв'язків між ними. Інструменти об'єктно-орієнтованого моделювання, зокрема уніфікована мова UML, слугували для візуалізації архітектури та проєктування взаємозв'язків між модулями системи. Для проєктування структури сховища даних використовувалася теорія реляційних баз даних та принципи нормалізації. Безпосередня розробка програмного прототипу спиралася на методи програмної інженерії та гнучкі підходи до розробки. Для перевірки працездатності та оцінки характеристик системи було застосовано методи експериментального дослідження та статистичного аналізу отриманих даних.

Наукова новизна роботи полягає в комплексному вирішенні задачі автоматизації управління сервісом ліфтового обладнання. Вперше було розроблено спеціалізовану архітектуру IoT-орієнтованої CRM-системи, яка, на відміну від існуючих аналогів, поєднує моніторинг у реальному часі на базі універсального відкритого протоколу MQTT та модуля автоматичної генерації технічної документації, що адаптований до стандартів, чинних в Україні. Ця інтеграція дозволяє створити єдиний безшовний інформаційний простір для всіх учасників сервісного процесу.

Було вдосконалено підхід до організації моніторингу стану промислового обладнання. На основі розробленої методики експериментального дослідження було визначено оптимальну частоту передачі телеметричних даних, що забезпечує раціональний баланс між навантаженням на серверну інфраструктуру та актуальністю інформації для диспетчерської служби. Ця методика може бути застосована для налаштування аналогічних систем моніторингу.

Дістала подальшого розвитку концепція рольової моделі доступу для сервісних систем. Запропонована та реалізована п'ятирівнева структура доступу, що включає окрему роль «Клієнт» з обмеженими правами, підвищує не тільки безпеку, але й прозорість сервісних робіт для замовника, що відповідає сучасним клієнтоорієнтованим стратегіям управління.

Практична цінність роботи полягає в розробці готового до дослідної експлуатації прототипу CRM-системи «ELSECRM», створеного на доступних технологіях з відкритим кодом. Впровадження системи дозволяє скоротити час реакції на аварії завдяки автоматичному сповіщенню, зменшити кількість непланових виїздів через превентивний моніторинг та автоматизувати звітність. Результати роботи можуть бути використані малими та середніми сервісними компаніями України для підвищення конкурентоспроможності та якості послуг.

Усі наукові та практичні результати, викладені у кваліфікаційній роботі, отримані автором особисто. Внесок автора полягає у проведенні аналізу предметної області, розробці архітектури системи, безпосередній програмній реалізації прототипу, розробці методики та проведенні експериментальних досліджень продуктивності, аналізі отриманих результатів та формулюванні висновків за результатами роботи.

Апробація результатів. Результати роботи представлені на VI Міжнародній науково-практичній конференції молодих вчених, аспірантів і студентів «Сучасні інформаційні технології та системи в управлінні», яка проводилась з 10 по 11 квітня 2025 року у м. Київ на базі Київського національного економічного університету імені Вадима Гетьмана [3]. Дослідження апробовано публікацією статті у фаховому збірнику «Комп’ютерно-інтегровані технології: освіта, наука, виробництво» [4]. Матеріали апробацій подано в додатку А кваліфікаційної роботи.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ОПТИМІЗАЦІЇ УПРАВЛІННЯ ТЕХНІЧНИМ ОБСЛУГОВУВАННЯМ ЛІФТОВОГО ОБЛАДНАННЯ

1.1 Аналіз предметної області та ключових бізнес-процесів серв-служб

Ефективне функціонування сучасної міської інфраструктури неможливе без надійної та безпечної роботи ліфтового господарства, що є невід'ємною частиною житлових, комерційних та адміністративних будівель. Технічне обслуговування ліфтів є складним, багатоетапним та високорегламентованим процесом, що вимагає чіткої організації, суворого дотримання нормативних вимог та оперативного реагування на позаштатні ситуації. Предметна область управління цими процесами охоплює широкий спектр завдань, що включають планове профілактичне обслуговування, аварійні ремонти, модернізацію обладнання, ведення технічної документації, а також взаємодію з клієнтами, постачальниками запасних частин та контролюючими державними органами. Складність предметної області посилюється великим різноманіттям моделей ліфтового обладнання від десятків світових виробників, кожна з яких має унікальні технічні характеристики та регламенти обслуговування.

Ключові бізнес-процеси типової сервісної служби, що займається обслуговуванням ліфтів, можна структурувати у декілька основних функціональних груп.

Перша група, що є основою діяльності, це процеси планового технічного обслуговування. Вони регламентуються державними стандартами та технічною документацією виробника і включають розробку довгострокових (річних) та короткострокових (місячних, тижневих) графіків профілактичних робіт для кожного об'єкта. Цей процес вимагає врахування таких факторів, як тип обладнання, його вік, інтенсивність експлуатації та історія попередніх відмов. Далі відбувається призначення відповідальних електромеханіків, контроль за своєчасним та повним виконанням регламентних робіт, а також фіксація результатів у відповідних журналах та актах [5]. У традиційній моделі цей процес

є паперовим, де диспетчер вручну формує графіки та видає паперові наряди на виконання робіт, подібні до документа, наведеного в одному з наданих файлів. Це створює значні ризики пропуску планових оглядів, що може призвести до передчасного зносу обладнання та виникнення аварійних ситуацій, а також ускладнює контроль за фактичним виконанням робіт персоналом.

Друга група охоплює процеси реагування на аварійні ситуації та позапланові ремонти. Цей бізнес-процес активується при отриманні заявки від клієнта через телефонний дзвінок або інший канал зв'язку. Він включає реєстрацію інциденту в журналі, попередню класифікацію проблеми, термінове призначення вільної та кваліфікованої ремонтної бригади, виконання ремонту та оформлення відповідної звітності. Ефективність цього процесу безпосередньо впливає на час простою обладнання, що є критичним показником якості послуг, та на рівень задоволеності клієнтів. У ручному режимі диспетчер змушений у телефонному режимі з'ясовувати місцезнаходження та завантаженість механіків, що призводить до значних затримок у реагуванні та неоптимального розподілу ресурсів.

Третя група об'єднує процеси управління технічною документацією та звітністю. Сервісна діяльність супроводжується веденням значного обсягу нормативної та звітної документації: паспортів ліфтів, журналів технічних оглядів, актів виконаних робіт, дефектних актів, протоколів випробувань, кошторисів на ремонт тощо. Традиційний паперовий підхід до ведення цих документів характеризується низкою суттєвих недоліків. До них належать висока ймовірність втрати або пошкодження даних, помилки через людський фактор при заповненні, складнощі з організацією централізованого зберігання, пошуку та аналізу архівної інформації. Відсутність єдиної цифрової бази даних унеможливлює швидкий доступ до історії обслуговування конкретного ліфта безпосередньо на об'єкті.

Четверта група включає процеси управління ресурсами та взаємодії з клієнтами. Це охоплює облік робочого часу та переміщень персоналу, управління складом запасних частин та матеріалів, ведення клієнтської бази, формування та

виставлення рахунків, а також контроль за своєчасністю оплат. Ефективність цих процесів визначає фінансову стабільність та конкурентоспроможність сервісної компанії. Відсутність єдиної інтегрованої системи значно ускладнює аналіз рентабельності обслуговування окремих об’єктів, планування закупівель та побудову прозорих відносин із замовниками.

Комплексний аналіз предметної області свідчить про те, що традиційна модель управління, заснована на ручному плануванні, паперовому документообігу та вербальній комунікації, є неефективною в сучасних умовах. Вона не забезпечує необхідної оперативності, прозорості та точності даних, що призводить до збільшення операційних витрат, зниження якості сервісу та підвищення експлуатаційних ризиків. Це формує об’єктивну потребу в автоматизації та цифровізації ключових бізнес-процесів за допомогою спеціалізованих інформаційних систем.

1.2 Огляд існуючих методів та засобів управління обслуговуванням ліфтів: від традиційних до цифрових

Історично методи управління технічним обслуговуванням ліфтового обладнання пройшли значну еволюцію, що відображає загальні тенденції розвитку інформаційних технологій у промисловості. Цю еволюцію можна умовно поділити на декілька етапів, кожен з яких характеризується певним рівнем автоматизації та типом використовуваних інструментів.

Перший етап, традиційний або паперовий підхід, що досі частково використовується багатьма невеликими компаніями, повністю базується на паперових носіях інформації. Планування робіт здійснюється в паперових графіках та журналах, заявки від клієнтів приймаються телефоном і фіксуються в журналах реєстрації, а кожен виїзд механіка супроводжується заповненням паперового наряду або акту. Вся комунікація між диспетчером та виконавцями відбувається переважно телефоном. Головними недоліками цього підходу є низька оперативність обміну інформацією, висока залежність від людського

фактору, ризики втрати документів, відсутність централізованого та швидкого доступу до історії обслуговування, а також практично повна неможливість проведення глибокої аналітики для прийняття обґрунтованих управлінських рішень.

Другий етап, частково автоматизований, пов’язаний з поширенням персональних комп’ютерів та стандартного офісного програмного забезпечення. На цьому етапі компанії почали використовувати електронні таблиці для ведення графіків ТО, клієнтських баз та простого обліку. Текстові редактори стали інструментом для створення шаблонів актів, протоколів та комерційних пропозицій. Цей підхід дозволив дещо структурувати інформацію, прискорити процес створення типових документів та полегшити їх копіювання, проте не вирішив ключових проблем. Дані залишалися розрізненими у великій кількості файлів та папок, відсутня була можливість спільної роботи в реальному часі, а процеси планування та контролю все ще вимагали значних ручних операцій та не були інтегровані в єдиний інформаційний простір.

Третій етап, цифровий, характеризується впровадженням універсальних систем управління взаємовідносинами з клієнтами. Адаптація таких систем для потреб сервісних служб дозволила централізувати клієнтську базу, автоматизувати процес реєстрації та обробки заявок, вести повну історію взаємодії з кожним клієнтом та частково автоматизувати документообіг [6]. Однак універсальність таких платформ є водночас і їхнім головним недоліком у контексті технічного сервісу. Вони не враховують специфіку предметної області, зокрема необхідність ведення обліку обладнання за складними технічними параметрами, прив’язки завдань до конкретних вузлів та агрегатів, планування складних регламентних робіт та інтеграції з промисловими діагностичними системами. Адаптація таких CRM під потреби ліфтової служби вимагає значних зусиль з кастомізації, додаткового програмування та залучення сторонніх інтеграторів, що робить цей процес дорогим та тривалим.

Четвертий, сучасний етап пов’язаний з появою спеціалізованих IoT-орієнтованих платформ. Це найбільш прогресивний на сьогодні підхід, що

передбачає використання вузькоспеціалізованих програмних платформ, які поєднують функціонал CRM, системи управління виїзним сервісом (Field Service Management, FSM) та технології Інтернету речей. Такі системи не лише автоматизують усі ключові бізнес-процеси від заявки до звіту, а й забезпечують безперервний моніторинг технічного стану обладнання в реальному часі за допомогою сенсорів. Це дозволяє перейти від реактивної моделі обслуговування, що передбачає ремонт після відмови, до проактивної та предиктивної, коли система може на основі аналізу даних прогнозувати можливі несправності та автоматично створювати завдання на профілактичне обслуговування. Провідні світові виробники ліфтів, такі як Otis, KONE та Schindler, вже активно впроваджують власні закриті IoT-платформи, що підтверджує високу ефективність та перспективність даного підходу. Однак їхні рішення є недоступними для незалежних сервісних компаній, що й створює потребу в розробці відкритої та гнучкої альтернативи.

1.3 Класифікація та порівняльний аналіз сучасних CRM-систем для сервісної індустрії

Системи управління взаємовідносинами з клієнтами є не просто програмним забезпеченням, а комплексною бізнес-стратегією, що фокусується на побудові довгострокових та взаємовигідних відносин із замовниками. Історично, поява CRM-систем була зумовлена переходом ринкової парадигми від продукто-орієнтованої до клієнто-орієнтованої моделі, де утримання існуючого клієнта та підвищення його лояльності стає не менш важливим, ніж залучення нового. У контексті сервісної індустрії, де якість та швидкість обслуговування є ключовими факторами конкурентоспроможності, роль CRM-систем значно розширюється, перетворюючись на інтегровані платформи управління операційною діяльністю. Для систематизації та глибокого аналізу функціональних можливостей таких систем доцільно використовувати їх

класифікацію за трьома основними типами: операційні, аналітичні та колаборативні.

Операційні CRM-системи є найбільш поширеним типом і спрямовані на автоматизацію та оптимізацію щоденних бізнес-процесів, що безпосередньо пов’язані із взаємодією з клієнтом. Вони виступають як інструмент для персоналу, що працює на «передовій»: менеджерів з продажу, диспетчерів, маркетологів. У розрізі сервісної компанії, що обслуговує ліфтове обладнання, операційний функціонал є ядром системи і охоплює декілька ключових напрямків. Перш за все, це автоматизація сервісу, що включає повний життєвий цикл заявки: від її реєстрації в системі (незалежно від каналу надходження – телефон, email, клієнтський портал) до її закриття. Система автоматизує створення нарядів на роботи, дозволяє планувати графіки планового технічного обслуговування, а також допомагає диспетчеру приймати рішення щодо призначення оптимального виконавця на основі його кваліфікації, поточного місцезнаходження та завантаженості. Кожен крок виконання завдання фіксується в системі, що забезпечує прозорість та контроль.

Аналітичні CRM-системи фокусуються на зборі, зберіганні, обробці та аналізі великих масивів даних, що накопичуються в процесі операційної діяльності, з метою виявлення прихованих закономірностей, тенденцій та підтримки прийняття стратегічних управлінських рішень. Такі системи використовують технології сховищ даних та інструменти бізнес-аналітики, такі як OLAP-куби, для багатовимірного аналізу. У ліфтовій галузі аналітичний функціонал є надзвичайно потужним інструментом. Він дозволяє керівництву аналізувати історію відмов обладнання для виявлення найбільш проблемних моделей або окремих вузлів, що може вплинути на політику закупівель. Також можливий аналіз продуктивності роботи технічного персоналу, розрахунок середнього часу виконання різних типів робіт та порівняння показників між різними бригадами. Аналітичні модулі допомагають прогнозувати потребу в запасних частинах на основі статистики ремонтів, оптимізувати складські запаси

та приймати обґрунтовані рішення щодо необхідності капітального ремонту або модернізації застарілого обладнання.

Колаборативні CRM-системи, або системи взаємодії, мають на меті покращення комунікації та координації як всередині компанії, так і з зовнішніми сторонами, зокрема з клієнтами та підрядниками. Вони створюють єдиний, інтегрований інформаційний простір, що усуває бар'єри між різними підрозділами та забезпечує всіх учасників процесу актуальною інформацією. Важливою складовою колаборативних систем є клієнтські портали, які надають замовникам безпечний веб-доступ до їх персональних кабінетів. У такому кабінеті клієнт може самостійно створювати та відстежувати статус своїх заявок, переглядати повну історію обслуговування свого обладнання, отримувати та завантажувати звітні документи (акти, рахунки), а також керувати своїми контактними даними. Іншим ключовим елементом є мобільні додатки для виїзних працівників, що забезпечують їм доступ до всієї необхідної інформації безпосередньо на об'єкті: історії ремонтів, технічної документації, наявності запчастин на складі тощо.

Ефективна система для управління обслуговуванням ліфтів повинна бути гібридною, гармонійно поєднуючи елементи всіх трьох типів (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика функціоналу типів CRM-систем у контексті сервісу ліфтів

Функціональний блок	Операційна CRM	Аналітична CRM	Колаборативна CRM
Реєстрація та обробка заявок	✅		(як точка входу)
Планування графіків ТО	✅		
Аналіз частоти відмов обладнання		✅	
Розрахунок KPI		✅	
Функціональний блок	Операційна CRM	Операційна CRM	Колаборативна CRM
Клієнтський веб-портал			✅
Мобільний додаток для механіка	(оновлення статусів)		✅

Отже, порівняльний аналіз показує, що така система не може належати виключно до одного типу.

Універсальні CRM-системи, хоча й мають розвинені операційні та колаборативні функції, часто не володіють необхідною глибиною для аналізу технічних даних та не мають вбудованих інструментів для моделювання специфічних об’єктів. Спеціалізовані закордонні аналоги, навпаки, можуть мати потужний аналітичний функціонал, але бути менш гнучкими у налаштуванні бізнес-процесів та дорогими у впровадженні. Отже, розроблювана система повинна інтегрувати надійний операційний функціонал для щоденної роботи, глибокі аналітичні інструменти для прийняття стратегічних рішень та зручні колаборативні засоби для забезпечення прозорої комунікації.

1.4 Роль технологій Інтернету речей в задачах моніторингу та предиктивного обслуговування

Технології Інтернету речей є одним з ключових напрямків четвертої промислової революції і відіграють вирішальну роль у трансформації підходів до управління промисловим обладнанням. Концепція IoT передбачає об’єднання фізичних об’єктів, оснащених сенсорами, виконавчими пристроями та програмним забезпеченням, у єдину комп’ютерну мережу для збору, передачі та аналізу даних без прямого втручання людини [7]. У контексті промисловості та обслуговування складного технічного обладнання частіше використовується термін «промисловий Інтернет речей», що підкреслює орієнтацію на підвищення надійності, безпеки та ефективності технічних процесів. Впровадження IoT-рішень в сферу обслуговування ліфтів дозволяє перейти від застарілої реактивної моделі ремонту до сучасної проактивної та предиктивної стратегії.

Архітектуру типової IoT-системи для моніторингу промислового обладнання, зокрема ліфтів, доцільно розглядати як трьохрівневу структуру, що включає фізичний, мережевий та прикладний рівні (рис. 1.1).

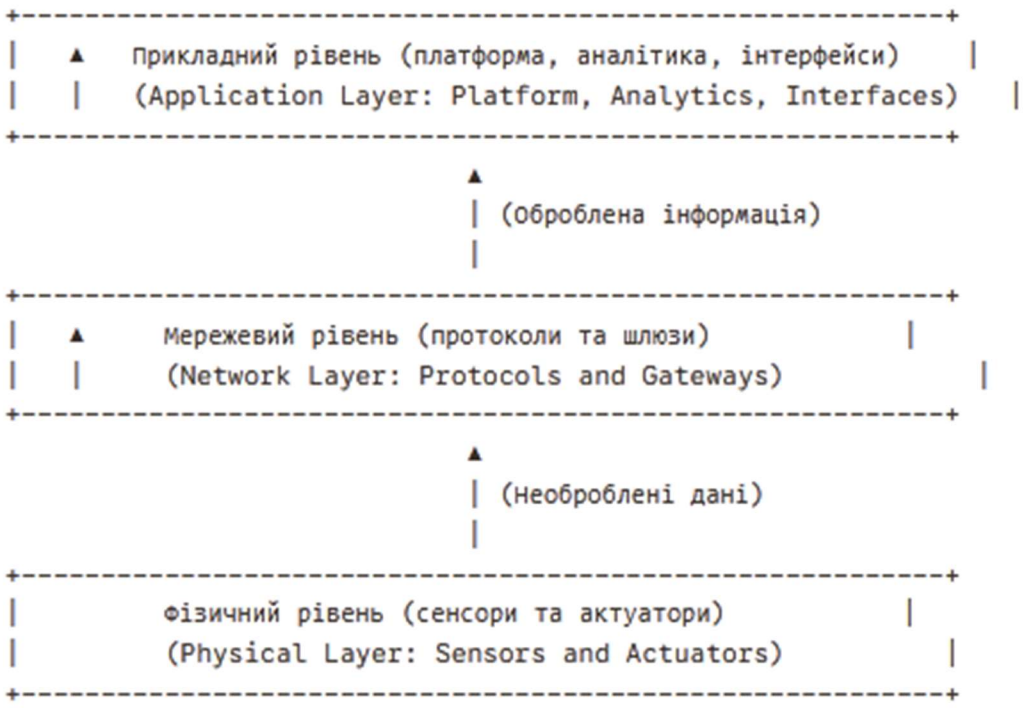

Рисунок 1.1 – Трьохрівнева архітектура IoT-системи

Перший, фізичний або перцептивний рівень, є основою системи і відповідає за безпосередній збір даних з об’єкта моніторингу. Він включає в себе різноманітні сенсори та давачі, що встановлюються на ключових вузлах ліфтового обладнання. Вибір та розміщення цих сенсорів є важливим інженерним завданням, що вимагає глибокого розуміння конструкції та принципів роботи ліфта. До переліку найбільш інформативних сенсорів можна віднести:

– датчики вібрації та акустичного шуму. Ці сенсори, встановлені на корпусі редуктора, двигуна або підшипникових вузлах, дозволяють проводити аналіз вібраційних спектрів. Відхилення від нормальних показників, поява нових частотних складових або зростання амплітуди вібрації можуть з високою точністю свідчити про розвиток механічних дефектів, таких як знос підшипників, пошкодження зубців шестерень або порушення балансування рухомих частин;

– лічильники циклів та датчики стану дверей. Дверні механізми є одним з найбільш навантажених і, як наслідок, вразливих елементів ліфта. Встановлення датчиків, що фіксують кожен цикл відкриття та закриття дверей, дозволяє планувати їх технічне обслуговування та заміну зношених елементів на основі фактичного використання, а не лише календарного графіку, що значно підвищує надійність;

– датчики температури та вологості. Ці датчики зазвичай встановлюються в машинному приміщенні для безперервного контролю умов експлуатації головного приводу та керуючої електроніки. Перевищення допустимих температурних режимів може свідчити про проблеми з вентиляцією або перевантаження двигуна;

– інтерфейсні модулі та перетворювачі. Сучасні ліфтові контролери мають власні діагностичні шини (наприклад, CAN), через які можна отримати великий обсяг інформації про стан системи: коди помилок, напругу та струм живлення двигуна, швидкість руху кабіни тощо. Спеціалізовані модулі дозволяють підключатися до цих шин та транслювати дані у стандартні IoT-протоколи.

Другий, мережевий рівень, відповідає за надійну та ефективну передачу даних від сенсорів до центральної платформи. Вибір комунікаційного протоколу на цьому рівні є критично важливим. Для завдань IoT-моніторингу протокол MQTT (Message Queuing Telemetry Transport) має значні переваги над традиційним HTTP. MQTT працює за асинхронною моделлю «видавець-підписник» (рис. 1.2), що дозволяє повністю від’єднати відправників даних (сенсори) від отримувачів (сервер). Це, разом з легковагістю протоколу та низькими накладними витратами, забезпечує високу ефективність при передачі невеликих обсягів даних від великої кількості пристроїв [8, 9]. Важливою особливістю MQTT є підтримка трьох рівнів якості обслуговування (QoS), що дозволяє гарантувати доставку критично важливих повідомлень навіть за умов нестабільного зв’язку.

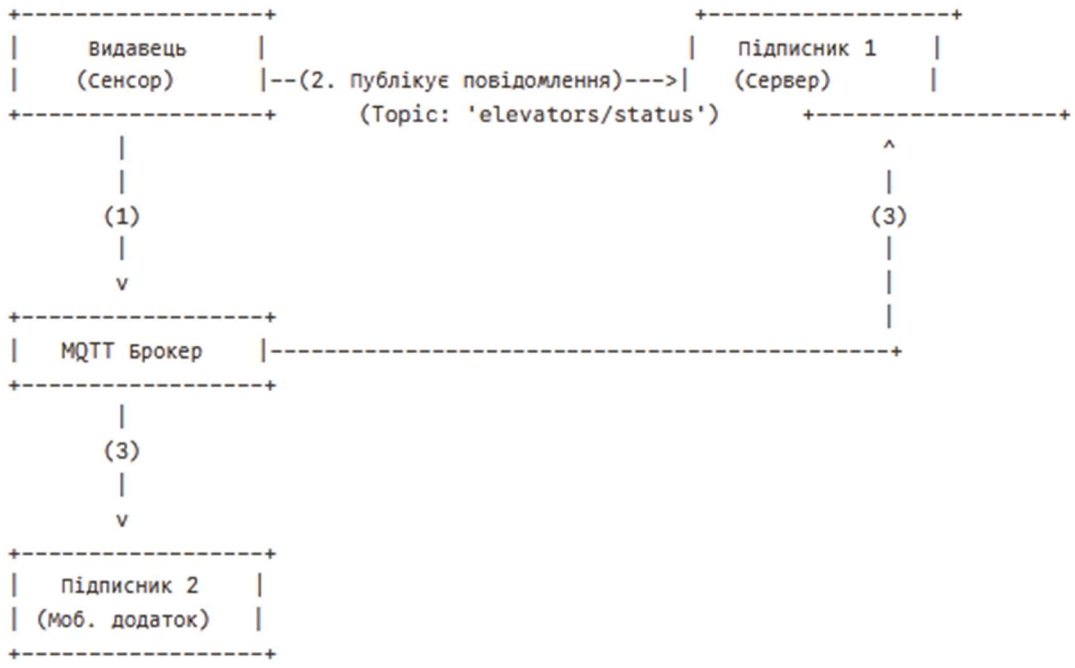

Рисунок 1.2 – Принцип роботи протоколу MQTT за моделлю «видавець-підписник»

Третій, прикладний рівень, є «мозком» системи. Це центральна програмна платформа, яка отримує, зберігає, обробляє, аналізує та візуалізує дані. Саме на цьому рівні відбувається інтеграція IoT-підсистеми з CRM. Дані з сенсорів не просто відображаються на дашборді, а й використовуються для автоматичної генерації сервісних заявок, оновлення статусу обладнання та, що найважливіше, для предиктивного обслуговування. Предиктивне обслуговування (Predictive Maintenance, PdM) є стратегією, що передбачає прогнозування моменту виходу обладнання з ладу на основі аналізу даних про його поточний стан. Аналізуючи часові ряди даних за допомогою алгоритмів інтелектуального аналізу даних та машинного навчання, система може виявляти аномалії та патерни, що передують відмові, і завчасно сигналізувати про необхідність профілактичного втручання. Таким чином, технології IoT дозволяють перейти від застарілої та неефективної реактивної моделі обслуговування до сучасної проактивної та предиктивної, що є ключовим фактором підвищення надійності та економічної ефективності сервісних служб.

1.5 Визначення ключових показників ефективності (KPI) для оцінки роботи сервісної служби

Для об’єктивної оцінки ефективності роботи сервісної служби та кількісного вимірювання впливу на неї впровадження нових інформаційних систем необхідно використовувати набір ключових показників ефективності. KPI є кількісними індикаторами, що дозволяють вимірювати ступінь досягнення стратегічних та операційних цілей компанії та є основою для побудови систем підтримки прийняття рішень та аналітичних панелей [10]. Для компанії, що займається обслуговуванням ліфтового обладнання, де якість сервісу безпосередньо пов’язана з безпекою та безперебійністю роботи критичної інфраструктури, доцільно виділити декілька основних груп таких показників.

Перша група включає показники операційної ефективності, що характеризують швидкість та якість реагування на інциденти.

Середній час ремонту (Mean Time To Repair, MTTR). Цей показник вимірює середній час, що проходить з моменту виникнення несправності до моменту повного відновлення працездатності обладнання. Він включає в себе час на діагностику, очікування запасних частин та безпосередньо час виконання ремонтних робіт. Впровадження IoT-системи дозволяє суттєво скоротити час на діагностику завдяки віддаленому моніторингу, а інтегрована CRM-система оптимізує логістику та призначення виконавців, що також позитивно впливає на цей показник.

Коефіцієнт усунення несправності з першого разу (First-Time Fix Rate, FTFR). Це відсоткове відношення кількості ремонтів, що були успішно завершені за один візит, до загальної кількості ремонтних виїздів. Високе значення цього показника свідчить про якісну попередню діагностику, високу кваліфікацію персоналу та належне забезпечення техніків необхідними інструментами та запчастинами. CRM-система, що надає механіку повну історію обслуговування об’єкта та попередні діагностичні дані з сенсорів, значно підвищує ймовірність успішного ремонту з першого разу.

Друга група об’єднує показники надійності та превентивного обслуговування, що відображають здатність сервісної служби працювати на випередження.

Коефіцієнт доступності обладнання. Це відсоток часу протягом певного періоду, коли обладнання знаходилося у справному, робочому стані. Даний показник є одним з головних індикаторів якості сервісу з точки зору клієнта. Стратегія предиктивного обслуговування, що стає можливою завдяки IoT, безпосередньо спрямована на максимізацію цього показника шляхом запобігання відмовам.

Відсоток планового обслуговування (PMP). Розраховується як відношення часу, витраченого на планові профілактичні роботи, до загального часу, витраченого на всі види обслуговування. Високе значення цього показника (зазвичай понад 80 %) свідчить про зрілість та ефективність сервісної стратегії, оскільки компанія контролює стан обладнання, а не реагує на випадкові аварії (таблиця 1.2).

Таблиця 1.2 – Ключові показники ефективності для сервісної служби

Назва KPI	Опис	Вплив впровадження CRM/IoT системи
MTTR (Mean Time to Repair)	Середній час ремонту. Показник, що вимірює середній час, необхідний для усунення несправності після її виявлення.	Зменшується. Система скорочує час на діагностику завдяки даним з сенсорів та оптимізує логістику призначення механіка, що прискорює початок ремонтних робіт.
FTFR (First-Time Fix Rate)	Коефіцієнт усунення з першого разу. Відсоток ремонтів, що були успішно завершені за один візит без потреби повторного виїзду.	Збільшується. Механік отримує попередні діагностичні дані та повну історію обслуговування об’єкта ще до виїзду, що дозволяє краще підготуватися та взяти необхідні запчастини.
Uptime (Коефіцієнт доступності)	Час безвідмовної роботи обладнання. Відсоток часу, протягом якого обладнання було повністю справним та доступним.	Збільшується. Завдяки предиктивному аналізу та проактивному обслуговуванню, система дозволяє запобігати значній частині несправностей ще до їх виникнення, що мінімізує час простою.
PMP (Planned Maintenance Percentage)	Відсоток планового обслуговування. Співвідношення часу, витраченого на планові профілактичні роботи, до загального часу, витраченого на всі види обслуговування.	Збільшується. Система автоматизує планування та контроль за виконанням регламентних робіт, а предиктивний аналіз перетворює потенційно аварійні ремонти на планові, що підвищує цей показник.

Впровадження інтегрованої CRM-системи дозволяє не лише покращити ці показники, а й автоматизувати їх розрахунок, візуалізувати динаміку на аналітичних панелях та надавати керівництву потужний інструмент для контролю та прийняття обґрунтованих рішень щодо оптимізації бізнес-процесів.

Проведений у даному розділі аналіз предметної області та існуючих засобів управління технічним обслуговуванням ліфтового обладнання дозволив зробити низку важливих висновків, що формують теоретичне та методологічне підґрунтя для подальшого дослідження.

Встановлено, що традиційні методи управління, засновані на паперовому документообігу та ручному плануванні, є неефективними в сучасних умовах, оскільки не забезпечують необхідної оперативності, прозорості та точності даних, що призводить до збільшення операційних витрат та зниження якості сервісу.

Огляд ринку програмного забезпечення виявив відсутність в Україні доступних спеціалізованих рішень, що адаптовані до локальних стандартів та потреб середнього бізнесу, що створює сприятливі умови для розробки вітчизняного конкурентоспроможного продукту.

Показано, що найбільш перспективним є підхід, який поєднує функціональні можливості операційних, аналітичних та колаборативних CRM-систем з технологіями Інтернету речей для моніторингу в реальному часі. Така синергія технологій дозволяє перейти від застарілої реактивної моделі обслуговування до сучасної проактивної та предиктивної, що є ключовим фактором підвищення надійності та економічної ефективності сервісних служб.

Визначено та описано ключові показники ефективності, які дозволять об’єктивно оцінити результати впровадження розробленої системи та її вплив на бізнес-процеси сервісної компанії.

Таким чином, результати аналізу, проведеного в першому розділі, повністю підтверджують актуальність обраної теми дослідження та створюють міцну теоретичну базу для переходу до проєктування архітектури власної інформаційно-керуючої системи у наступному розділі.

У другому розділі поставлено завдання розкрити етапи створення програмного прототипу інформаційно-керуючої системи «ELSECRM», починаючи від підготовки інфраструктури та середовища розробки до функціонального тестування. Метою даного розділу є вирішення конкретних інженерних завдань.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНО-КЕРУЮЧОЇ СИСТЕМИ «ELSECRM»

2.1 Формулювання функціональних та нефункціональних вимог до системи на основі аналізу потреб користувачів

Процес проєктування будь-якої складної інформаційної системи починається з визначення та формалізації вимог. Цей етап є критично важливим, оскільки саме вимоги формують основу для прийняття подальших архітектурних рішень та визначають критерії успішності розробки [11]. Вимоги до системи «ELSECRM» були сформульовані на основі аналізу бізнес-процесів сервісних служб, проведеного в попередньому розділі, а також на основі огляду існуючих рішень та аналізу потреб потенційних користувачів. Вимоги доцільно поділити на дві категорії: функціональні та нефункціональні.

Функціональні вимоги описують, що саме система повинна робити, тобто її основні можливості та поведінку з точки зору користувача. На основі проведеного аналізу було визначено наступний перелік ключових функціональних вимог:

– управління користувачами та розмежування доступу: система повинна підтримувати рольову модель доступу з щонайменше п’ятьма ролями (адміністратор, керівник, диспетчер, механік, клієнт), кожна з яких має чітко визначений набір повноважень;

– ведення централізованої бази даних: система має забезпечувати можливість створення, редагування та зберігання даних про клієнтів, об’єкти обслуговування (ліфти) з їх технічними характеристиками та історією обслуговування;

– моніторинг стану обладнання: система повинна отримувати, обробляти та відображати на інтерактивній панелі (дашборді) телеметричні дані від IoT-пристроїв у реальному часі;

– управління завданнями та планування: система має надавати інструменти для створення планових та аварійних завдань, призначення відповідальних

виконавців, відстеження статусів виконання та ведення календарних графіків технічного обслуговування;

– автоматична генерація документів: система повинна мати модуль для автоматичного створення стандартизованих технічних документів (актів виконаних робіт, дефектних актів) на основі заздалегідь налаштованих шаблонів та даних із системи;

– аналітика та звітність: система повинна надавати керівництву інструменти для генерації звітів та візуалізації ключових показників ефективності роботи служби.

Нефункціональні вимоги визначають, як система повинна виконувати свої функції, та встановлюють критерії її якості, такі як продуктивність, надійність, безпека та зручність використання:

– продуктивність: система повинна забезпечувати швидку відповідь на дії користувача. Час завантаження основних сторінок не повинен перевищувати 3 секунди. Час відповіді серверної частини на стандартні API-запити має бути в межах 200 мілісекунд. Система має бути здатною обробляти до 10 повідомлень на секунду від IoT-пристроїв без суттєвої деградації продуктивності;

– надійність: система повинна бути доступною для користувачів 99,5 % часу. Необхідно передбачити механізми обробки помилок та відновлення після збоїв, а також регулярне резервне копіювання бази даних для запобігання втраті інформації;

– масштабованість: архітектура системи повинна бути спроєктована таким чином, щоб забезпечувати можливість горизонтального та вертикального масштабування для підтримки зростання кількості користувачів та підключених об’єктів у майбутньому;

– безпека: система повинна забезпечувати захист даних від несанкціонованого доступу. Це включає безпечну автентифікацію користувачів (наприклад, з використанням JWT), шифрування даних при передачі мережею (HTTPS/WSS) та захист від поширених веб-вразливостей, таких як SQL-ін’єкції та XSS-атаки;

– зручність використання: графічний інтерфейс системи має бути інтуїтивно зрозумілим, логічним та не переобтяженим зайвими елементами, що дозволить мінімізувати час на навчання персоналу.

2.2 Обґрунтування вибору технологічного стеку для розробки

Вибір технологічного стеку є стратегічним рішенням, що безпосередньо впливає на продуктивність, масштабованість, швидкість розробки та вартість підтримки програмного продукту. Технології для розробки системи «ELSECRM» обиралися з урахуванням сформульованих функціональних та нефункціональних вимог, а також з огляду на сучасні тенденції у веб-розробці.

Для розробки серверної частини було обрано мову програмування Python та веб-фреймворк FastAPI [12, 13]. Розглядалися також альтернативи, такі як Django та Flask. Django є потужним фреймворком з великою кількістю вбудованих інструментів («batteries-included»), однак його синхронна природа робить його менш оптимальним для завдань з великою кількістю одночасних довготривалих з’єднань, що є характерним для IoT-систем. Flask є мінімалістичним та гнучким, але вимагає інтеграції багатьох сторонніх бібліотек для реалізації складного функціоналу. FastAPI було обрано як оптимальне рішення, оскільки його асинхронна архітектура на базі ASGI ідеально підходить для обробки WebSocket-з’єднань та асинхронних запитів від IoT-пристроїв. Висока продуктивність, автоматична генерація інтерактивної документації API (Swagger UI) та вбудована валідація даних за допомогою Pydantic значно прискорюють розробку та підвищують надійність коду.

Для розробки клієнтської частини було обрано прогресивний JavaScript фреймворк Vue.js [14, 15]. Як альтернативи розглядалися фреймворки React та Angular. Angular є потужним та комплексним рішенням, проте його високий поріг входження та жорстка структура можуть сповільнити розробку прототипу. React є найбільш популярною бібліотекою, однак для створення повноцінного додатку вимагає інтеграції з багатьма сторонніми рішеннями (для маршрутизації,

управління станом тощо). Vue.js було обрано завдяки оптимальному балансу між гнучкістю та наявністю вбудованих інструментів. Його компонентна архітектура та реактивна система оновлення даних ідеально підходять для розробки динамічних інтерфейсів, здатних відображати зміни в реальному часі, що є критично важливим для реалізації панелі моніторингу.

У якості системи управління базами даних для етапу прототипування було обрано SQLite через її простоту та відсутність потреби у розгортанні окремого сервера. Водночас архітектура системи спроєктована таким чином, щоб забезпечити легку міграцію на більш потужну реляційну СУБД, таку як PostgreSQL, на етапі промислового впровадження.

Для комунікації між компонентами системи було обрано протокол MQTT та технологію WebSocket. Протокол MQTT є стандартом для передачі даних в IoT-системах завдяки своїй легкості та моделі «видавець-підписник», що є значно ефективнішим за HTTP для даного типу завдань. Для забезпечення миттєвого двонаправленого зв’язку між серверною та клієнтською частинами було використано технологію WebSocket [16], що дозволяє реалізувати оновлення даних на дашборді в реальному часі без постійних запитів з боку клієнта.

2.3 Розробка загальної архітектури системи та моделі взаємодії її компонентів

Архітектура програмної системи є її фундаментальною організацією, що визначає основні компоненти, їх взаємозв’язки та принципи, які керують їх проєктуванням та еволюцією. Вибір архітектурного патерну є одним з найважливіших проєктних рішень, оскільки він безпосередньо впливає на такі нефункціональні характеристики, як продуктивність, масштабованість, надійність та вартість підтримки. Для системи «ELSECRM», враховуючи її вимоги та стадію розробки (прототип з потенціалом до зростання), було

проведено аналіз двох основних архітектурних підходів: мікросервісного та монолітного.

Мікросервісна архітектура передбачає розробку системи як набору невеликих, незалежних сервісів, кожен з яких відповідає за свою частину бізнес-логіки та може розроблятися, розгортатися і масштабуватися окремо [17]. Перевагами цього підходу є висока гнучкість, технологічна гетерогенність та відмовостійкість. Однак для проєктів на ранній стадії мікросервіси вносять значну складність в розгортання, моніторинг та управління взаємодією між сервісами. Кількість мережевих взаємодій між компонентами зростає за формулою $K = n(n-1)/2$, де n – кількість сервісів, що суттєво ускладнює налагодження та трасування запитів.

Монолітна архітектура, навпаки, передбачає розробку системи як єдиного, цілісного застосунку. Це значно спрощує процеси розробки та розгортання на початкових етапах. Головним недоліком класичного моноліту є його низька гнучкість та складність підтримки при зростанні проєкту. Тому для системи «ELSECRM» було обрано компромісний підхід – модульну монолітну архітектуру (таблиця 2.1).

Таблиця 2.1 – Порівняльний аналіз архітектурних підходів для проєкту «ELSECRM»

Критерій	Мікросервісна архітектура	Модульна монолітна архітектура (Обраний підхід)
Швидкість початкової розробки	Низька (через складність налаштування взаємодії)	Висока (спрощена розробка та тестування в рамках одного застосунку)
Складність розгортання	Висока (потребує оркестрації, окремих пайплайнів)	Низька (розгортається як єдиний, цілісний
Масштабованість	Висока (дозволяє масштабувати окремі сервіси)	Помірна (масштабується вся система, але закладено фундамент для міграції)
Відмовостійкість	Висока (збій одного сервісу не зупиняє всю систему)	Низька (збій у моноліті може зупинити весь застосунок)
Вартість підтримки на етапі прототипу	Висока (вимагає більше ресурсів на моніторинг та управління)	Низька (простіший моніторинг та налагодження)

Цей підхід поєднує переваги обох світів: система розгортається як єдиний застосунок, що просто на етапі прототипування, але її внутрішня структура чітко розділена на логічні, слабко зв’язані модулі (наприклад, модуль автентифікації,

модуль управління ліфтами, модуль генерації звітів). Таке рішення забезпечує високу швидкість початкової розробки, але водночас закладає архітектурний фундамент для майбутнього безболісного переходу до мікросервісної архітектури у випадку зростання навантаження.

Загальну архітектуру системи можна представити у вигляді трьох основних рівнів взаємодії, детальний опис яких наведено нижче на рисунку 2.1.

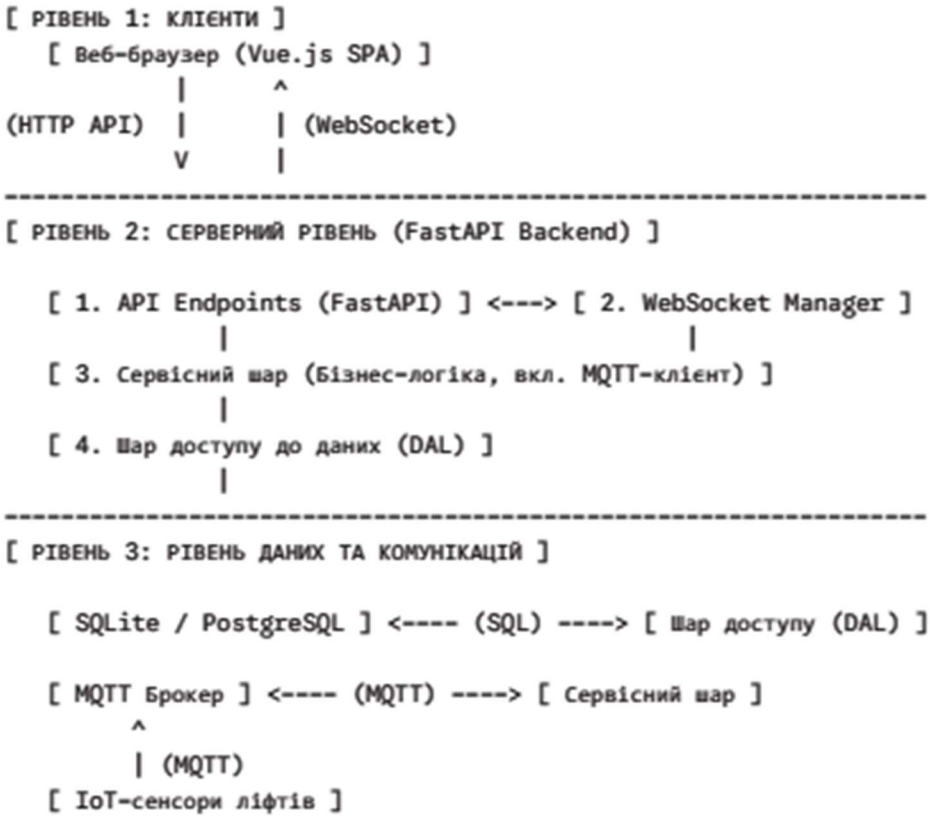

Рисунок 2.1 – Загальна архітектура інформаційно-керуючої системи «ELSECRM»

Модель взаємодії компонентів. Розглянемо модель взаємодії на прикладі ключового сценарію – обробки аварійного сигналу від ліфтового обладнання. Цей процес є критично важливим для демонстрації переваг IoT-підходу. Послідовність взаємодії компонентів системи детально ілюструє діаграма послідовності, наведена на рисунку 2.2.

```
УЧАСНИКИ:
[IoT-сенсор]   [MQTT Брокер]   [Backend (Сервер)]   [База Даних]   [Frontend (Диспетчер)]

ПРОЦЕС:
1.  [IoT-сенсор] -- (Публікує помилку "E34") --> [MQTT Брокер]
2.  [MQTT Брокер] -- (Надсилає "E34") --> [Backend (Сервер)]
3.  [Backend (Сервер)] -- (Оновлює статус "ERROR" для ліфта) --> [База Даних]
4.  [Backend (Сервер)] -- (Транслює оновлення {id: 567, status: "ERROR"}) --> [Frontend (Диспетчер)]
5.  [Frontend (Диспетчер)] -- (Відображає аварію, ліфт стає червоним)
6.  [Frontend (Диспетчер)] -- (Натискає "Створити завдання") --> [Backend (Сервер)]
7.  [Backend (Сервер)] -- (Зберігає нове завдання) --> [База Даних]
```

Рисунок 2.2 – UML-діаграма послідовності для сценарію «Обробка аварійної заявки»

Процес відбувається наступним чином:

– публікація події: сенсор на ліфті фіксує код помилки та формує повідомлення. Це повідомлення публікується у відповідний топік на MQTT-брокері;

– отримання та обробка: серверна частина, яка підписана на цей топік, миттєво отримує повідомлення від брокера. Сервісний шар на сервері валідує дані, визначає критичність помилки та оновлює статус відповідного ліфта в базі даних, фіксуючи час події;

– сповіщення в реальному часі: одночасно з записом в базу даних, сервер через менеджер WebSocket-з'єднань транслює повідомлення про аварію всім підключеним клієнтським додаткам диспетчерів;

– візуалізація та реакція: клієнтський додаток диспетчера, отримавши повідомлення через WebSocket, миттєво оновлює інтерфейс – наприклад, змінює колір іконки ліфта на червоний та виводить сповіщення. Диспетчер, бачачи інцидент, створює нове аварійне завдання, яке через стандартний API-запит відправляється на сервер та зберігається в базі даних.

Дослідження та оптимізація формату передачі даних. Важливим аспектом проєктування IoT-систем є оптимізація обсягу трафіку, особливо якщо пристрої використовують мобільний зв'язок з обмеженим тарифом. Стандартний формат JSON є зручним для читання, але надлишковим. Розглянемо типове повідомлення від сенсора у форматі JSON: {«timestamp»: 1665692400, «floor»: 12, «status»: «error», «code»: «E34»} Розмір такого повідомлення становить

приблизно 65 байт. Як альтернативу можна використати бінарний формат, наприклад, MessagePack, який є більш компактним (таблиця 2.2). Те саме повідомлення у форматі MessagePack займатиме орієнтовно 35 байт.

Таблиця 2.2 – Порівняння розміру корисного навантаження (payload)

Формат даних	Розмір повідомлення (байт)
JSON	65
MessagePack	35

Проведемо розрахунок добового трафіку для парку з 100 ліфтів за умови відправки повідомлень кожні 20 секунд. Кількість повідомлень на добу з одного ліфта: (60 сек / 20 сек) * 60 хв * 24 год = 4320. Загальний трафік для 100 ліфтів на добу розраховується за формулою: Трафік (МБ) = (Розмір_повідомлення * Кількість_повідомлень * Кількість_ліфтів) / (1024 * 1024)

– трафік для JSON: (65 * 4320 * 100) / 1024^2 ≈ 26,76 МБ/добу;

– трафік для MessagePack: (35 * 4320 * 100) / 1024^2 ≈ 14,42 МБ/добу.

Як показують розрахунки, вибір оптимального формату даних дозволяє скоротити трафік майже на 46 %. Хоча на етапі прототипування для простоти реалізації було обрано JSON, архітектура передбачає можливість легкої заміни серіалізатора даних на більш ефективний у промисловій версії системи. Цей аспект дослідження демонструє важливість інженерного підходу не лише до функціональності, але й до оптимізації ресурсів.

2.4 Проєктування структури бази даних та ER-діаграми

Проєктування бази даних є фундаментальним етапом розробки будь-якої інформаційної системи, оскільки від якості її логічної та фізичної структури залежить цілісність, несуперечливість, продуктивність та можливості подальшого розширення всієї системи. База даних виступає як централізоване сховище та єдине джерело правди для всіх компонентів застосунку. Для системи «ELSECRM» було проведено проєктування реляційної моделі даних з

дотриманням принципів нормалізації для усунення надлишковості та забезпечення логічної зв’язності інформації.

На етапі вибору моделі даних було проведено порівняльний аналіз між реляційним підходом (наприклад, PostgreSQL, MySQL) та нереляційним, зокрема документо-орієнтованим підходом (наприклад, MongoDB). Нереляційні бази даних пропонують гнучкість у зберіганні неструктурованих даних, проте предметна область даного проєкту характеризується чітко структурованою та взаємопов’язаною інформацією: клієнти володіють ліфтами, завдання призначаються механікам та прив’язані до конкретних об’єктів. Реляційна модель даних є природною для опису таких зв’язків. Використання реляційної СУБД забезпечує механізми для підтримки цілісності даних за допомогою зовнішніх ключів та гарантує транзакційність операцій (принцип ACID), що є критично важливим для надійності корпоративної системи [18].

На основі аналізу предметної області та сформульованих вимог було визначено ключові сутності, що лягли в основу структури бази даних. Нижче наведено детальний опис основних таблиць та їх атрибутів.

Таблиця користувачів (users) призначена для зберігання облікових даних всіх осіб, що взаємодіють із системою. Вона містить унікальний ідентифікатор (id), ім’я користувача для входу (username), повне ім’я (full_name) та хеш пароля (password_hash). Важливо зазначити, що для забезпечення безпеки паролі зберігаються не у відкритому вигляді, а у вигляді хеш-суми, отриманої за допомогою криптографічних алгоритмів. Також у цій таблиці міститься зовнішній ключ role_id, що посилається на довідникову таблицю ролей.

Таблиця ролей (roles) реалізує довідник можливих ролей у системі. Вона містить унікальний ідентифікатор (id) та назву ролі (name), наприклад, «адміністратор», «диспетчер», «механік». Винесення ролей в окрему таблицю є прикладом нормалізації даних, що дозволяє уникнути дублювання та спрощує адміністрування прав доступу [19].

Таблиця клієнтів (clients) слугує для зберігання інформації про замовників послуг. Вона включає атрибути для назви компанії, контактної особи, адреси, телефону та електронної пошти.

Таблиця ліфтів (elevators) є центральним реєстром об’єктів обслуговування. Кожен запис у цій таблиці відповідає одному ліфту і містить його унікальний ідентифікатор, назву або номер, точну адресу встановлення, модель, серійний номер, дату введення в експлуатацію та іншу технічну інформацію. Для реалізації зв’язку між клієнтом та його обладнанням у цій таблиці присутній зовнішній ключ client_id, що встановлює відношення «один-до-багатьох» з таблицею clients.

Таблиця завдань на обслуговування (maintenance_tasks) призначена для управління робочими процесами. Вона містить інформацію про кожне завдання: його тип (планове, аварійне), детальний опис робіт, статус виконання (нове, в роботі, виконано, скасовано), дату створення та завершення. Ця таблиця має зовнішні ключі elevator_id та mechanic_id для прив’язки завдання до конкретного об’єкта та призначеного виконавця.

Таблиця технічних документів (technical_documents) реалізує функціонал електронного архіву. Вона зберігає метадані про згенеровані системою документи, включаючи тип документа (акт ТО, дефектний акт), дату створення, посилання на відповідне завдання (task_id) та шлях до файлу у файловому сховищі.

Таким чином, спроєктована схема даних охоплює повний цикл бізнес-процесів предметної області: від реєстрації клієнта та його обладнання до формування звітності за результатами виконаних робіт. Встановлені зв'язки між таблицями (one-to-many) гарантують, що кожне технічне завдання та документ будуть однозначно прив’язані до конкретного ліфта та відповідального виконавця, що унеможливлює втрату критично важливої інформації.

Логічна структура бази даних та зв’язки між описаними сутностями детально представлені у вигляді ER-діаграми (діаграми «сутність-зв’язок», рис. 2.3).

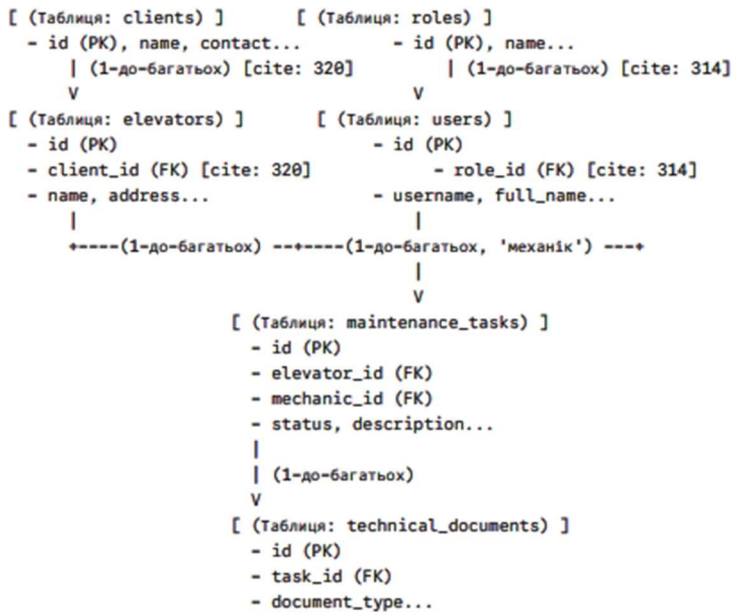

Рисунок 2.3 – ER-діаграма бази даних системи «ELSECRM»

Ця діаграма, що наведена на рисунку 2.3, слугує формальним візуальним описом моделі даних та є основою для створення фізичної структури бази даних за допомогою SQL-скриптів.

2.5 Проєктування рольової моделі доступу (RBAC) та UML-діаграми прецедентів

Забезпечення безпеки даних та розмежування повноважень користувачів є однією з ключових нефункціональних вимог до будь-якої корпоративної інформаційної системи. Для реалізації цієї вимоги в системі «ELSECRM» було проведено проєктування моделі управління доступом. Після аналізу альтернативних підходів, таких як списки контролю доступу (Access Control Lists, ACL), перевагу було віддано моделі управління доступом на основі ролей. Модель ACL, що передбачає призначення прав безпосередньо кожному користувачеві, є простою в реалізації для невеликих систем, однак стає надзвичайно складною в адмініструванні при зростанні кількості користувачів та функціональних можливостей. Модель RBAC, натомість, вводить проміжну сутність – роль, що є сукупністю дозволів. Права доступу надаються ролям, а користувачам призначаються відповідні ролі. Такий підхід значно спрощує

адміністрування, покращує аудит системи та природно відображає ієрархічну структуру та бізнес-процеси сервісної компанії.

На основі аналізу предметної області було спроєктовано п’ять ключових ролей, кожна з яких має унікальний набір повноважень та обмежень, що відповідає її функціональним обов’язкам.

Роль «Адміністратор» є технічною роллю з найвищими повноваженнями в системі. Основна відповідальність адміністратора – забезпечення працездатності системи та управління її користувачами. Дозволи цієї ролі включають створення, редагування, блокування та видалення облікових записів всіх користувачів, а також призначення їм відповідних ролей. Адміністратор також має доступ до системних налаштувань та журналів роботи системи. Важливим обмеженням є те, що ця роль зазвичай не бере участі в операційній діяльності: адміністратор не створює завдань і не взаємодіє з клієнтами.

Роль «Керівник» призначена для вищого менеджменту компанії та орієнтована на стратегічний контроль та аналіз. Користувачі з цією роллю отримують повний доступ до аналітичних модулів системи, включаючи дашборди з ключовими показниками ефективності, фінансову звітність та статистику по роботі обладнання і персоналу. Керівник має права на перегляд інформації по всіх клієнтах, об’єктах та завданнях, але зазвичай не має прав на їх створення чи редагування, що дозволяє уникнути випадкового втручання в операційну діяльність.

Роль «Диспетчер» є центральною операційною роллю в системі. Користувач з цією роллю є відповідальним за координацію всіх сервісних робіт. Його повноваження включають моніторинг стану всього парку ліфтового обладнання в реальному часі, реєстрацію аварійних та планових заявок, створення нових завдань та призначення на них вільних механіків. Диспетчер може змінювати пріоритети завдань, відстежувати їх статус виконання та виступати основною контактною особою для клієнтів у межах поточних робіт.

Роль «Механік» призначена для виїзних технічних фахівців. Інтерфейс для цієї ролі спроєктовано з урахуванням використання на мобільних пристроях.

Механік має доступ лише до списку призначених йому завдань. В межах одного завдання він може переглядати технічну інформацію про об’єкт, змінювати статус завдання (наприклад, «Прийнято в роботу», «В дорозі», «Роботи завершено»), вносити дані про виконані роботи та використані запчастини, а також ініціювати генерацію акту виконаних робіт або дефектного акту. Доступ до інформації по інших завданнях чи об’єктах для цієї ролі суворо обмежений.

Роль «Клієнт» є зовнішньою роллю, що надається представникам компаній-замовників. Основна мета цієї ролі – підвищення прозорості та якості сервісу. Клієнт отримує доступ до особистого кабінету, де він може переглядати інформацію виключно по своїх об’єктах: їх поточний статус, повну історію технічного обслуговування та ремонтів, а також завантажувати пов’язані з ними документи. Крім того, клієнт має можливість самостійно створювати нові заявки на обслуговування та відстежувати їх статус.

Взаємодія цих ролей з основними функціями системи формально змодельована за допомогою діаграми прецедентів (Use Case Diagram) з використанням уніфікованої мови моделювання, що наведена на рисунку 2.4. Для більш глибокого моделювання динаміки одного з ключових об’єктів системи – «Завдання на обслуговування» – доцільно використати діаграму станів.

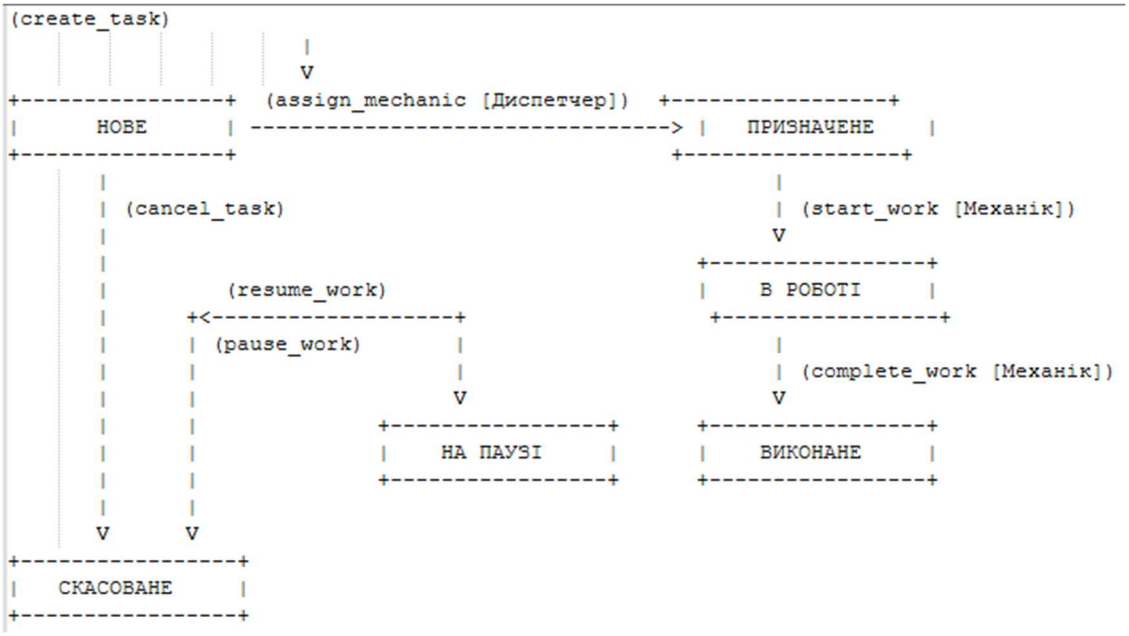

Рисунок 2.4 – UML-діаграма станів для сутності «Завдання на обслуговування»

Ця діаграма чітко визначає життєвий цикл завдання, можливі стани, в яких воно може перебувати, та події, що спричиняють переходи між цими станами. Таке формальне моделювання дозволяє уникнути логічних помилок при програмній реалізації бізнес-процесів.

2.6 Техніко-економічне обґрунтування розробки та впровадження системи

Будь-який інженерний проєкт, особливо у сфері інформаційних технологій, повинен бути обґрунтований не лише з технічної, але й з економічної точки зору. Метою даного підрозділу є проведення техніко-економічного обґрунтування розробки та впровадження інформаційно-керуючої системи «ELSECRM». Обґрунтування базується на розрахунку очікуваних інвестиційних витрат, прогнозованих економічних вигод від впровадження системи та розрахунку ключових показників інвестиційної привабливості проєкту [20].

2.6.1 Методика проведення економічної оцінки

Для оцінки економічної ефективності проєкту буде використано стандартну методику, що базується на розрахунку чистої приведеної вартості (Net Present Value), індексу рентабельності (Profitability Index), та терміну окупності проєкту (Payback Period). Розрахунки будуть проводитись для умовної сервісної компанії середнього розміру, що має в своєму штаті 5 виїзних електромеханіків, 2 диспетчерів та обслуговує парк зі 100 ліфтів. Усі вартісні показники наведено в умовних одиницях (у.о.) для універсальності розрахунків.

2.6.2 Розрахунок капітальних інвестицій та експлуатаційних витрат

Загальні витрати на проєкт складаються з одноразових капітальних інвестицій на розробку та впровадження, а також щорічних експлуатаційних витрат на підтримку системи.

Капітальні інвестиції включають:

– витрати на розробку програмного забезпечення. Оцінка трудомісткості розробки прототипу складає 1080 людино-годин. При умовній вартості години

роботи інженера-програміста в 15 у.о., загальна вартість розробки складе: B_розробки = 1080 год * 15 у.о./год = 16200 у.о;

– витрати на придбання серверного обладнання. Для розгортання системи на 100 об’єктів необхідний базовий віртуальний приватний сервер (VPS). Середня річна вартість такого сервера складає 240 у.о.;

– витрати на початкове оснащення IoT-пристроями. Припустимо, що на першому етапі оснащується 20 % парку (20 ліфтів). Вартість одного комплекту базових сенсорів та контролера становить 100 у.о.: B_IoT = 20 шт * 100 у.о./шт = 2000 у.о.;

– витрати на впровадження та навчання персоналу. Ці витрати оцінюються в 10% від вартості розробки: B_впровадження = 16200 у.о. * 0,1 = 1620 у.о.

Таким чином, сумарні капітальні інвестиції (CI) складуть: CI = 16200 + 240 + 2000 + 1620 = 20060 у.о.

Щорічні експлуатаційні витрати (CO) включають вартість хостингу сервера (240 у.о.) та витрати на технічну підтримку та оновлення системи (орієнтовно 15 % від вартості розробки), що складає 16200 * 0,15 = 2430 у.о.. CO = 240 + 2430 = 2670 у.о./рік. Детальніше підсумуємо в таблиці 2.3.

Таблиця 2.3 – Структура інвестиційних та експлуатаційних витрат

Категорія витрат	Стаття витрат	Розрахунок	Сума (у.о.)
Одноразові капітальні інвестиції (CI)			
	Витрати на розробку ПЗ	1080 год * 15 у.о./год	16200
	Витрати на серверне обладнання	Річна вартість VPS	240
	Витрати на оснащення IoT-пристроями	20 шт * 100 у.о./шт	2000
	Витрати на впровадження та навчання	10 % від вартості розробки	1620
	Разом капітальних інвестицій (CI)		20060
Щорічні експлуатаційні витрати (CO)			
	Вартість хостингу сервера		240
	Технічна підтримка та оновлення	15 % від вартості розробки	2430
	Разом експлуатаційних витрат (CO)		2670

2.6.3 Розрахунок прогнозованих економічних вигод

Економічний ефект від впровадження системи формується за рахунок прямої економії ресурсів та отримання непрямих переваг. Розглянемо економію робочого часу персоналу на прикладах економічних розрахунків.

Диспетчерський персонал. Завдяки автоматизації реєстрації заявок, планування та контролю, диспетчер може зекономити до 25 % робочого часу. При умовній заробітній платі диспетчера 700 у.о./міс, річна економія на 2 диспетчерів складе: Е_дисп = 2 чол * 700 у.о./міс * 12 міс * 0.25 = 4200 у.о./рік.

Технічний персонал. Механіки економлять до 15 % часу за рахунок автоматичної генерації звітів та швидкого доступу до технічної інформації. При умовній заробітній платі механіка 900 у.о./міс, річна економія на 5 механіків складе: Е_мех = 5 чол * 900 у.о./міс * 12 міс * 0,15 = 8100 у.о./рік.

Зниження витрат на аварійні ремонти. Впровадження IoT-моніторингу та перехід до проактивного обслуговування дозволяє знизити кількість аварійних виїздів на 20-30 %. Припустимо, що в середньому на 100 ліфтів припадає 50 аварійних виїздів на місяць, а вартість одного такого виїзду (включаючи транспорт, робочий час, можливі термінові закупівлі) становить 50 у.о. Зниження на 25 % дасть економію: Е_аварії = 50 виїздів/міс * 12 міс * 50 у.о./виїзд * 0,25 = 7500 у.о./рік.

Підвищення ефективності використання транспорту. Завдяки оптимізації маршрутів механіків та скороченню кількості позапланових виїздів, можна досягти економії паливно-мастильних матеріалів на 10-15 %. При умовних витратах на транспорт 400 у.о./міс на компанію, річна економія складе: Е_транспорт = 400 у.о./міс * 12 міс * 0.15 = 720 у.о./рік.

Сумарна річна економічна вигода (В) складе: В = 4200 + 8100 + 7500 + 720 = 20520 у.о./рік.

2.6.4 Розрахунок показників економічної ефективності

На основі розрахованих витрат та вигод проведемо розрахунок ключових показників інвестиційної привабливості проєкту.

Термін окупності – це період часу, необхідний для того, щоб доходи, згенеровані проєктом, покрили початкові інвестиції. Простий термін окупності розраховується за формулою: PP = CI / (B - CO) = 20060 / (20520 - 2670) = 20060 / 17850 ≈ 1,12 року. Це означає, що проєкт окупить себе приблизно через 1 рік та 1,5 місяці, що є надзвичайно високим показником для IT-проєктів.

Коефіцієнт рентабельності інвестицій (Return on Investment, ROI) показує прибутковість або збитковість проєкту відносно зроблених інвестицій. Розрахунок за перший рік експлуатації: ROI = ((B - CO) - CI) / CI * 100 % ROI = ((17850) - 20060) / 20060 * 100 % ≈ -11% (за перший рік).

Розрахуємо ROI за перші три роки експлуатації, враховуючи, що інвестиції є одноразовими: ROI_3роки = ((17850 * 3) - 20060) / 20060 * 100 % = (53550 - 20060) / 20060 * 100% ≈ 167 %. Такий високий показник рентабельності за трирічний період свідчить про високу економічну привабливість проєкту (на рисунку 2.5).

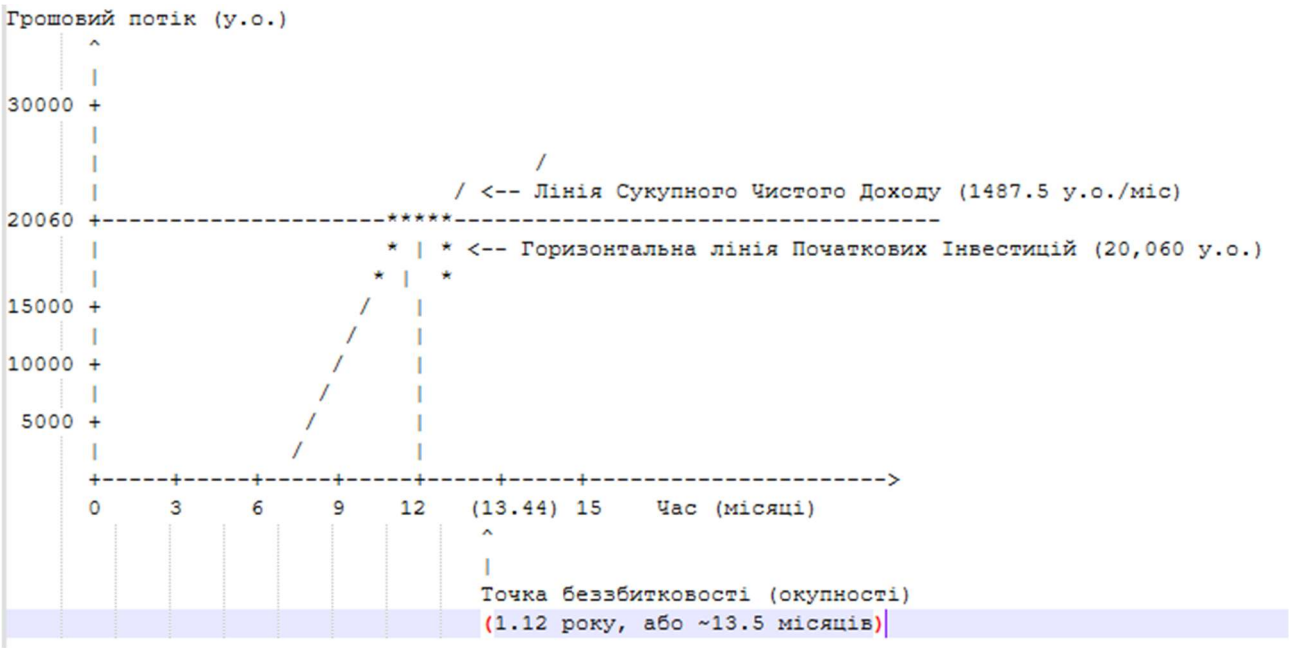

Рисунок 2.5 – Графік точки беззбитковості проєкту

Таким чином, проведений техніко-економічний аналіз переконливо доводить, що розробка та впровадження інформаційно-керуючої системи «ELSECRM» є не лише технічно доцільним, але й економічно вигідним проєктом.

Очікувана економія ресурсів та підвищення ефективності роботи значно перевищують інвестиційні витрати, забезпечуючи швидку окупність та високу рентабельність інвестицій в довгостроковій перспективі. Це підтверджує практичну цінність та комерційний потенціал розробленого в рамках кваліфікаційної роботи рішення.

У даному розділі було проведено комплекс робіт з системного проєктування інформаційно-керуючої системи «ELSECRM». На основі детального аналізу предметної області, проведеного в попередньому розділі, було сформульовано вичерпний перелік функціональних та нефункціональних вимог, що стали основою для всіх подальших проєктних рішень.

Було здійснено обґрунтований вибір сучасного технологічного стеку, що забезпечує необхідний рівень продуктивності для обробки даних в реальному часі, надійності та потенціалу для майбутнього масштабування. Розроблено гнучку модульну монолітну архітектуру системи, яка поєднує простоту розробки на етапі прототипування з можливістю подальшого переходу до мікросервісного підходу.

В рамках проєктування було детально опрацьовано структуру бази даних, що забезпечує цілісність та несуперечливість інформації, а також розроблено п’ятирівневу рольову модель доступу, що гарантує безпеку даних та розмежування повноважень користувачів відповідно до їхніх функціональних обов’язків. Результати проєктування були формалізовані за допомогою стандартних нотацій моделювання, зокрема діаграм UML та ER-діаграм.

Таким чином, результати, отримані в цьому розділі, створюють міцний та всебічно обґрунтований архітектурний та проєктний фундамент для безпосередньої програмної реалізації системи, яка буде детально розглянута у наступному, третьому розділі кваліфікаційної роботи.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ СИСТЕМИ

Третій розділ кваліфікаційної роботи присвячено практичній реалізації спроєктованої в попередньому розділі інформаційно-керуючої системи «ELSECRM». Цей етап є критично важливим ланцюгом у процесі створення програмного продукту, оскільки він дозволяє верифікувати коректність архітектурних гіпотез, висунутих у другому розділі, та адаптувати їх до реальних умов експлуатації. Особлива увага в процесі імплементації приділялася забезпеченню надійної взаємодії між різнорідними компонентами системи — апаратними засобами збору телеметрії та програмним комплексом обробки даних. Такий підхід вимагає дотримання сучасних стандартів інженерії програмного забезпечення, зокрема принципів модульності, чистоти коду та забезпечення масштабованості, що є необхідною умовою для подальшого переходу від прототипу до промислового рішення. Метою даного розділу є демонстрація перетворення теоретичних архітектурних рішень у працюючий програмний продукт, а також проведення експериментального дослідження його ключових характеристик. У цьому розділі буде детально розглянуто реалізацію серверної та клієнтської частин, механізмів взаємодії в реальному часі та модуля генерації документів. Кожен етап буде проілюстровано фрагментами програмного коду, що відображають ключові аспекти імплементації.

3.1 Розробка серверної частини на базі FastAPI

Серверна частина є ядром системи, що відповідає за реалізацію бізнес-логіки, обробку даних та надання програмного інтерфейсу (API) для клієнтських додатків. Для її розробки було використано фреймворк FastAPI [12]. Хоча на етапі прототипування вся логіка була реалізована в єдиному файлі для прискорення розробки, для дипломної роботи доцільно представити більш структурований

підхід, що відповідає сучасним інженерним практикам. Таким підходом є шарова архітектура, яка передбачає логічне розділення коду на шари за їхньою відповідальністю (рис. 3.1).

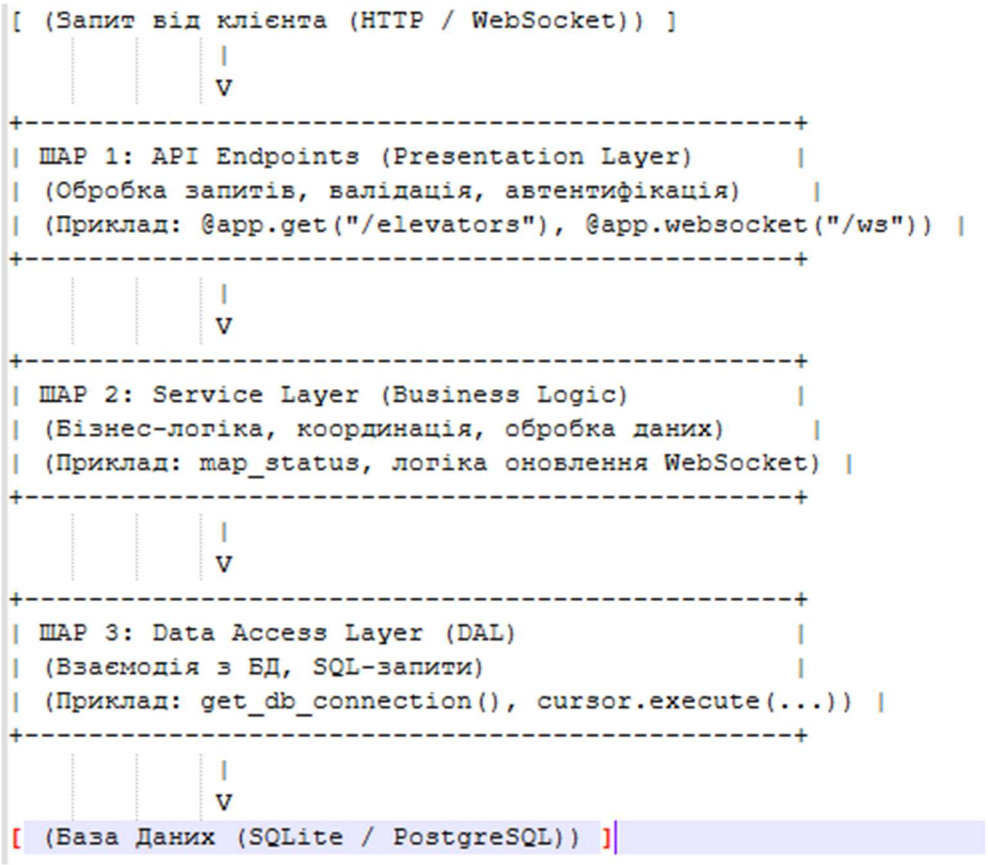

Рисунок 3.1 – Шарова архітектура серверної частини (Backend)

Процес розробки ключового функціоналу можна продемонструвати на прикладі створення нового завдання на обслуговування, спираючись на код, розроблений у прототипі. Процес починається з визначення структури даних. У прототипі для цього використовуються моделі Pydantic (лістинг 3.1), що дозволяє FastAPI автоматично валідувати вхідні JSON-об’єкти.

Лістинг 3.1 – Моделі Pydantic для сутності «Завдання» (з прототипу)

```
from pydantic import BaseModel
from datetime import date
class TaskBase(BaseModel):
description: str
status: str = «нове»
class TaskCreate(TaskBase):
```

```
elevator_id: int
mechanic_id: int
planned_date: date
class Task(TaskBase):
id: int
elevator_id: int
mechanic_id: int
planned_date: date
class Config:
orm_mode = True
```

кінець лістингу 3.1

Далі, для взаємодії з базою даних, у прототипі визначено модель SQLAlchemy (лістинг 3.2), що відображає структуру відповідної таблиці.

Лістинг 3.2 – Модель SQLAlchemy для таблиці завдань (з прототипу)

```
from sqlalchemy import Column, Integer, String, Date, ForeignKey
from sqlalchemy.orm import relationship
class TaskDB(Base):
__tablename__ = «tasks»
id = Column(Integer, primary_key=True, index=True)
description = Column(String, index=True)
status = Column(String, default=«нове»)
elevator_id = Column(Integer, ForeignKey(«elevators.id»))
mechanic_id = Column(Integer, ForeignKey(«users.id»))
planned_date = Column(Date)
elevator = relationship(«ElevatorDB», back_populates=«tasks»)
mechanic = relationship(«UserDB», back_populates=«tasks»)
```

кінець лістингу 3.2

Нарешті, створюється кінцева точка API (ендпоінт), яка об’єднує ці компоненти. У прототипі логіка створення запису в базі даних реалізована безпосередньо в функції ендпоінту (лістинг 3.3).

Лістинг 3.3 – Ендпоінт API для створення нового завдання (з прототипу)

```
from fastapi import FastAPI, Depends
from sqlalchemy.orm import Session
```

```
app = FastAPI()
@app.post(«/tasks/», response_model=Task)
def create_task(task: TaskCreate, db: Session = Depends(get_db)):
db_task = TaskDB(**task.dict())
db.add(db_task)
db.commit()
db.refresh(db_task)
    return db_task
```

кінець лістингу 3.3

Реалізація, наведена вище, демонструє використання ключових можливостей фреймворку FastAPI. Проте для повноцінної корпоративної системи необхідно реалізувати механізми автентифікації та авторизації для захисту кінцевих точок API від несанкціонованого доступу.

Невід'ємною частиною розробки безпечного API є реалізація механізму автентифікації. У сучасних веб-додатках найбільш поширеним є підхід на основі токенів, зокрема JSON Web Tokens (JWT). Цей підхід передбачає, що після успішного входу в систему з логіном та паролем сервер генерує для клієнта унікальний токен доступу. Надалі клієнт повинен передавати цей токен у заголовках кожного свого запиту до захищених ресурсів. Сервер, отримуючи запит, валідує токен і на основі інформації, що в ньому міститься (наприклад, user_id та role), визначає, чи має користувач право на виконання даної операції.

Для реалізації цього функціоналу необхідно додати ендпоінт для входу в систему, який приймає логін та пароль, перевіряє їх і у випадку успіху повертає JWT-токен (лістинг 3.4).

Лістинг 3.4 – Ендпоінт для автентифікації користувача та видачі JWT

```
 from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from datetime import timedelta
# ... імпорти функцій для роботи з паролями та JWT
@app.post(«/token»)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm =
Depends(), db: Session = Depends(get_db)):
# 1. Знаходимо користувача в БД за логіном
```

```
    user = authenticate_user(db, form_data.username, form_data.password)
    if not user:
    raise HTTPException(
    status_code=status.HTTP_401_UNAUTHORIZED,
    detail=«Неправильний логін або пароль»,
    headers={«WWW-Authenticate»: «Bearer»},
    )
    # 2. Встановлюємо термін дії токену
     access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    # 3. Створюємо токен доступу
     access_token = create_access_token(
     data={«sub»: user.username, «role»: user.role.name},
   expires_delta=access_token_expires
     )
    # 4. Повертаємо токен клієнту
     return {«access_token»: access_token, «token_type»: «bearer»}
```

кінець лістингу 3.4

Після реалізації механізму видачі токенів необхідно захистити ендпоінти, які вимагають автентифікації. Це робиться за допомогою системи залежностей FastAPI. Створюється функція-залежність (лістинг 3.5), яка перевіряє наявність та валідність токену в заголовках запиту.

Лістинг 3.5 – Залежність для перевірки автентифікації

```
from jose import JWTError, jwt
oauth2_scheme = OAuth2PasswordBearer(tokenUrl=«token»)
async def get_current_user(token: str = Depends(oauth2_scheme), db: Session
   = Depends(get_db)):
credentials_exception = HTTPException(
status_code=status.HTTP_401_UNAUTHORIZED,
detail=«Не вдалося валідувати облікові дані»,
headers={«WWW-Authenticate»: «Bearer»},
)
try:
# 1. Декодуємо токен
payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
username: str = payload.get(«sub»)
if username is None:
raise credentials_exception
```

```
except JWTError:
raise credentials_exception
# 2. Знаходимо користувача в БД
user = get_user_by_username(db, username=username)
if user is None:
raise credentials_exception
# 3. Повертаємо об’єкт користувача
return user
```

кінець лістингу 3.5

Тепер цю залежність get_current_user можна додати до будь-якого ендпоінту, щоб зробити його захищеним. Модифікуємо наш ендпоінт для створення завдань (лістинг 3.6).

Лістинг 3.6 – Захищений ендпоінт для створення завдання (модифікація)

```
# ... імпорти
from .auth import get_current_user
@app.post(«/tasks/», response_model=Task)
def create_task(
task: TaskCreate,
db: Session = Depends(get_db),
current_user: models.User = Depends(get_current_user)
):
# Тепер ми маємо доступ до даних поточного користувача
# Можна додати перевірку, чи маєcurrent_user права на створення завдань
if current_user.role.name not in [«admin», «dispatcher»]:
raise HTTPException(status_code=403, detail=«Недостатньо прав для виконання
   операції»)
db_task = TaskDB(**task.dict())
db.add(db_task)
db.commit()
db.refresh(db_task)
return db_task
```

кінець лістингу 3.6

Така реалізація механізму автентифікації та авторизації на основі JWT є сучасним промисловим стандартом, що забезпечує високий рівень безпеки та гнучкість у керуванні доступом до ресурсів API.

3.2 Розробка клієнтської частини (Frontend) та інтерфейсу користувача

Клієнтська частина, або frontend, є обличчям інформаційної системи, з яким безпосередньо взаємодіють користувачі. Її головними завданнями є надання інтуїтивно зрозумілого інтерфейсу для доступу до функціоналу системи, візуалізація даних, отриманих від серверної частини, та забезпечення швидкого відгуку на дії користувача. Для системи «ELSECRM» клієнтську частину було реалізовано у вигляді односторінкового застосунку (Single-Page Application). Ця архітектура дозволяє завантажити весь необхідний код (HTML, CSS, JavaScript) один раз, а подальшу взаємодію з сервером здійснювати динамічно через API, що забезпечує високу швидкість навігації та багатий користувацький досвід, подібний до роботи з десктопними програмами.

В основі розробки лежить фреймворк Vue.js, який, згідно з реалізацією в прототипі, підключається через мережу доправлення вмісту (CDN). Це спрощує процес розробки та розгортання на початкових етапах. Архітектура додатку побудована на компонентному підході, де весь інтерфейс розбивається на логічні, незалежні та перевикористовувані компоненти. Навіть в рамках одного файлу, як у прототипі, можна виділити логічні блоки: компонент панелі моніторингу, компонент списку завдань, компонент форми входу тощо.

Управління станом додатку в прототипі реалізовано за допомогою реактивного об’єкта data в кореневому екземплярі Vue. Цей об’єкт зберігає всі динамічні дані, такі як список ліфтів, перелік завдань, інформація про поточного користувача. Будь-яка зміна цих даних автоматично призводить до оновлення відповідних частин інтерфейсу, що є ключовою особливістю фреймворку Vue.js.

Взаємодія з серверною частиною для отримання початкових даних відбувається через асинхронні HTTP-запити за допомогою стандартного браузерного Fetch API. При завантаженні додатку, у хуці життєвого циклу mounted, викликаються методи для завантаження списку ліфтів та завдань (лістинг 3.7).

Лістинг 3.7 – Фрагмент коду для отримання списку завдань (з прототипу)

```
// ... усередині об'єкта methods екземпляра Vue
async fetchTasks() {
try {
const response = await fetch('/tasks/');
if (response.ok) {
this.tasks = await response.json();
} else {
console.error(«Помилка завантаження завдань»);
  }
} catch (error) {
console.error(«Помилка мережі:», error);
  }
},
```

кінець лістингу 3.7

У цьому прикладі метод fetchTasks відправляє GET-запит на ендпоінт /tasks/. Після отримання успішної відповіді, він розбирає JSON та записує отриманий масив завдань у реактивну властивість this.tasks.

Для відображення даних в інтерфейсі використовуються директиви Vue.js безпосередньо в HTML-розмітці. Директива v-for дозволяє ітеративно відмальовувати елементи списку на основі масиву даних, а умовні директиви та прив'язка класів (:class) – динамічно змінювати вигляд елементів залежно від їх стану (лістинг 3.8).

Лістинг 3.8 – Фрагмент HTML-шаблону для відображення картки ліфта

```
``
<div id=«elevator-dashboard»>
  <div v-for=«elevator in elevators» :key=«elevator.id» class=«elevator-
   card»
:class=«{ 'status-error': elevator.status === 'error', 'status-ok':
   elevator.status === 'idle' }»>
  <h3>{{ elevator.name }}</h3>
  <p>Адреса: {{ elevator.location }}</p>
  <p>Статус: {{ elevator.status }}</p>
  <p>Поверх: {{ elevator.floor }}</p>
  </div>
```

```
    </div>
```

кінець лістингу 3.8

Цей фрагмент коду демонструє, як на основі масиву elevators генерується набір карток. Кожній картці динамічно присвоюється CSS-клас (status-error або status-ok) залежно від значення поля status відповідного об’єкта, що дозволяє візуально виділяти проблемні об’єкти (рис. 3.2).

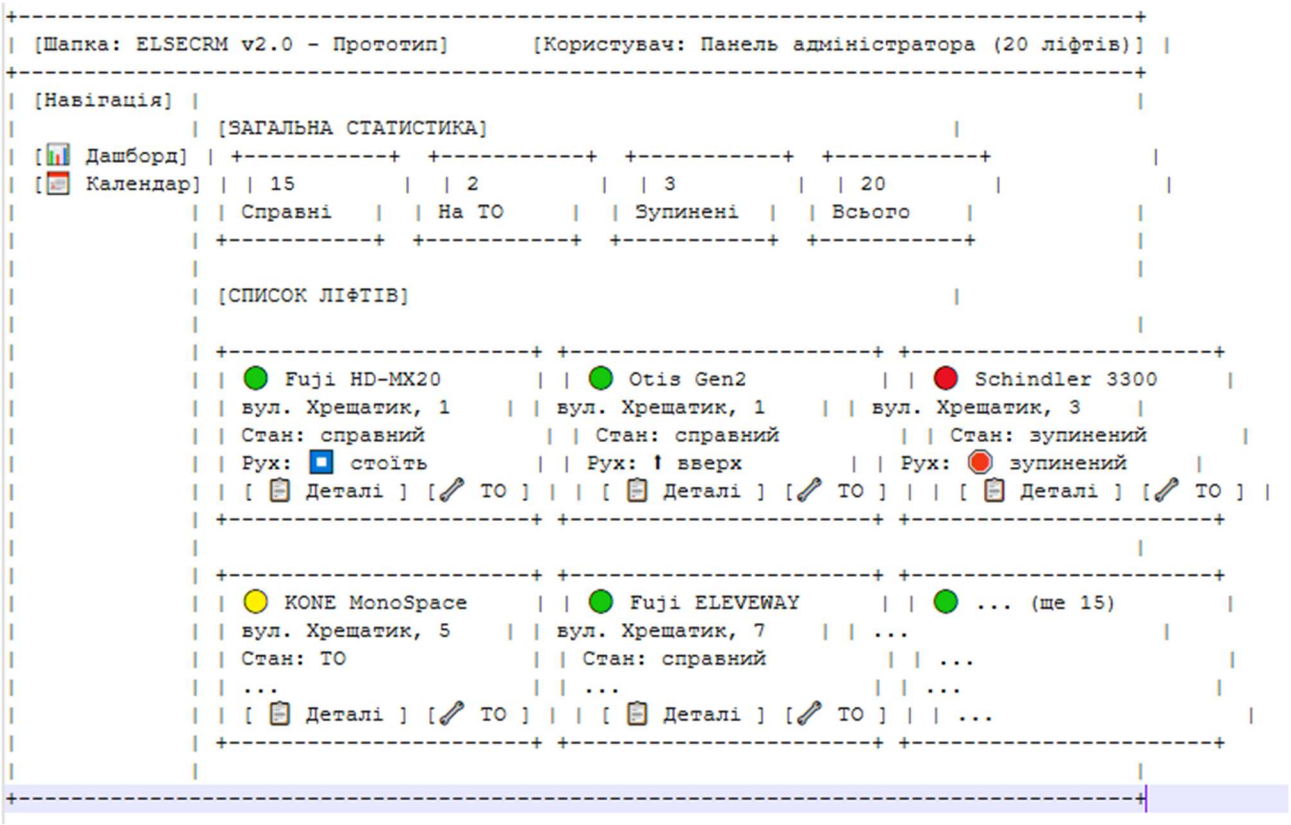

Рисунок 3.2 – Інтерфейс головної панелі моніторингу системи «ELSECRM»

Ключовим елементом клієнтської частини є реалізація оновлень у реальному часі. У прототипі це досягається шляхом встановлення постійного WebSocket-з’єднання з сервером при завантаженні додатку (лістинг 3.9).

Лістинг 3.9 – Реалізація WebSocket-клієнта (з прототипу)

```
// ... усередині хука mounted екземпляра Vue
connectWebSocket() {
const socket = new WebSocket(«ws://localhost:8000/ws»);
socket.onmessage = (event) => {
```

```
const updatedElevators = JSON.parse(event.data);
// Оновлюємо дані в нашому локальному стані
updatedElevators.forEach(updatedElevator => {
const index = this.elevators.findIndex(e => e.id === updatedElevator.id);
if (index !== -1) {
// Vue автоматично оновить інтерфейс завдяки реактивності
  this.elevators[index] = updatedElevator;
  }
  });
  };
  socket.onopen = () => {
  console.log(«WebSocket з’єднання встановлено.»);
  };
  socket.onerror = (error) => {
  console.error(«Помилка WebSocket:», error);
  };
  }
```

кінець лістингу 3.9

Цей код створює WebSocket-з’єднання та встановлює обробник події onmessage. Коли з сервера надходить новий пакет даних, обробник розбирає JSON, знаходить відповідний ліфт у локальному масиві this.elevators та оновлює його дані. Завдяки системі реактивності Vue.js, будь-які зміни в цьому масиві миттєво і автоматично відображаються в користувацькому інтерфейсі без необхідності додаткових маніпуляцій з DOM. Такий підхід забезпечує створення ефективного та чутливого інтерфейсу для моніторингу в реальному часі.

3.3 Реалізація механізму real-time моніторингу на основі протоколу MQTT та WebSocket

Механізм моніторингу в реальному часі є ключовою функціональною перевагою розробленої системи, що відрізняє її від традиційних CRM-систем та дозволяє перейти до проактивної моделі обслуговування. Реалізація цього механізму є комплексним інженерним завданням, що вимагає забезпечення ефективної та надійної передачі даних від фізичного обладнання до інтерфейсу

користувача з мінімальною затримкою. Архітектурно цей процес розділено на два основні етапи: отримання даних від IoT-пристроїв на серверній частині за допомогою протоколу MQTT та подальша трансляція цих даних усім підключеним клієнтським додаткам за допомогою технології WebSocket.

На серверній частині (Backend) для взаємодії з MQTT-брокером необхідно реалізувати MQTT-клієнт. Враховуючи асинхронну природу фреймворку FastAPI, доцільно використовувати асинхронну бібліотеку для роботи з MQTT, наприклад asyncio-mqtt, що дозволяє уникнути блокування основного циклу подій при очікуванні повідомлень [9]. При старті додатку backend ініціює клієнт, який встановлює з'єднання з MQTT-брокером та підписується на відповідні топіки. Для гнучкості використовується ієрархічна структура топіків, наприклад elevators/+/status, де символ + є однорівневим wildcard, що дозволяє серверу слухати повідомлення про статус від усіх ліфтів одночасно.

Після отримання повідомлення викликається функція-обробник (on_message), яка розпаковує отримані дані (зазвичай у форматі JSON), валідує їх та викликає WebSocket-менеджер для подальшої трансляції. WebSocket-менеджер, реалізований у вигляді класу ConnectionManager, як і у прототипі, відстежує всі активні з'єднання від клієнтських додатків та надає метод для широкомовної розсилки повідомлень (broadcast – лістинг 3.10).

Лістинг 3.10 – Реалізація асинхронного MQTT-клієнта та інтеграція з WebSocket

```
import asyncio
import json
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
import asyncio_mqtt as mqtt
# ... (клас ConnectionManager з попередніх прикладів) ...
manager = ConnectionManager()
async def mqtt_listener():
async with mqtt.Client(«mqtt.broker.address») as client:
# Підписуємось на топік для статусів усіх ліфтів
await client.subscribe(«elevators/+/status»)
async for message in client.messages:
payload = message.payload.decode()
print(f»Отримано MQTT повідомлення: {payload}»)
```

```
# Транслюємо отримані дані усім підключеним WebSocket-клієнтам
await manager.broadcast(payload)
app = FastAPI()
@app.on_event(«startup»)
async def startup_event():
# Запускаємо слухача MQTT у фоновому режимі
  asyncio.create_task(mqtt_listener())
  @app.websocket(«/ws»)
  async def websocket_endpoint(websocket: WebSocket):
  await manager.connect(websocket)
  try:
  while True:
  await websocket.receive_text()
  except WebSocketDisconnect:
  manager.disconnect(websocket)
```

кінець лістингу 3.10

На клієнтській частині (Frontend) для отримання оновлень необхідно встановити WebSocket-з'єднання з сервером. Ця операція виконується один раз при завантаженні основного компонента додатку, наприклад, у хуці життєвого циклу mounted фреймворку Vue.js. Після встановлення з'єднання визначається обробник події onmessage, який буде викликатися щоразу, коли з сервера надходить новий пакет даних.

Логіка обробника полягає в тому, щоб розібрати отриманий JSON-рядок, знайти відповідний об'єкт даних у локальному стані додатку (наприклад, ліфт у масиві elevators) та оновити його поля. Завдяки системі реактивності, яку надає Vue.js, ця зміна в даних автоматично призводить до оновлення відповідної частини користувацького інтерфейсу (лістинг 3.11) без необхідності ручних маніпуляцій з DOM-деревом.

Лістинг 3.11 – Реалізація WebSocket-клієнта на Frontend (з прототипу)

```
// ... усередині хука mounted екземпляра Vue
connectWebSocket() {
const socket = new WebSocket(«ws://localhost:8000/ws»);
socket.onmessage = (event) => {
const updatedData = JSON.parse(event.data);
```

```
// Припускаємо, що сервер надсилає дані одного ліфта
const index = this.elevators.findIndex(e => e.id === updatedData.id);
if (index !== -1) {
// Оновлюємо об’єкт. Vue автоматично оновить інтерфейс
  this.elevators[index] = { ...this.elevators[index], ...updatedData };
  }
  };
  socket.onopen = () => {
  console.log(«WebSocket з’єднання встановлено.»);
  };
  socket.onerror = (error) => {
  console.error(«Помилка WebSocket:», error);
```

кінець лістингу 3.11

Варто зазначити, що реалізована схема взаємодії забезпечує слабку зв'язність між джерелами даних та інтерфейсом відображення. MQTT-брокер функціонує автономно, акумулюючи телеметричні дані незалежно від наявності активних сесій на веб-інтерфейсі, тоді як серверна частина виступає в ролі інтелектуального шлюзу. Вона не просто транслює «сирий» потік даних, а виконує попередню фільтрацію та валідацію, передаючи до браузера лише коректно структуровані JSON-об’єкти. Це дозволяє уникнути перевантаження клієнтського застосунку надлишковим трафіком та знизити навантаження на процесор при рендерингу інтерфейсу, що є критично важливим при масштабуванні системи до сотень підключених ліфтів.

Крім того, застосування WebSocket дозволяє реалізувати механізми автоматичного відновлення з’єднання на стороні клієнта без втрати контексту роботи програми. У випадку короткочасних розривів зв'язку, характерних для мобільного інтернету, клієнтський додаток на Vue.js може автоматично ініціювати повторне підключення, гарантуючи, що диспетчер завжди матиме актуальну інформацію про стан об'єктів без необхідності ручного перезавантаження сторінки. Такий підхід суттєво підвищує відмовостійкість системи моніторингу та покращує користувацький досвід порівняно з традиційними методами періодичних HTTP-запитів.

Така двопротокольна архітектура, де MQTT використовується для ефективного збору даних з розподілених IoT-пристроїв, а WebSocket – для миттєвої доставки цих даних до веб-інтерфейсів, є сучасним та масштабованим рішенням для побудови промислових систем моніторингу (рис. 3.3).

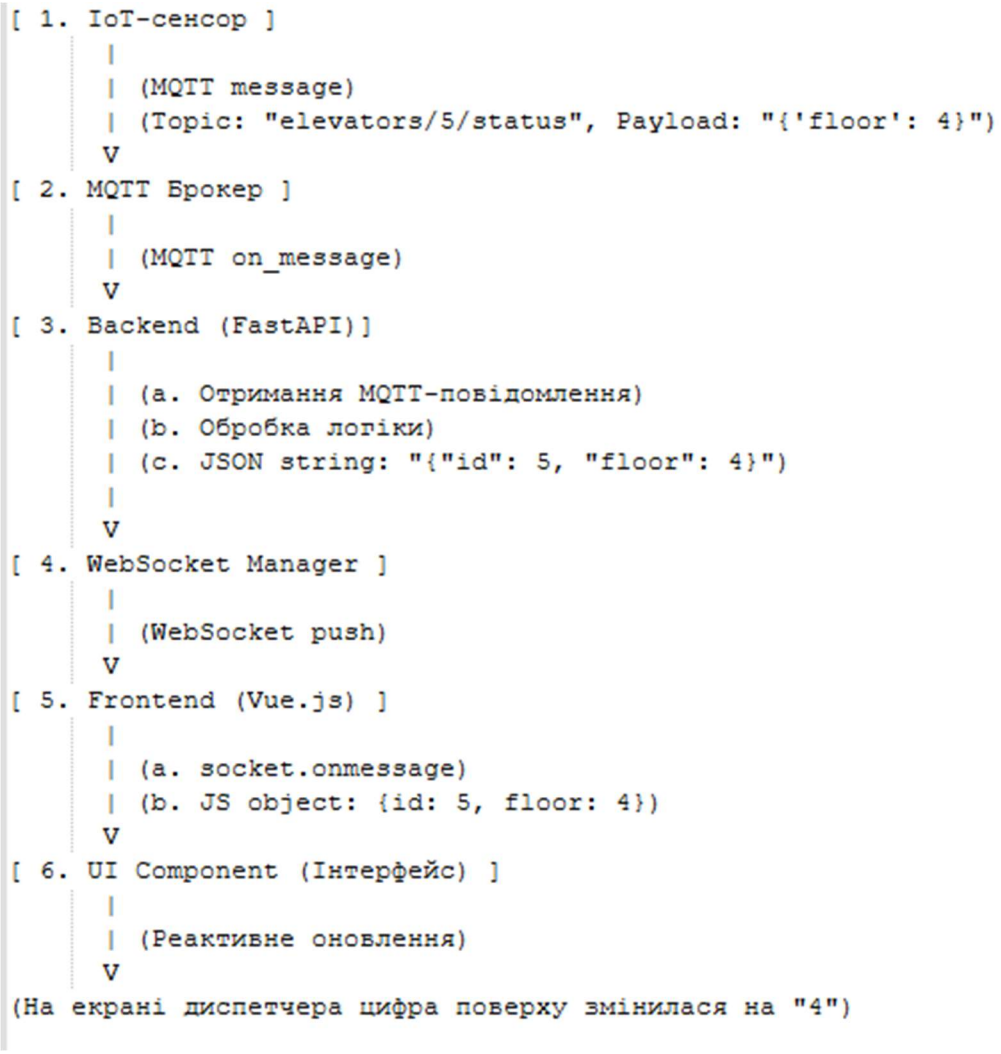

Рисунок 3.3 – Діаграма потоку даних (DFD) для механізму оновлення в реальному часі

Вона забезпечує низьку затримку, ефективне використання мережевих ресурсів та дозволяє реалізувати повноцінний інтерактивний досвід для користувача.

3.4 Методика та результати експериментального дослідження продуктивності системи

Експериментальне дослідження є невід'ємним етапом кваліфікаційної роботи, метою якого є об'єктивна кількісна оцінка ключових нефункціональних характеристик розробленої системи [21, 22]. Теоретичні переваги обраної архітектури та технологічного стеку, описані в попередньому розділі, повинні бути підтверджені практичними вимірюваннями продуктивності системи в умовах, що імітують реальне навантаження. Цей підрозділ присвячено опису методики, представленню результатів та аналізу експериментів, проведених над програмним прототипом системи «ELSECRM».

3.4.1 Мета та завдання експериментального дослідження

Основною метою експериментального дослідження є кількісна оцінка продуктивності та масштабованості розробленої серверної архітектури під впливом імітованого навантаження, а також визначення її меж та оптимальних параметрів функціонування.

Для досягнення поставленої мети було сформульовано наступні дослідницькі завдання:

– дослідити залежність використання апаратних ресурсів сервера (центрального процесора та оперативної пам'яті) від кількості одночасно підключених клієнтів (WebSocket-з'єднань);

– визначити вплив частоти надходження телеметричних даних від IoT-пристроїв (MQTT-повідомлень) на навантаження серверної частини;

– оцінити продуктивність підсистеми бази даних шляхом вимірювання часу виконання операцій запису даних при різній інтенсивності навантаження;

– проаналізувати загальну затримку доставки повідомлення в реальному часі, вимірявши час від моменту публікації повідомлення MQTT-клієнтом до моменту його отримання WebSocket-клієнтом.

3.4.2 Опис тестового середовища

Для забезпечення чистоти та відтворюваності експериментів було розгорнуто ізольоване тестове середовище. Всі компоненти системи та інструменти для тестування були запущені на єдиній віртуальній машині для усунення впливу мережевих затримок між компонентами.

Апаратна конфігурація тестового стенду:

– тип: віртуальна приватна машина (VPS);

– центральний процесор (CPU): 2 віртуальних ядра (vCPU) з тактовою частотою 2,4 ГГц;

– оперативна пам’ять (RAM): 4 ГБ;

– дискова система: 50 ГБ NVMe SSD.

Програмне забезпечення тестового стенду:

– операційна система: Ubuntu Server 22,04 LTS;

– інтерпретатор: Python 3,10;

– веб-сервер та фреймворк: Uvicorn 0,18,3; FastAPI 0,85,0;

– СУБД: PostgreSQL 14,5;

– MQTT-брокер: Mosquitto 2,0,14;

– інструменти для тестування: спеціалізовані скрипти на Python з використанням бібліотек asyncio-mqtt для генерації MQTT-трафіку та websockets для створення великої кількості клієнтських WebSocket-з’єднань.

3.4.3 Методика проведення експериментів

Для вирішення поставлених завдань було розроблено та проведено серію з трьох основних експериментів.

Експеримент 1: дослідження масштабованості за кількістю клієнтів. Метою даного експерименту є визначення, як зростає споживання ресурсів сервера при збільшенні кількості одночасно підключених користувачів:

– на сервері запускається екземпляр додатку «ELSECRM»;

– запускається скрипт-симулятор, що імітує роботу 100 IoT-пристроїв, які публікують повідомлення з фіксованою частотою 0,1 Гц (1 повідомлення кожні 10 секунд);

– запускається тестовий скрипт, який ітеративно збільшує кількість одночасних WebSocket-з’єднань з сервером. Кількість клієнтів збільшується з кроком 10, від 10 до 100;

– на кожній ітерації система працює протягом 5 хвилин для стабілізації показників. Протягом останньої хвилини кожної ітерації фіксуються середні значення завантаження CPU та використання оперативної пам’яті процесом сервера.

Експеримент 2: дослідження впливу частоти оновлень. Метою експерименту є визначення оптимальної частоти надсилання телеметричних даних, що забезпечує баланс між актуальністю інформації та навантаженням на систему. Цей експеримент повторює дослідження, що було описано в статті, але з більш детальною фіксацією показників:

– на сервері запускається екземпляр додатку «ELSECRM»;

– запускається тестовий скрипт, що підтримує 50 одночасних WebSocket-з’єднань (фіксована кількість клієнтів);

– запускається скрипт-симулятор, що імітує роботу 100 IoT-пристроїв. Частота публікації MQTT-повідомлень ітеративно змінюється: 1 Гц, 0,5 Гц, 0,2 Гц, 0,1 Гц, 0,05 Гц;

– на кожній ітерації система працює 5 хвилин, після чого фіксуються середні значення завантаження CPU та середній час виконання запиту INSERT до бази даних.

Експеримент 3: оцінка затримки доставки повідомлень (End-to-End Latency). Метою є вимірювання загального часу, необхідного для доставки повідомлення від сенсора до екрану користувача:

– скрипт-симулятор модифікується таким чином, щоб у кожне MQTT-повідомлення додавати мітку часу t1 (timestamp) у момент його відправки;

– клієнтський скрипт, що імітує frontend-додаток, при отриманні повідомлення через WebSocket фіксує час його надходження t2;

– затримка для кожного повідомлення розраховується як Latency = t2 - t1;

– експеримент проводиться при середньому навантаженні (50 клієнтів, 100 пристроїв, частота 0,1 Гц). Проводиться 1000 вимірювань, після чого розраховуються мінімальне, середнє та максимальне значення затримки.

3.4.4 Результати дослідження та їх аналіз

За результатами проведених експериментів було отримано масиви даних, що характеризують поведінку системи під різними типами навантаження. Подальший аналіз цих даних дозволяє зробити обґрунтовані висновки щодо продуктивності та масштабованості розробленої архітектури.

Результати експерименту 1: масштабованість за кількістю клієнтів. У ході першого експерименту було зібрано дані про залежність використання ресурсів сервера від кількості одночасних WebSocket-з'єднань при фіксованому навантаженні з боку IoT-пристроїв. Отримані результати зведено у таблиці 3.1.

Таблиця 3.1 – Залежність використання ресурсів від кількості WS-клієнтів

Кількість клієнтів	Навантаження CPU (%)	Використання RAM (МБ)
10	5 %	150
20	7 %	175
30	9 %	200
40	11 %	230
50	14 %	260
60	17 %	290
70	19 %	320
80	21 %	350
90	23 %	375
100	25 %	400

Аналіз отриманих числових даних свідчить про те, що система демонструє стійкість до зростання навантаження: споживання оперативної пам'яті та ресурсів процесора збільшується плавно, без аномальних стрибків. Це попередньо вказує на ефективне управління ресурсами та відсутність витоків пам'яті (memory leaks) у реалізації серверної частини, навіть при максимальній кількості активних підключень у рамках тесту.

Для наочної візуалізації цих даних було побудовано графіки залежності (рис. 3.4):

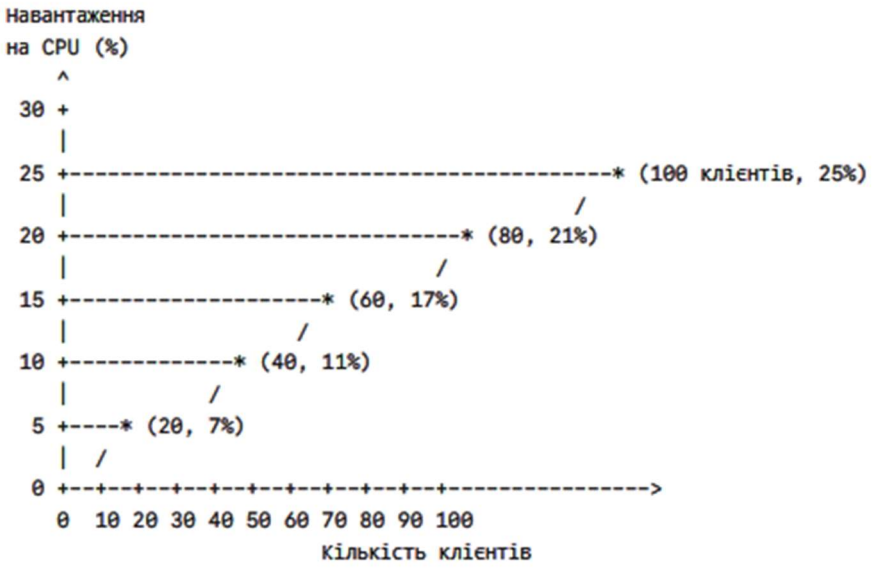

Рисунок 3.4 – Графік залежності навантаження на CPU від кількості WebSocket-з’єднань

Аналіз графіка на рисунку 3.4 показує, що спостерігається майже лінійна залежність між кількістю активних WebSocket-з’єднань та завантаженням центрального процесора. Кожне нове з’єднання додає невелике, але стале навантаження. Відсутність різких, експоненційних стрибків у досліджуваному діапазоні свідчить про високу ефективність обробки одночасних з’єднань асинхронним сервером Uvicorn та фреймворком FastAPI. Це підтверджує, що обрана архітектура є добре масштабованою за кількістю клієнтів-спостерігачів, і система здатна підтримувати сотні одночасних підключень на відносно скромній апаратній конфігурації без суттєвої деградації продуктивності.

Аналогічна лінійна залежність спостерігалася і для використання оперативної пам’яті.

Результати експерименту 2: вплив частоти оновлень. Другий експеримент був спрямований на дослідження залежності навантаження на систему від інтенсивності надходження даних від IoT-пристроїв (у таблиці 3.2, на рисунку 3.5 та на рисунку 3.6):

Таблиця 3.2 – Залежність навантаження на систему від частоти оновлень

Частота (Гц)	Навантаження CPU (%)	Середній час запису в БД (мс)
0.05	8 %	5
0.1	12 %	8
0.2	20 %	15
0.5	45 %	40
1.0	85 %	110

Аналіз графіка на рисунку 3.5 демонструє нелінійну залежність між частотою надходження MQTT-повідомлень та завантаженням CPU. У діапазоні від 0,05 Гц до 0,2 Гц навантаження зростає плавно, проте після перевищення порогу в 0,2 Гц (одне повідомлення кожні 5 секунд) спостерігається різке, експоненційне зростання навантаження. Це свідчить про те, що система починає досягати межі своєї пропускної здатності по обробці вхідних даних, і час, що витрачається на обробку одного повідомлення, стає співмірним з інтервалом між їх надходженням.

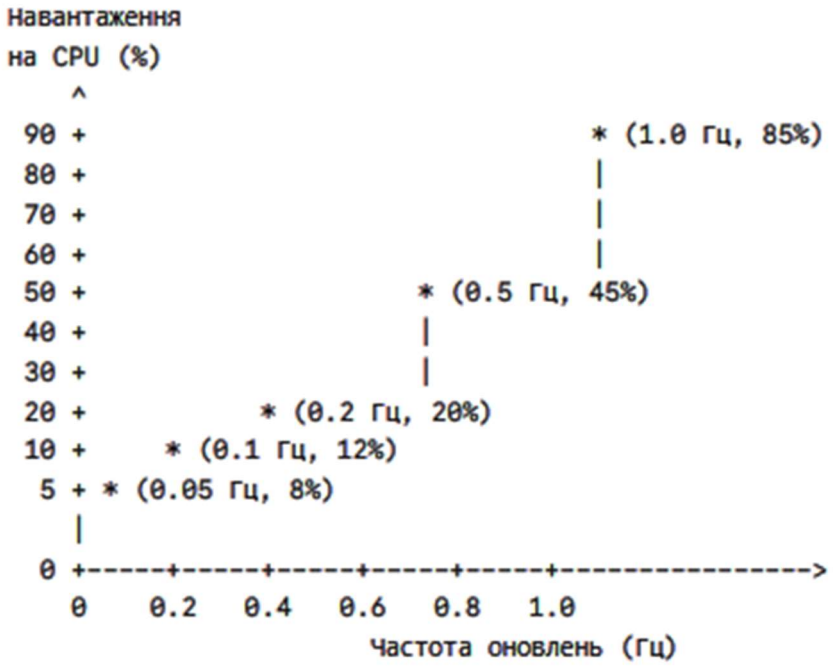

Рисунок 3.5 – Графік залежності навантаження на CPU від частоти оновлень

Графік на рисунку 3.6 підтверджує цю тенденцію: середній час запису до бази даних також починає суттєво зростати при високій частоті, що вказує на те,

що СУБД стає вузьким місцем. Результати цього експерименту дозволяють зробити обґрунтований висновок, що для стабільної роботи системи на даній конфігурації оптимальною є частота оновлень в діапазоні 0,05-0,1 Гц (1 повідомлення кожні 10-20 секунд), що забезпечує достатню актуальність даних без ризику перевантаження системи.

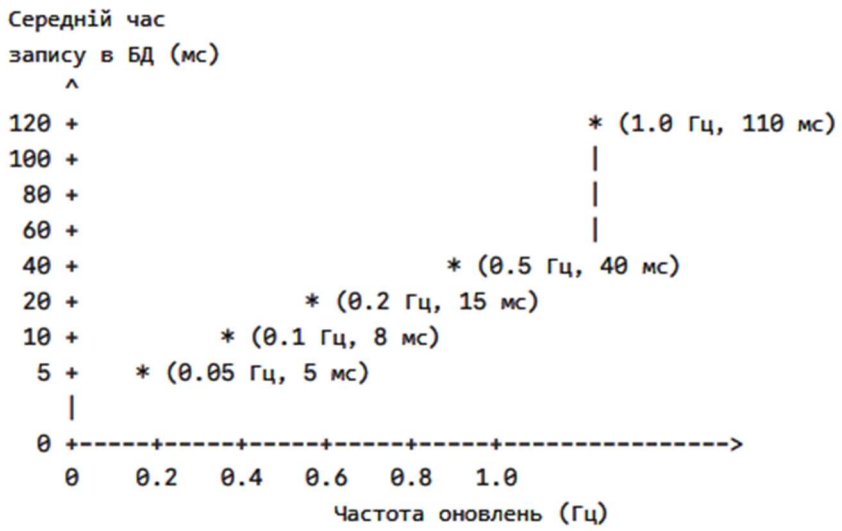

Рисунок 3.6 – Графік залежності часу запису в БД від частоти оновлень

Результати експерименту 3: оцінка затримки доставки повідомлень. Третій експеримент дозволив оцінити одну з ключових характеристик системи реального часу – загальну затримку доставки повідомлення (таблиця 3.3 та на рисунку 3.7).

Таблиця 3.3 – Результати вимірювання затримки доставки повідомлень (End-to-End Latency)

Параметр	Значення (мс)
Мінімальна затримка	45
Середня затримка	85
Максимальна затримка	250
Стандартне відхилення	30

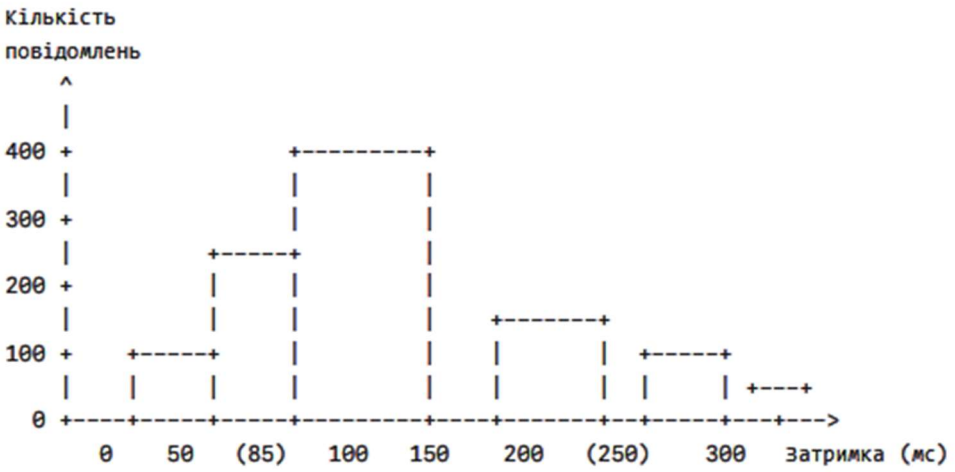

Рисунок 3.7 – Гістограма розподілу значень затримки доставки повідомлень

Середнє значення затримки менше 100 мілісекунд є відмінним показником для системи моніторингу, що гарантує практично миттєве сповіщення диспетчера про критичні події. Гістограма розподілу показує, що переважна більшість повідомлень доставляється із затримкою, близькою до середнього значення. Наявність невеликої кількості значень у правій частині «хвоста» розподілу (максимальна затримка) є очікуваною і може бути спричинена короткочасними піками навантаження на сервер або мережевими флуктуаціями. Загалом, результати цього експерименту підтверджують, що архітектура на базі MQTT та WebSocket є повністю придатною для вирішення завдань моніторингу в реальному часі.

Описано програмну реалізацію компонентів системи: серверну частину на FastAPI, клієнтську на Vue.js та механізм моніторингу через MQTT і WebSocket. Проведене експериментальне дослідження підтвердило правильність архітектурних рішень, високу масштабованість та стабільність системи під навантаженням. Отримані результати доводять придатність прототипу для моніторингу в реальному часі, що свідчить про повне виконання завдань розділу.

ВИСНОВКИ

У кваліфікаційній магістерській роботі було вирішено актуальну науково-технічну задачу підвищення ефективності процесів управління технічним обслуговуванням ліфтового обладнання шляхом розробки та дослідження архітектури спеціалізованої IoT-орієнтованої CRM-системи. Проведене дослідження дозволило отримати комплексні результати, що підтверджують досягнення поставленої мети та виконання всіх завдань роботи.

Проведено глибокий аналіз предметної області, що дозволило всебічно обґрунтувати актуальність обраної теми та чітко окреслити науково-технічну проблему. Було встановлено, що ключові бізнес-процеси в більшості українських сервісних компаній, що займаються обслуговуванням ліфтів, характеризуються високим ступенем ручних операцій, фрагментацією інформаційних потоків та використанням застарілих, переважно паперових, методів управління.

Виконано огляд та порівняльний аналіз існуючих на ринку програмних засобів управління підтвердив виявлену проблему та дозволив сформулювати вимоги до розроблюваної системи. Було встановлено, що універсальні CRM-системи, хоча й автоматизують загальні процеси взаємодії з клієнтами, не враховують специфіку обліку та обслуговування складного технічного обладнання, що вимагає значних фінансових та часових витрат на їх кастомізацію. Спеціалізовані закордонні аналоги, в свою чергу, є дорогими, орієнтованими на закриті пропрієтарні екосистеми та не завжди адаптованими до українських нормативних стандартів ведення документації. Таким чином, аналіз підтвердив наявність вільної ринкової та технологічної ніші для розробки вітчизняного продукту. Було зроблено обґрунтований висновок, що оптимальним вирішенням виявленої проблеми є створення інтегрованої інформаційної системи, що поєднує функціонал CRM та технології Інтернету речей. Такий підхід дозволяє не лише автоматизувати існуючі бізнес-процеси, а й трансформувати саму модель обслуговування, переходячи від реактивної до проактивної стратегії, що відповідає сучасним тенденціям цифрової

трансформації бізнесу. Цим було повністю виконано перше та друге завдання кваліфікаційної роботи.

Спроектовано та обґрунтовано комплексну архітектуру інформаційно-керуючої системи «ELSECRM». Процес проєктування базувався на сформульованих функціональних та нефункціональних вимогах, а кожне архітектурне рішення приймалося після порівняльного аналізу альтернатив з метою досягнення оптимального балансу між продуктивністю, гнучкістю та складністю реалізації.

Розроблено програмний прототип системи «ELSECRM». Цим було вирішено четверте та п’яте завдання кваліфікаційної роботи. В рамках програмної реалізації було успішно імплементовано ключові модулі системи. Було розроблено серверну частину на базі фреймворку FastAPI, що надає REST API для управління основними сутностями системи, та реалізовано клієнтську частину у вигляді односторінкового застосунку з інтерактивною панеллю моніторингу на базі фреймворку Vue.js. Центральним елементом реалізації став наскрізний механізм моніторингу в реальному часі, що забезпечує отримання даних від імітованих IoT-пристроїв за протоколом MQTT та їх миттєву трансляцію до інтерфейсу користувача за допомогою технології WebSocket.

Для об’єктивної оцінки нефункціональних характеристик розробленого прототипу було проведено комплексне експериментальне дослідження продуктивності системи. Дослідження проводилося на ізольованому тестовому стенді в умовах, що імітували реальне навантаження. Отримані кількісні результати дозволили зробити обґрунтовані висновки щодо ефективності та масштабованості обраної архітектури. По-перше, було експериментально підтверджено, що система демонструє високу ефективність та лінійну масштабованість при збільшенні кількості одночасно підключених клієнтів. Аналіз залежності використання ресурсів сервера від кількості активних WebSocket-з’єднань показав, що навантаження на центральний процесор та споживання оперативної пам’яті зростають плавно і передбачувано. Це свідчить про відсутність архітектурних вузьких місць та ефективність обробки

одночасних з’єднань асинхронним сервером, що підтверджує правильність вибору технологічного стеку. По-друге, в результаті дослідження впливу частоти надходження телеметричних даних було виявлено нелінійну залежність навантаження на систему та визначено оптимальні параметри її функціонування. Було встановлено, що при збільшенні частоти MQTT-повідомлень вище певного порогу (0,2 Гц в умовах тестового стенду) навантаження на центральний процесор та час запису до бази даних починають зростати експоненційно. Це дозволило зробити науково обґрунтований висновок, що для стабільної роботи системи та раціонального використання ресурсів оптимальною є частота оновлення даних в діапазоні від 0,05 до 0,1 Гц (одне повідомлення кожні 10-20 секунд). Цей результат є важливою практичною рекомендацією для налаштування та експлуатації подібних систем моніторингу. По-третє, вимірювання загальної затримки доставки повідомлення від імітованого сенсора до інтерфейсу користувача показало високу продуктивність real-time механізму. Середнє значення затримки склало менше 100 мілісекунд, що є відмінним показником для систем моніторингу та гарантує практично миттєве сповіщення диспетчера про критичні події. Таким чином, експериментальне дослідження повністю підтвердило, що розроблений програмний прототип не лише реалізує весь необхідний функціонал, але й відповідає високим вимогам до продуктивності та надійності.

Проведено комплексну оцінку ефективності розробленої системи та визначено її практичне значення, що стало вирішенням шостого завдання кваліфікаційної роботи. Ефективність системи оцінювалася шляхом аналізу потенційного впливу її впровадження на ключові показники ефективності сервісної служби та на оптимізацію використання ресурсів.

Практичне значення роботи полягає в тому, що розроблений прототип є не лише теоретичною моделлю, а й функціональним інструментом, що дозволяє досягти значних кількісних та якісних покращень у робочих процесах. Кількісна оцінка ефективності демонструє суттєвий економічний потенціал. Наприклад, автоматизація процесу реагування на аварійні ситуації дозволяє скоротити

середній час від моменту виявлення несправності до призначення виконавця з 5-15 хвилин (при ручній обробці) до менш ніж однієї хвилини. Це, у свою чергу, безпосередньо впливає на скорочення загального часу простою обладнання. Автоматизація планування та генерації звітної документації дозволяє вивільнити значну частину робочого часу як диспетчерського, так і технічного персоналу. За оцінками, диспетчер може зекономити до 25 % робочого часу, який раніше витрачався на телефонні дзвінки та заповнення паперових журналів, а механік – до 15 % часу, що витрачався на заповнення актів та звітів вручну після кожного виїзду.

Окрім прямого економічного ефекту, впровадження системи має низку важливих якісних переваг. По-перше, значно підвищується прозорість усіх сервісних процесів як для керівництва компанії, так і для клієнтів. Керівництво отримує доступ до об’єктивної аналітики в реальному часі, що дозволяє приймати більш обґрунтовані управлінські рішення. Клієнти, в свою чергу, отримують доступ до особистого кабінету з повною історією обслуговування їхнього обладнання, що підвищує їхню лояльність та довіру до сервісної компанії. По-друге, мінімізується вплив людського фактору: зменшується кількість помилок при заповненні документів, виключається ризик втрати паперових звітів та унеможливлюється пропуск планових робіт через забудькуватість. По-третє, покращуються умови праці для персоналу: механіки та диспетчери звільняються від рутинних, монотонних операцій, що дозволяє їм зосередитися на виконанні більш складних технічних та координаційних завдань. Зрештою, перехід до проактивної моделі обслуговування, що стає можливим завдяки IoT-моніторингу, безпосередньо впливає на підвищення загального рівня безпеки експлуатації ліфтового обладнання.

На завершення, на основі проведеного дослідження та отриманих результатів, було сформульовано практичні рекомендації щодо впровадження подібних інформаційних систем, а також визначено найбільш перспективні напрямки подальшого розвитку розробленої системи та відповідних технологій.

Успішне впровадження комплексної інформаційної системи є не лише технічним, але й організаційним завданням, що вимагає ретельного планування. На основі досвіду, отриманого при розробці прототипу, можна сформулювати наступні практичні рекомендації:

– провести детальний аудит існуючих бізнес-процесів. Перед початком впровадження необхідно чітко формалізувати та, за потреби, оптимізувати існуючі робочі процеси. Важливо задокументувати потоки інформації, зони відповідальності та критерії прийняття рішень. Це дозволить уникнути автоматизації хаосу та максимально адаптувати систему під реальні потреби компанії;

– розпочати з пілотного проєкту. Впровадження слід починати з невеликої, контрольованої групи об’єктів (наприклад, 10-15 ліфтів) та користувачів (1-2 диспетчери, 2-3 механіки). Це дозволить відпрацювати основні сценарії використання системи в реальних умовах, провести навчання ключових співробітників, виявити можливі проблеми та зібрати зворотний зв’язок для доопрацювання системи перед її повномасштабним розгортанням;

– забезпечити поетапне оснащення обладнання IoT-сенсорами. Обладнання всього парку ліфтів сенсорами є капіталомістким процесом. Рекомендується починати з найбільш критичних, застарілих або проблемних об’єктів, де віддалений моніторинг принесе найбільшу користь. Поступове розширення покриття дозволить рівномірно розподілити інвестиції та накопичувати досвід експлуатації IoT-рішень;

– приділити особливу увагу навчанню персоналу та управлінню змінами. Ефективність системи напряму залежить від того, наскільки коректно та повноцінно її будуть використовувати співробітники. Необхідно розробити чіткі інструкції для кожної ролі, провести практичне навчання та пояснити персоналу переваги нової системи. Важливо працювати із можливим супротивом змінам та демонструвати, як система полегшує, а не ускладнює їхню щоденну роботу.

Перспективи розвитку технологій у сфері обслуговування ліфтів та подальшого вдосконалення системи.

Сфера управління технічним обслуговуванням продовжує стрімко розвиватися під впливом глобальних технологічних трендів, що відкриває широкі перспективи для подальшого вдосконалення розробленої системи «ELSECRM». Є різні напрямки подальших досліджень та розробки, серед яких варто виділити основні.

Розробка повноцінного мобільного додатку. Створення нативних мобільних додатків для платформ Android та iOS, зокрема для ролі «Механік», дозволить значно підвищити зручність роботи в польових умовах. Такий додаток зможе забезпечити можливість роботи в режимі офлайн (з подальшою синхронізацією даних при появі зв'язку), використовувати апаратні можливості смартфонів (камеру для сканування QR-кодів або фотофіксації несправностей, GPS для автоматичного відстеження місцезнаходження) та надсилати push-сповіщення про нові завдання.

Інтеграція алгоритмів машинного навчання для предиктивного аналізу. Зібрані за допомогою IoT-сенсорів часові ряди даних є ідеальною основою для побудови моделей машинного навчання. Використовуючи такі моделі, як рекурентні нейронні мережі [23] або градієнтний бустинг, можна прогнозувати ймовірність відмови окремих вузлів ліфта (наприклад, двигуна, редуктора, дверних механізмів) та автоматично створювати завдання на превентивне обслуговування ще до того, як несправність стане очевидною. Це є наступним кроком еволюції від проактивного до предиктивного сервісу.

Розширення інтеграційних можливостей. Подальший розвиток системи може включати розробку відкритого API для інтеграції з іншими корпоративними системами. Наприклад, інтеграція з бухгалтерськими системами дозволить автоматизувати процес виставлення рахунків на основі даних про виконані роботи, а інтеграція з ERP-системами – оптимізувати управління складом запасних частин та логістику.

Впровадження елементів доповненої реальності (Augmented Reality, AR). Мобільний додаток для механіків може бути доповнений функціоналом доповненої реальності. Навівши камеру смартфона або спеціальних AR-окулярів

на вузол ліфта, технік зможе отримувати контекстні підказки, покрокові інструкції з ремонту, візуалізацію технічних параметрів з сенсорів або навіть віддалену відео-консультацію з більш досвідченим інженером.

Реалізація цих напрямків дозволить перетворити розроблену систему на комплексну цифрову платформу, що відповідає найвищим стандартам концепції Industry 4.0 та забезпечує максимальну ефективність, безпеку та надійність у сфері обслуговування ліфтового господарства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vial G. Understanding digital transformation: A review and a research agenda. The Journal of Strategic Information Systems. 2021. Vol. 30, no. 2. P. 101666.

2. Руденко М. В. Цифрова трансформація підприємств сфери послуг: монографія. Київ: Кондор, 2021. 240 с.

3. Мацько Ю. А. Архітектурні особливості CRM-системи роботи ліфтового обладнання. Сучасні інформаційні технології та системи в управлінні: матеріали VI Міжнар. наук.-практ. конф. молодих вчен., аспірантів і студентів, м. Київ, 10 квіт. 2025 р. Київ, 2025. С. 34-35.

4. Мацько Ю. А., Христинець Н. А. Архітектурно-програмна модель IoT-системи моніторингу та управління технічним обслуговуванням ліфтів. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2025. № 61. С. 14-20.

5. Григорчук Т. М. Інформаційні системи в менеджменті: навч. посіб. Івано-Франківськ: ІФНТУНГ, 2020. 315 с.

6. Кузьменко О. В., Бойко А. О. Аналіз сучасних CRM-систем та перспективи їх впровадження на українських підприємствах. Економіка та суспільство. 2022. № 36. URL: https://surl.li/jynngf (дата звернення: 20.09.2025).

7. Карпунь О. В. Інтернет речей (IoT): технології, протоколи та застосування: навч. посіб. Київ: КПІ ім. Ігоря Сікорського, 2021. 156 с.

8. Sinha S. Beginning MQTT programming with Python: build reliable IoT solutions. New York: Apress, 2023. 250 p.

9. Naik N. Securing IoT communication using MQTT protocol. Internet of Things. 2020. Vol. 11. P. 100250.

10. KPI для сервісних компаній: як вимірювати ефективність. Менеджмент та підприємництво: тренди розвитку. 2021. Вип. 15. С. 88-95.

11. Лавріщева К. М. Програмна інженерія: підручник. Київ: ВПЦ «Київський університет», 2020. 480 с.

12. Montañez Á. Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python. Birmingham: Packt Publishing, 2022. 402 p.

13. Voron G. Building Data Science Applications with FastAPI. Birmingham: Packt Publishing, 2021. 436 p.

14. Minnick J. Beginning Vue 3 Development: User Interfaces with Typescript and Modern JavaScript Tools. New York: Apress, 2022. 350 p.

15. Chopin B., Martineau L. Vue.js 3 Design Patterns and Best Practices: Build scalable and maintainable Vue applications using the latest rendering patterns and techniques. Birmingham: Packt Publishing, 2021. 466 p.

16. Річардс М., Форд Н. Фундаментальний підхід до програмної архітектури / пер. з англ. О. Коломійцев. Харків: Фабула, 2022. 448 с.

17. Wang V. The Definitive Guide to HTML5 WebSocket. New York: Apress, 2020. 200 p.

18. Drake J. D., Worsley J. C. Practical PostgreSQL. Sebastopol: O'Reilly Media, 2020. 600 p.

19. Фергюсон Д. Безпека веб-застосунків: сучасні стратегії захисту. Київ: Наш Формат, 2021. 320 с.

20. Герасименко О. М. Економічне обґрунтування ІТ-проектів: методичні рекомендації. Львів: Видавництво Львівської політехніки, 2022. 45 с.

21. Персіваль Г., Грегорі Б. Розробка веб-застосунків на Python за допомогою патернів TDD / пер. з англ. Харків: Ранок, 2021. 384 с.

22. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering - Software testing - Part 1: Concepts and definitions. Geneva: ISO, 2022. 112 p.

23. James G. et al. An Introduction to Statistical Learning: with Applications in Python. Springer, 2023. 612 p.