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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизована система підтримки клієнтів інтернет-магазину електроніки»

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

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

_____ Віктор СОКОЛ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Віктор СОКОЛ

Керівник: Ігор ГАМАНЮК
ст. викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Соколу Віктору Івановичу

1. Тема кваліфікаційної роботи: «Автоматизована система підтримки клієнтів інтернет-магазину електроніки»
керівник кваліфікаційної роботи ст. викладач Ігор ГАМАНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про організацію клієнтської підтримки в інтернет-магазинах, методи автоматизації обробки звернень клієнтів, підходи до категоризації запитів, контролю статусів звернень, збереження історії комунікації, формування бази знань та звітності, опис принципів побудови клієнт-серверних web-застосунків та технічна документація.

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз предметної галузі підтримки клієнтів інтернет-магазину електроніки.

2. Аналіз існуючих програмних засобів для автоматизації клієнтської підтримки, обробки звернень, роботи з тікетами, базами знань та звітністю.

3. Огляд засобів реалізації web-застосунку ElectroCare для автоматизації підтримки клієнтів інтернет-магазину електроніки.

4. Проєктування структури web-застосунку, бази даних, основних модулів.

5. Програмна реалізація та опис функціонування web-застосунку ElectroCare.

6. Тестування web-застосунку.

7. Апробація результатів дослідження.

5. Перелік демонстраційних матеріалів: *презентація*

1. Логічна структура системи.
2. Діаграма варіантів використання web-застосунку ElectroCare.
3. Діаграма доменної моделі системи.
4. Діаграми діяльності основних процесів системи.
5. Діаграма класів web-застосунку ElectroCare.
6. Діаграма компонентів web-застосунку.
7. Діаграма розгортання web-застосунку.
8. Проєктування бази даних.
9. Екранні форми web-застосунку.
10. Схема та результати тестування програмного забезпечення.
11. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури з теми клієнтської підтримки, електронної комерції та автоматизації обробки звернень	23.02–05.03.2026	
2	Аналіз та дослідження існуючих аналогів програмного забезпечення для підтримки клієнтів інтернет-магазинів	05.03–10.03.2026	
3	Аналіз предметної галузі підтримки клієнтів інтернет-магазину електроніки та визначення вимог	11.03–15.03.2026	
4	Огляд засобів реалізації web-застосунку ElectroCare для автоматизації підтримки клієнтів	16.03–21.03.2026	
5	Проєктування web-застосунку ElectroCare, логічної структури системи, бази даних та UML-діаграм	22.03–06.04.2026	
6	Програмна реалізація серверної та клієнтської частин	07.04–23.04.2026	
7	Тестування та оформлення результатів тестування	24.04–04.05.2026	
8	Оформлення роботи: вступ, висновки, реферат, список джерел та додатки	06.05–12.05.2026	
9	Розробка демонстраційних матеріалів	13.05–17.05.2026	
10	Попередній захист роботи	18.05–22.05.2026	

Здобувач вищої освіти _____ Віктор СОКОЛ
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор., 3 табл., 27 рис., 20 джерел.

Мета роботи – спрощення підтримки клієнтів шляхом впровадження інтернет-магазину електроніки.

Об'єкт дослідження – процеси підтримки клієнтів інтернет-магазину електроніки.

Предмет дослідження – засоби, методи та технології реалізації автоматизації підтримки клієнтів інтернет-магазину електроніки.

Короткий зміст роботи: У роботі проаналізовано предметну галузь підтримки клієнтів інтернет-магазину електроніки та визначено основні проблеми, пов'язані з великою кількістю звернень, ручною обробкою запитів, відсутністю єдиної історії комунікації, складністю контролю статусів, дублюванням типових відповідей та обмеженими можливостями аналізу якості обслуговування. Проаналізовано існуючі програмні засоби для автоматизації підтримки клієнтів: Zendesk, Freshdesk, Intercom, Help Scout та HubSpot Service Hub.

Розроблено структуру web-застосунку ElectroCare та визначено його ключові функціональні можливості, зокрема: реєстрацію та авторизацію користувачів, розмежування ролей клієнта, оператора підтримки та адміністратора, створення і редагування звернень, перегляд статусу звернення, збереження історії повідомлень, категоризацію запитів, зміну статусів, використання шаблонів відповідей, керування довідковою інформацією, перегляд звітів та адміністрування користувачів. У роботі використано Java та Spring Boot для реалізації серверної частини, PostgreSQL для зберігання даних, React і TypeScript для створення клієнтського інтерфейсу, а також REST API для організації взаємодії між клієнтською та серверною частинами системи.

Сферою використання web-застосунку є організація клієнтської підтримки в інтернет-магазинах електроніки, які потребують зручного інструменту для

централізованої обробки звернень, контролю статусів, збереження історії комунікації, використання бази знань та аналізу роботи служби підтримки.

КЛЮЧОВІ СЛОВА: КЛІЄНТСЬКА ПІДТРИМКА, ІНТЕРНЕТ-МАГАЗИН, ЕЛЕКТРОННА КОМЕРЦІЯ, ELECTROCARE, WEB-ЗАСТОСУНОК, ЗВЕРНЕННЯ КЛІЄНТА, ТІКЕТ, БАЗА ЗНАНЬ, JAVA, SPRING BOOT, POSTGRESQL, REACT, TYPESCRIPT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 11
ВСТУП 13
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 16
1.1 Загальна характеристика предметної галузі 16
1.2 Актуальність створення системи підтримки клієнтів 17
1.3 Цільова аудиторія та основні користувачі системи 18
1.4 Основні функції та бізнес-логіка системи 19
1.5 Способи вирішення задачі та роль інфокомунікаційних засобів 21
1.6 Основні вимоги до програмного продукту 22
1.7 Аналіз проблем і недоліків існуючих підходів 23
2 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПІДТРИМКИ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ЕЛЕКТРОНІКИ 25
2.1 Zendesk 26
2.2 Freshdesk 27
2.3 Intercom 28
2.4 Help Scout 30
2.5 Порівняльний аналіз програмних засобів 31
3 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ПІДТРИМКИ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ЕЛЕКТРОНІКИ 34
3.1 Логічна архітектура 34
3.2 Мова програмування Java 35
3.3 Spring Boot та Spring Framework 36
3.4 REST API та формат обміну даними JSON 36
3.5 PostgreSQL 37
3.6 React та TypeScript 38
4 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ ELECTROCARE 40
4.1 Логічна структура системи 41
4.2 Діаграма варіантів використання web-застосунку ElectroCare 43
4.3 Діаграма доменної моделі web-застосунку ElectroCare 44
4.4 Діаграми діяльності 47
4.4.1 Діаграма діяльності процесу створення звернення клієнтом 48
4.4.2 Діаграма діяльності процесу обробки звернення оператором 49
4.4.3 Діаграма діяльності процесу керування користувачами адміністратором 50
4.5 Діаграма класів web-застосунку ElectroCare 51
4.6 Діаграма компонентів web-застосунку ElectroCare 53
4.7 Діаграма розгортання web-застосунку ElectroCare 54

4.8 Проєктування бази даних 56
5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ ELECTROCARE 59
5.1 Загальна характеристика реалізованого web-застосунку 59
5.2 Реалізація серверної частини 60
5.3 Реалізація клієнтської частини 62
5.4 Опис основних сторінок web-застосунку 63
5.4.1 Сторінка “Огляд” 63
5.4.2 Сторінка “Всі задачі” для адміністратора 64
5.4.3 Сторінка “Мої задачі” для оператора 65
5.4.4 Сторінка “Мої задачі” для клієнта 66
5.4.5 Сторінка “Деталі звернення” 67
5.4.6 Сторінка “База знань” 68
5.4.7 Сторінка “Профіль” 69
5.4.8 Сторінка “Користувачі” 70
5.5 Реалізація унікальних функцій системи 70
5.5.1 Smart Triage Radar 70
5.5.2 Operator Load Balancer 71
5.5.3 SLA Radar 71
5.5.4 Activity Log 72
5.5.5 Рольовий доступ 72
5.6 Тестування програмного забезпечення 73
5.7 Тест-кейси web-застосунку ElectroCare 74
5.8 Результати тестування 76
ВИСНОВКИ 78
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 83
ДОДАТОК В. ЛИСТИНГИ ОСНОВНИХ МОДУЛІВ 95

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що забезпечує взаємодію між різними програмними компонентами або сервісами.

CRUD – базові операції роботи з даними: створення, читання, оновлення та видалення записів.

CSS – мова стилів, що використовується для оформлення зовнішнього вигляду web-сторінок.

DB – база даних, призначена для збереження структурованої інформації системи.

DTO – об’єкт передавання даних між клієнтською та серверною частинами застосунку.

ERD – діаграма «сутність-зв’язок», що використовується для проєктування структури бази даних.

Git – система контролю версій, що дозволяє відстежувати зміни у програмному коді.

GitHub – web-сервіс для зберігання репозиторіїв, спільної роботи з кодом та керування версіями проєкту.

Helpdesk – програмна система для приймання, обробки, контролю та аналізу звернень користувачів.

HTML – мова розмітки гіпертексту, що використовується для створення структури web-сторінок.

HTTP – протокол передавання даних у мережі Інтернет.

HTTPS – захищена версія протоколу HTTP, що використовується для безпечного обміну даними.

IDE – інтегроване середовище розробки програмного забезпечення.

JPA – специфікація Java для роботи з реляційними базами даних через об’єктну модель.

JSON – текстовий формат обміну даними між клієнтською та серверною частинами системи.

JWT – формат токена, що може використовуватися для авторизації та перевірки прав доступу користувача.

ORM – технологія об’єктно-реляційного відображення, яка забезпечує зв’язок між об’єктами програми та таблицями бази даних.

PostgreSQL – об’єктно-реляційна система керування базами даних.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

REST – архітектурний стиль побудови web-сервісів для обміну даними між клієнтом і сервером.

SPA – односторінковий web-застосунок, у якому основна взаємодія з користувачем відбувається без повного перезавантаження сторінки.

SQL – мова структурованих запитів для роботи з реляційними базами даних.

Spring Boot – фреймворк для швидкої розробки серверних web-застосунків мовою Java.

Ticket – клієнтське звернення, яке фіксується в системі та проходить певний цикл обробки.

TypeScript – мова програмування на основі JavaScript із підтримкою статичної типізації.

UI – користувацький інтерфейс програмного забезпечення.

UML – уніфікована мова моделювання, що використовується для побудови діаграм програмної системи.

UX – користувацький досвід під час взаємодії з програмним продуктом.

Web-застосунок – програмне забезпечення, доступ до якого здійснюється через web-браузер.

База знань – структурований розділ системи з відповідями на типові питання клієнтів.

ВСТУП

Актуальність теми. Сфера електронної комерції активно розвивається, а інтернет-магазини електроніки є одним із найбільш затребуваних напрямів онлайн-торгівлі. Користувачі все частіше купують смартфони, ноутбуки, побутову техніку, комплектуючі, аксесуари та інші електронні товари через web-сервіси. Разом із цим зростає кількість звернень до служби підтримки, оскільки покупці потребують консультацій щодо характеристик товарів, сумісності пристроїв, оформлення замовлень, способів оплати, доставки, гарантійного обслуговування, повернення та обміну продукції [1-3].

Для інтернет-магазину електроніки якісна підтримка клієнтів має важливе значення, оскільки від швидкості та точності відповіді часто залежить рішення користувача про покупку. Якщо звернення обробляються вручну або через різні неузгоджені канали комунікації, виникають типові проблеми: втрата повідомлень, дублювання відповідей, складність пошуку історії спілкування, затримки в обробці запитів і відсутність чіткого контролю статусів. У результаті збільшується навантаження на операторів підтримки, а якість обслуговування клієнтів знижується [4; 5].

Одним зі способів вирішення цієї проблеми є створення автоматизованої системи підтримки клієнтів, яка дозволяє централізовано приймати, зберігати, категоризувати та опрацьовувати звернення. Така система повинна забезпечувати клієнту можливість створити звернення, переглянути його статус, отримати відповідь та ознайомитися з довідковою інформацією. Оператор підтримки має отримати інструменти для перегляду списку звернень, фільтрації запитів, зміни статусів, використання шаблонів відповідей і ведення історії комунікації. Адміністратор, у свою чергу, повинен мати розширені права для керування користувачами, категоріями, шаблонами відповідей, довідковими матеріалами та звітністю [4; 5].

Розробка web-застосунку ElectroCare є актуальною, оскільки він орієнтований саме на потреби інтернет-магазину електроніки та дозволяє об’єднати основні процеси клієнтської підтримки в одному середовищі. На відміну від універсальних комерційних helpdesk-систем, такий застосунок може бути адаптований під конкретні бізнес-процеси магазину без надлишкового функціоналу. Це дає можливість спростити роботу операторів, скоротити час обробки звернень, зменшити ризик втрати інформації та підвищити прозорість взаємодії між клієнтом і службою підтримки [4; 5].

Мета роботи – спрощення підтримки клієнтів шляхом впровадження інтернет-магазину електроніки.

Об’єкт дослідження – процеси підтримки клієнтів інтернет-магазину електроніки.

Предмет дослідження – засоби, методи та технології реалізації автоматизації підтримки клієнтів інтернет-магазину електроніки.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну галузь підтримки клієнтів інтернет-магазину електроніки.
2. Визначити основні проблеми ручної обробки клієнтських звернень.
3. Провести аналіз існуючих програмних засобів для автоматизації клієнтської підтримки.
4. Визначити функціональні та нефункціональні вимоги до web-застосунку.
5. Обґрунтувати вибір технологій для реалізації клієнтської, серверної частини та бази даних.
6. Спроєктувати логічну структуру системи, доменну модель, базу даних та основні UML-діаграми.
7. Реалізувати основні модулі web-застосунку ElectroCare.
8. Провести тестування програмного забезпечення та оцінити результати перевірки.

Методи дослідження. У роботі використовуються методи аналізу предметної галузі, порівняльного аналізу програмних аналогів, об’єктно-орієнтованого проєктування, UML-моделювання, проєктування бази даних, розробки клієнт-серверного web-застосунку та тестування програмного забезпечення [4; 5].

Наукова новизна одержаних результатів полягає у формуванні структури web-застосунку ElectroCare, який поєднує основні функції підтримки клієнтів інтернет-магазину електроніки: роботу зі зверненнями, категоризацію запитів, контроль статусів, історію повідомлень, шаблони відповідей, базу знань, звітність і розмежування ролей користувачів.

Практична значущість одержаних результатів полягає в тому, що розроблений web-застосунок може бути використаний для організації роботи служби підтримки інтернет-магазину електроніки. Його впровадження дозволяє централізувати обробку звернень, скоротити час відповіді клієнтам, зберігати історію взаємодії, контролювати роботу операторів та аналізувати якість обслуговування. У подальшому систему можна розширити за рахунок інтеграції з CRM, Telegram-ботом, онлайн-чатом або AI-модулем для автоматичного формування відповідей [4; 5].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Загальна характеристика предметної галузі

Електронна комерція є одним із найбільш активних напрямів розвитку сучасних інформаційних технологій. Значна частина товарів і послуг сьогодні реалізується через web-сервіси, маркетплейси та інтернет-магазини. Окреме місце в цій сфері займають інтернет-магазини електроніки, оскільки вони працюють із технічно складними товарами: смартфонами, ноутбуками, комп'ютерними комплектуючими, побутовою електронікою, аксесуарами та іншими пристроями [1-3].

Особливістю інтернет-магазину електроніки є те, що клієнт часто потребує не лише можливості оформити замовлення, а й отримати додаткову консультацію. Покупець може уточнювати технічні характеристики товару, сумісність пристроїв, умови гарантії, способи доставки, правила повернення, наявність товару або особливості оплати. Тому якісна клієнтська підтримка є важливою частиною роботи такого підприємства [1-5].

Підтримка клієнтів інтернет-магазину електроніки охоплює процеси приймання звернень, їх класифікації, опрацювання оператором, надання відповіді клієнту, контролю статусу та збереження історії взаємодії. Якщо ці процеси виконуються вручну або через різні неузгоджені канали комунікації, виникають проблеми з контролем запитів, швидкістю відповіді та якістю обслуговування [4; 5].

У межах даної кваліфікаційної роботи предметна галузь розглядається з позиції розробки web-застосунку ElectroCare, який призначений для автоматизації підтримки клієнтів інтернет-магазину електроніки. Такий застосунок має забезпечити централізовану роботу зі зверненнями, розмежування ролей

користувачів, контроль статусів, збереження повідомлень, використання бази знань і формування звітної інформації.

1.2 Актуальність створення системи підтримки клієнтів

Актуальність розробки автоматизованої системи підтримки клієнтів зумовлена зростанням кількості онлайн-замовлень і збільшенням навантаження на служби підтримки інтернет-магазинів. Чим більший обсяг продажів, тим більше клієнтів звертаються із питаннями щодо товарів, доставки, гарантії, повернення або стану замовлення [1-5].

У разі ручної обробки звернень оператори змушені працювати з різними джерелами інформації: електронною поштою, телефонними зверненнями, месенджерами, соціальними мережами або формами на сайті. Це ускладнює пошук попередньої історії спілкування, призводить до дублювання відповідей і збільшує ризик втрати важливого повідомлення [4; 5].

Для клієнта така ситуація проявляється у вигляді довгого очікування відповіді, необхідності повторно пояснювати проблему, відсутності чіткого розуміння статусу звернення та зниження довіри до магазину. Для підприємства це означає підвищене навантаження на персонал, складність контролю роботи операторів і відсутність повноцінної аналітики щодо типових проблем клієнтів [4; 5].

Автоматизована система підтримки дозволяє об’єднати основні процеси в одному середовищі. Клієнт створює звернення через web-інтерфейс, система зберігає його в базі даних, визначає категорію та статус, а оператор отримує можливість опрацювати запит у єдиному робочому просторі. Такий підхід підвищує керованість процесу підтримки та робить комунікацію з клієнтом більш прозорою [4; 5].

1.3 Цільова аудиторія та основні користувачі системи

Цільовою аудиторією web-застосунку ElectroCare є інтернет-магазини електроніки, які потребують інструменту для централізованої обробки клієнтських звернень. Система може бути корисною для малого та середнього бізнесу, де немає потреби у складних комерційних helpdesk-платформах, але є необхідність у базовому контролі звернень, статусів, повідомлень і довідкових матеріалів [4; 5].

У системі передбачено три основні ролі користувачів: клієнт, оператор підтримки та адміністратор.

Клієнт — це користувач інтернет-магазину, який створює звернення щодо товару, замовлення, доставки, гарантії, повернення або іншої проблеми. Для нього важливо швидко подати запит, бачити його статус, отримати відповідь і мати доступ до довідкової інформації.

Оператор підтримки — це працівник, який опрацьовує звернення клієнтів. Він переглядає список запитів, фільтрує їх за статусом або категорією, відповідає клієнтам, змінює статуси звернень і використовує шаблони відповідей для типових ситуацій.

Адміністратор — це користувач із розширеними правами. Він може виконувати частину дій оператора, а також керувати користувачами, ролями, категоріями звернень, шаблонами відповідей, довідковими матеріалами та звітністю. Наявність адміністратора дозволяє підтримувати структуру системи в актуальному стані та контролювати якість роботи служби підтримки.

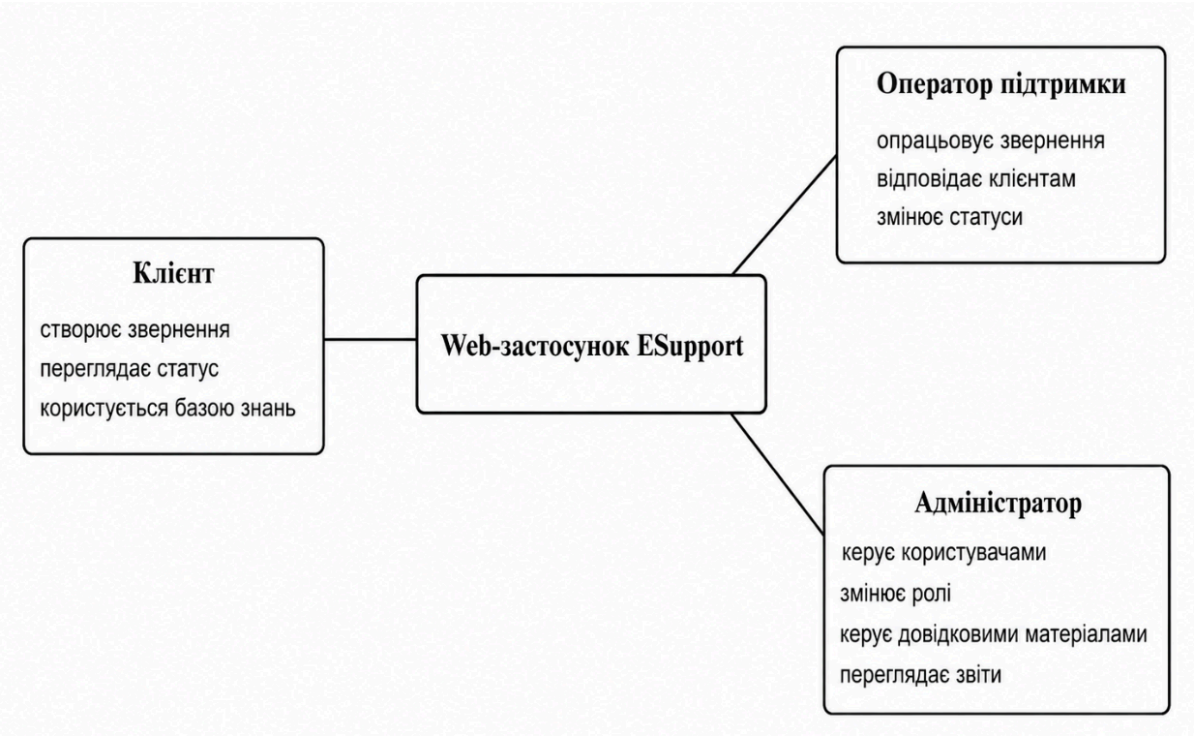


Рис. 1.1 Основні ролі користувачів web-застосунку ElectroCare

На діаграмі варіантів використання має бути показано основні ролі користувачів системи: клієнт, оператор підтримки та адміністратор. Клієнт повинен мати можливість керувати власними зверненнями, переглядати статус і довідкову інформацію. Оператор повинен опрацьовувати звернення, відповідати клієнтам і змінювати статуси. Адміністратор повинен мати розширені права, які включають функції оператора та додаткове керування користувачами, категоріями, шаблонами, довідковими матеріалами і звітами.

1.4 Основні функції та бізнес-логіка системи

Основна бізнес-логіка системи ElectroCare пов’язана з повним циклом роботи клієнтського звернення. Цей цикл починається зі створення звернення клієнтом і завершується його вирішенням або закриттям оператором чи адміністратором [4; 5].

Клієнт створює звернення через web-форму, у якій зазначає тему, категорію та опис проблеми. Після цього система присвоює зверненню початковий статус,

зберігає його в базі даних і робить доступним для перегляду оператором підтримки. Оператор відкриває звернення, аналізує його зміст, за потреби використовує шаблон відповіді, надсилає повідомлення клієнту та змінює статус звернення [4; 5].

У системі доцільно передбачити такі статуси звернення: створено, в роботі, очікує уточнення, вирішено, закрито. Такий підхід дозволяє клієнту бачити поточний стан свого запиту, а оператору та адміністратору — контролювати навантаження і своєчасність опрацювання звернень [4; 5].

До основних функцій web-застосунку ElectroCare належать:

– реєстрація та авторизація користувачів;
– розмежування доступу за ролями;
– створення та редагування звернення клієнтом;
– перегляд статусу звернення;
– збереження історії повідомлень;
– категоризація звернень;
– фільтрація звернень за статусом, категорією, датою або клієнтом;
– надання відповіді оператором;
– використання шаблонів відповідей;
– керування довідковою інформацією;
– формування звітів щодо кількості звернень, статусів і навантаження операторів.

Окремим елементом бізнес-логіки є база знань. Вона дозволяє розміщувати відповіді на типові питання клієнтів: умови доставки, правила повернення, гарантійне обслуговування, способи оплати, перевірка товару під час отримання. Це зменшує кількість повторюваних звернень і допомагає клієнтам самостійно знаходити потрібну інформацію [4; 5].

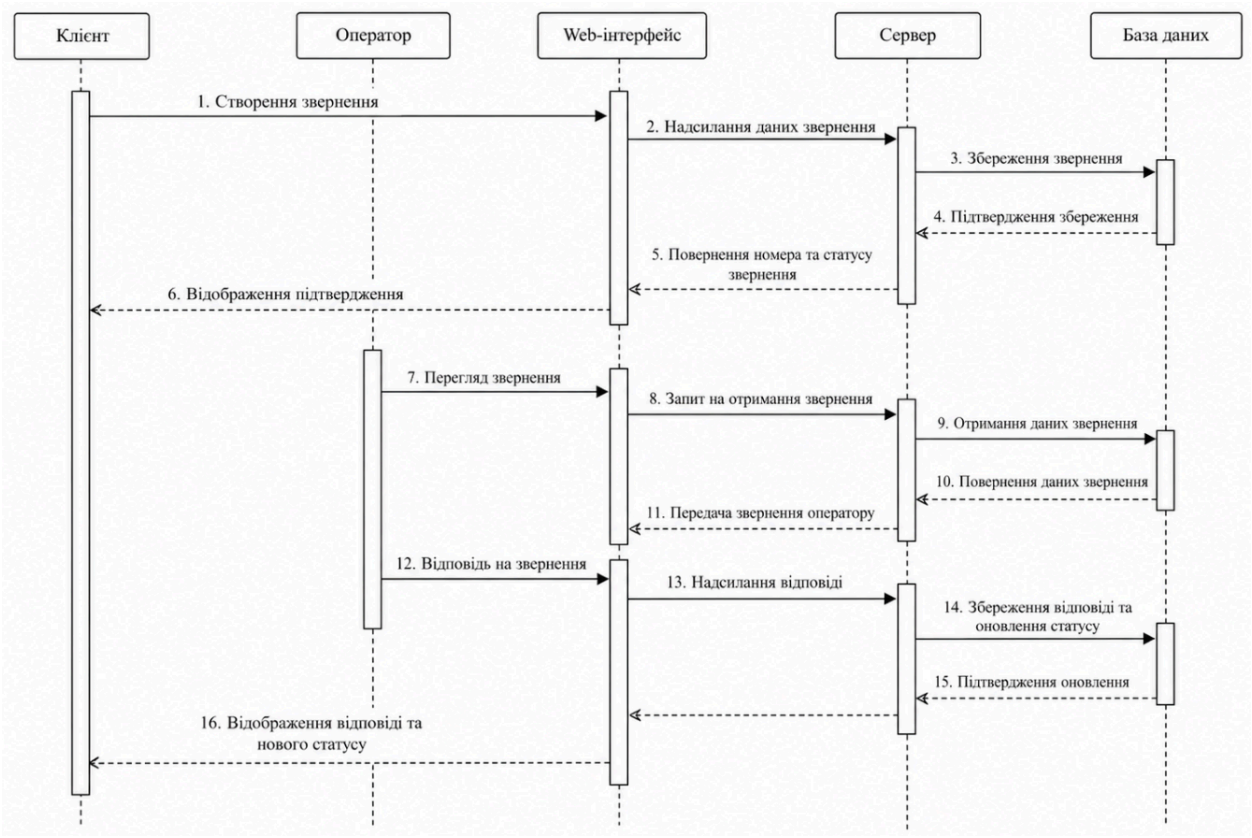


Рис. 1.2 Діаграма послідовності процесу створення та опрацювання звернення клієнта

На діаграмі послідовності необхідно показати синхронну взаємодію між клієнтом, web-інтерфейсом, серверною частиною, базою даних та оператором підтримки. Клієнт створює звернення, система зберігає його, повертає підтвердження, оператор відкриває звернення, надсилає відповідь і оновлює статус. Специфікація виконання повинна тривати до моменту отримання відповіді, оскільки користувач очікує результат у межах одного сценарію взаємодії.

1.5 Способи вирішення задачі та роль інфокомунікаційних засобів

Задача організації підтримки клієнтів може вирішуватися різними способами. Найпростішим варіантом є використання електронної пошти або месенджерів. Такий підхід не потребує складного впровадження, але погано

підходить для системної роботи з великою кількістю звернень, оскільки не забезпечує повноцінного контролю статусів, категоризації та аналітики [4; 5].

Іншим варіантом є використання CRM-систем або helpdesk-платформ. Вони мають широкий набір можливостей для обробки звернень, але часто є комерційними, складними в налаштуванні та надлишковими для невеликого або середнього інтернет-магазину електроніки. Крім того, такі системи не завжди легко адаптувати під конкретну логіку підприємства [4; 5].

Розробка власного web-застосунку дозволяє створити систему, яка відповідає конкретним потребам предметної галузі. У випадку ElectroCare система орієнтована саме на підтримку клієнтів інтернет-магазину електроніки, тому її функціональність може бути сфокусована на найбільш важливих процесах: зверненнях, категоріях, статусах, повідомленнях, шаблонах, довідковій інформації та звітах.

Інфокомунікаційні засоби відіграють ключову роль у вирішенні цієї задачі. Web-інтерфейс забезпечує доступ користувачів до системи через браузер, REST API організовує обмін даними між клієнтською та серверною частинами, база даних забезпечує збереження інформації, а серверна логіка відповідає за обробку запитів і контроль правил роботи системи [19].

Завдяки такому підходу клієнти, оператори та адміністратори можуть працювати в єдиному інформаційному середовищі. Це зменшує кількість ручних операцій, підвищує швидкість обробки звернень і дозволяє краще контролювати якість клієнтського сервісу [4; 5].

1.6 Основні вимоги до програмного продукту

Web-застосунок ElectroCare повинен відповідати функціональним і нефункціональним вимогам, які визначають як можливості системи, так і якість її роботи [4; 5].

До функціональних вимог належать:

– система повинна підтримувати реєстрацію та авторизацію користувачів;

- у системі мають бути передбачені три ролі: клієнт, оператор підтримки, адміністратор;
- клієнт повинен мати можливість створити звернення за 1–2 хвилини;
- звернення повинні розподілятися щонайменше за 6 категоріями: товар, замовлення, доставка, оплата, гарантія, повернення;
- кожне звернення повинно мати статус: створено, в роботі, очікує уточнення, вирішено або закрито;
- система повинна зберігати історію повідомлень у межах кожного звернення;
- адміністратор повинен мати можливість керувати користувачами, категоріями, шаблонами, довідковими матеріалами та звітами.

До нефункціональних вимог належать:

- час завантаження основних сторінок системи — не більше 3 секунд;
- час відповіді сервера на стандартний запит — до 1 секунди;
- система повинна підтримувати одночасну роботу не менше 50 користувачів;
- система повинна забезпечувати збереження не менше 10 000 звернень без втрати стабільності;
- інтерфейс повинен бути адаптований для екранів від 1366×768 px і вище;
- основні дії користувача повинні виконуватися не більше ніж за 2–3 кліки;
- доступ до функцій системи повинен обмежуватися відповідно до ролі користувача;

Зазначені вимоги дозволяють сформувати основу для подальшого проєктування системи, визначення її модулів, структури бази даних і сценаріїв тестування [4; 5].

1.7 Аналіз проблем і недоліків існуючих підходів

Існуючі підходи до організації підтримки клієнтів мають низку недоліків. Якщо інтернет-магазин використовує лише пошту, телефон або месенджери, виникає проблема розрізненості інформації. Звернення не зберігаються в єдиній

системі, історія комунікації може втрачатися, а оператори не завжди мають повний контекст звернення [4; 5].

Комерційні helpdesk-системи вирішують багато цих проблем, але мають інші обмеження.

Основними проблемами, які потребують вирішення, є:

– відсутність єдиного центру обробки звернень;
– складність контролю статусів клієнтських запитів;
– втрата історії комунікації при використанні різних каналів зв’язку;
– збільшення часу відповіді через ручне опрацювання;
– дублювання типових звернень і відповідей;
– відсутність зручної бази знань;
– складність аналізу роботи операторів;

З огляду на ці проблеми, розробка web-застосунку ElectroCare є доцільною. Система дозволяє об’єднати основні процеси підтримки клієнтів в одному середовищі, забезпечити централізоване збереження звернень, контроль статусів, розмежування ролей і використання довідкової інформації [4; 5].

2 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПІДТРИМКИ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ЕЛЕКТРОНІКИ

Для організації підтримки клієнтів інтернет-магазину електроніки можуть використовуватися різні програмні засоби: CRM-системи, helpdesk-платформи, сервіс-деск рішення, чат-системи, бази знань, системи керування зверненнями та спеціалізовані web-застосунки. Основне призначення таких засобів полягає у централізованому прийманні звернень клієнтів, їх категоризації, розподілі між операторами, контролі статусів, збереженні історії комунікації та формуванні звітної інформації [4; 5].

Для інтернет-магазину електроніки така функціональність є особливо важливою, оскільки клієнтські звернення можуть бути пов’язані не лише з оформленням замовлення, а й із технічними характеристиками товарів, сумісністю пристроїв, гарантією, поверненням, доставкою, оплатою та післяпродажним обслуговуванням. Через це система підтримки повинна забезпечувати не просто обмін повідомленнями, а повний цикл роботи зі зверненням [1-5].

Найпростішими засобами комунікації з клієнтами є електронна пошта, телефон, месенджери та соціальні мережі. Проте ці інструменти не забезпечують повноцінного контролю над зверненнями. У таких умовах оператору складно швидко визначити статус запиту, знайти попередню історію спілкування, передати звернення іншому працівнику або сформувати звіт щодо якості роботи служби підтримки. Тому для системної роботи доцільно використовувати спеціалізоване програмне забезпечення [4; 5].

У межах роботи було розглянуто декілька програмних засобів, які можуть застосовуватися для автоматиз1визначити його місце серед існуючих рішень [6-9].

2.1 Zendesk

Zendesk є однією з найбільш відомих платформ для організації клієнтської підтримки. Основою її роботи є система тікетів, коли кожне звернення клієнта перетворюється на окремий запит. Такий запит можна призначити оператору, змінювати його статус, додавати внутрішні коментарі, переглядати історію спілкування та контролювати процес опрацювання [6].

Для інтернет-магазину електроніки Zendesk може використовуватися для обробки звернень щодо вибору товарів, доставки, гарантії, повернення, технічних консультацій та стану замовлення. Платформа підтримує різні канали комунікації, зокрема електронну пошту, чат, web-форми та соціальні мережі. Завдяки цьому всі звернення клієнтів можуть потрапляти в єдину систему [6].

Важливою перевагою Zendesk є наявність бази знань, автоматизації робочих процесів, звітності та аналітики. Це дозволяє зменшити навантаження на операторів і швидше відповідати на типові питання. Наприклад, система може використовувати готові шаблони відповідей або спрямовувати клієнта до відповідного довідкового матеріалу [6].

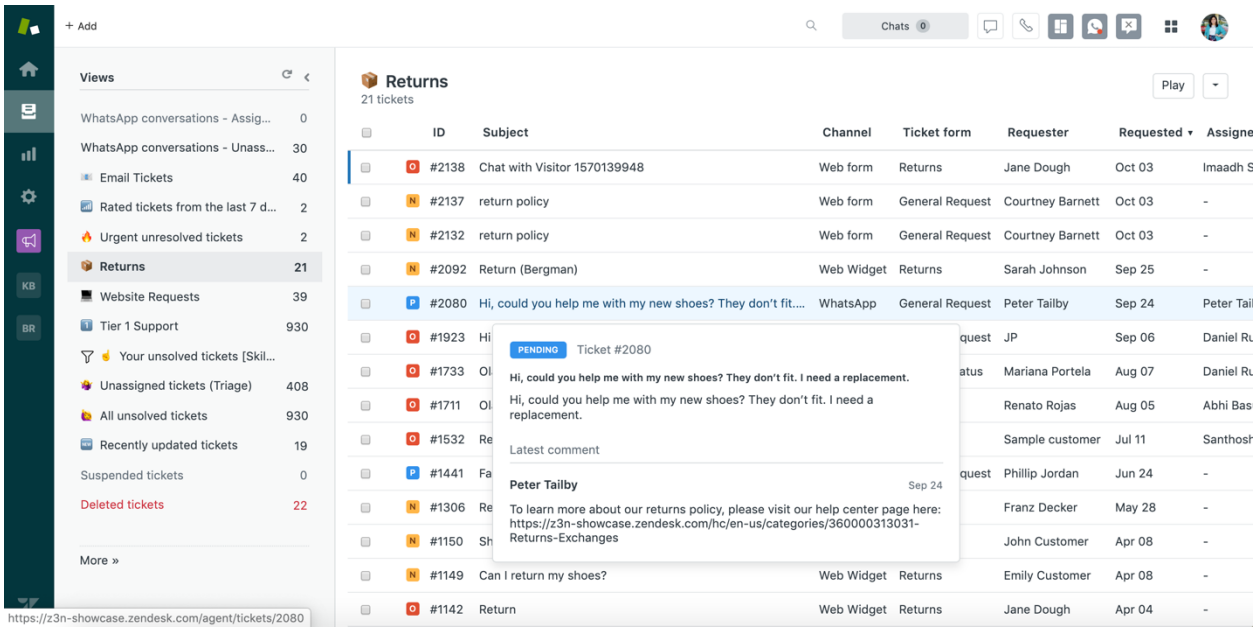


Рис. 2.1 Приклад інтерфейсу платформи Zendesk

До переваг Zendesk можна віднести:

– централізовану систему тікетів;

– підтримку багатьох каналів комунікації;

– можливість автоматизації типових дій;

– наявність бази знань;

– аналітику та звітність;

– гнучке налаштування ролей і прав доступу [6].

Недоліком Zendesk є досить висока складність впровадження та вартість використання. Для невеликого інтернет-магазину електроніки функціональність платформи може бути надлишковою, а налаштування бізнес-правил, інтеграцій і ролей може потребувати додаткових технічних ресурсів [6].

2.2 Freshdesk

Freshdesk є helpdesk-платформою, призначеною для обробки звернень клієнтів, керування тікетами, налаштування SLA, створення бази знань і формування звітів. Система дозволяє приймати звернення з різних каналів і перетворювати їх на тікети, які можуть оброблятися операторами підтримки [7].

Для інтернет-магазину електроніки Freshdesk може бути корисним завдяки можливості групувати звернення за категоріями, пріоритетами та статусами. Наприклад, звернення щодо гарантії може мати вищий пріоритет, ніж загальне питання про характеристики товару. Такий підхід дозволяє ефективніше розподіляти навантаження між операторами [7].

Freshdesk також підтримує клієнтський портал і базу знань. Це дозволяє клієнтам самостійно знаходити відповіді на поширені питання щодо доставки, оплати, повернення або гарантійного обслуговування. Для служби підтримки це зменшує кількість однотипних звернень і прискорює обробку складніших запитів [7].

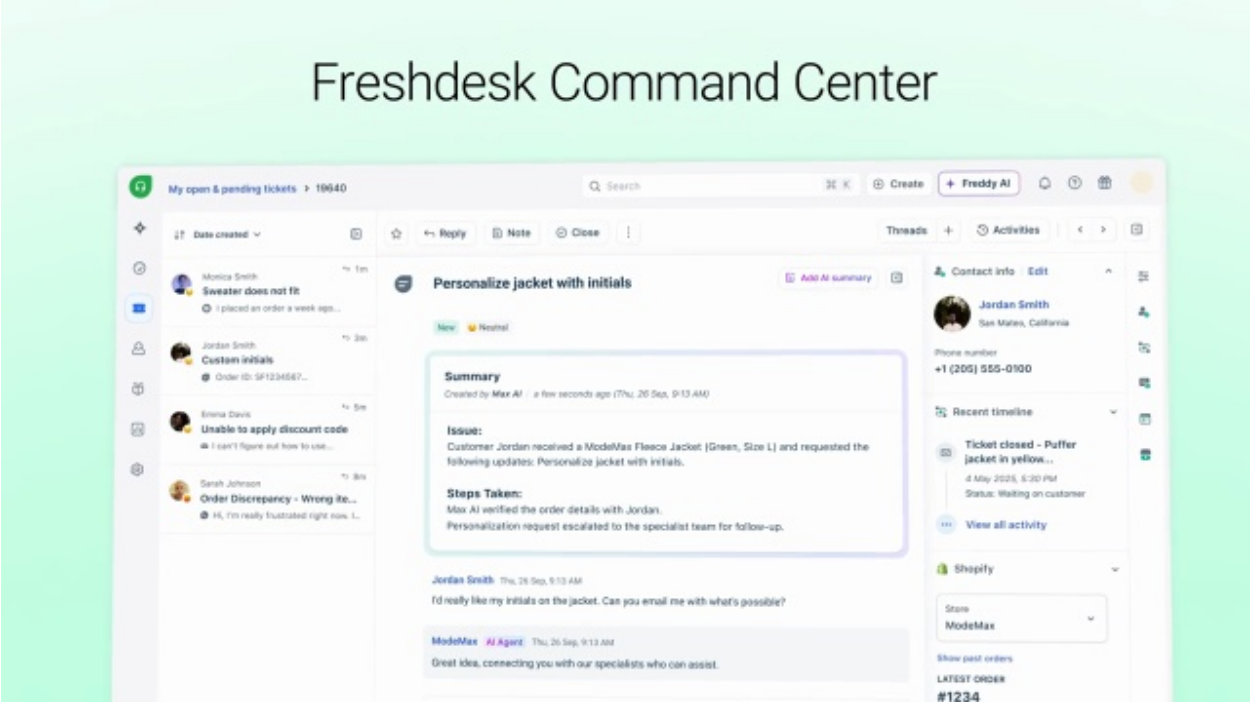


Рис. 2.2 Приклад інтерфейсу платформи Freshdesk

Переваги Freshdesk:

– зручна система тікетів;

– підтримка статусів і пріоритетів;

– можливість налаштування SLA;

– база знань і клієнтський портал;

– автоматизація робочих процесів;

– наявність аналітики та звітності [7].

До недоліків Freshdesk можна віднести те, що частина важливих функцій доступна лише в платних тарифах. Також система є універсальною і не орієнтована виключно на специфіку інтернет-магазинів електроніки, тому для її повної адаптації можуть знадобитися додаткові налаштування [7].

2.3 Intercom

Intercom є платформою для комунікації з клієнтами, яка поєднує чат, helpdesk-функції, базу знань, автоматизацію та AI-інструменти. Основний акцент у

цій системі зроблено на швидку взаємодію з користувачем безпосередньо на сайті або в web-застосунку [8].

Для інтернет-магазину електроніки Intercom може бути корисним у випадках, коли клієнту потрібно швидко отримати відповідь під час перегляду товару або оформлення замовлення. Наприклад, користувач може уточнити сумісність аксесуара з певною моделлю смартфона, запитати про наявність товару або умови доставки [8].

Intercom також дозволяє автоматизувати частину відповідей і використовувати базу знань. Це зручно для типових питань, які часто повторюються. Крім того, система підтримує інструменти для персоналізованої комунікації з користувачем і може бути інтегрована з іншими сервісами [8].

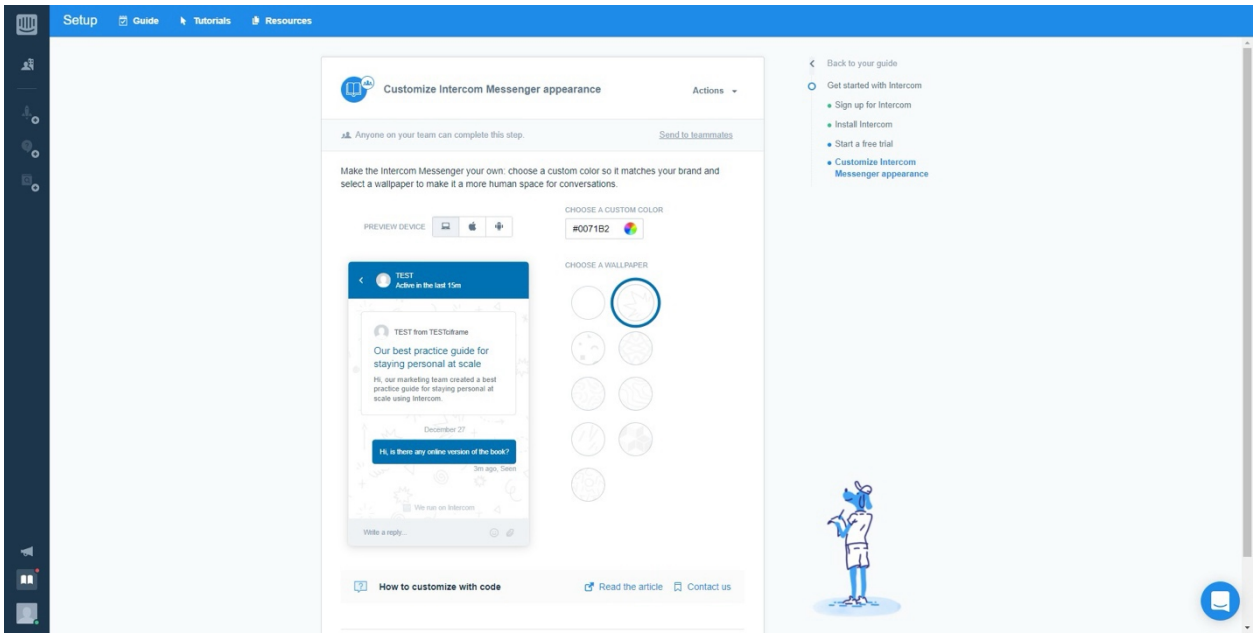


Рис. 2.3 Приклад інтерфейсу платформи Intercom

Переваги Intercom:

– зручна чат-комунікація;

– швидка взаємодія з клієнтом;

– автоматизація типових відповідей;

– підтримка бази знань;

– AI-інструменти для допомоги оператору;

– можливість інтеграції з web-сайтом [8].

Недоліком Intercom є те, що система більше орієнтована на онлайн-комунікацію та SaaS-сервіси. Для звичайного інтернет-магазину електроніки частина функцій може бути надлишковою. Також повноцінне використання платформи потребує налаштування сценаріїв автоматизації, бази знань та інтеграцій [8].

2.4 Help Scout

Help Scout — це платформа для організації підтримки клієнтів, яка робить акцент на простоті використання та роботі зі спільною поштовою скринькою. Основна ідея системи полягає в тому, що команда підтримки може спільно працювати з повідомленнями клієнтів, залишати внутрішні нотатки, призначати відповідальних і контролювати відповіді [9].

Для інтернет-магазину електроніки Help Scout може бути корисним у випадках, коли основним каналом підтримки є електронна пошта. Наприклад, клієнт надсилає запит щодо повернення товару, гарантії або уточнення деталей замовлення, а оператори можуть спільно переглядати це звернення, додавати коментарі та формувати відповідь [9].

Перевагою Help Scout є простий інтерфейс і менша складність впровадження порівняно з великими helpdesk-платформами. Такий засіб може бути зручним для малого або середнього бізнесу, де немає потреби у складній автоматизації, але важливо мати впорядковану роботу зі зверненнями [9].

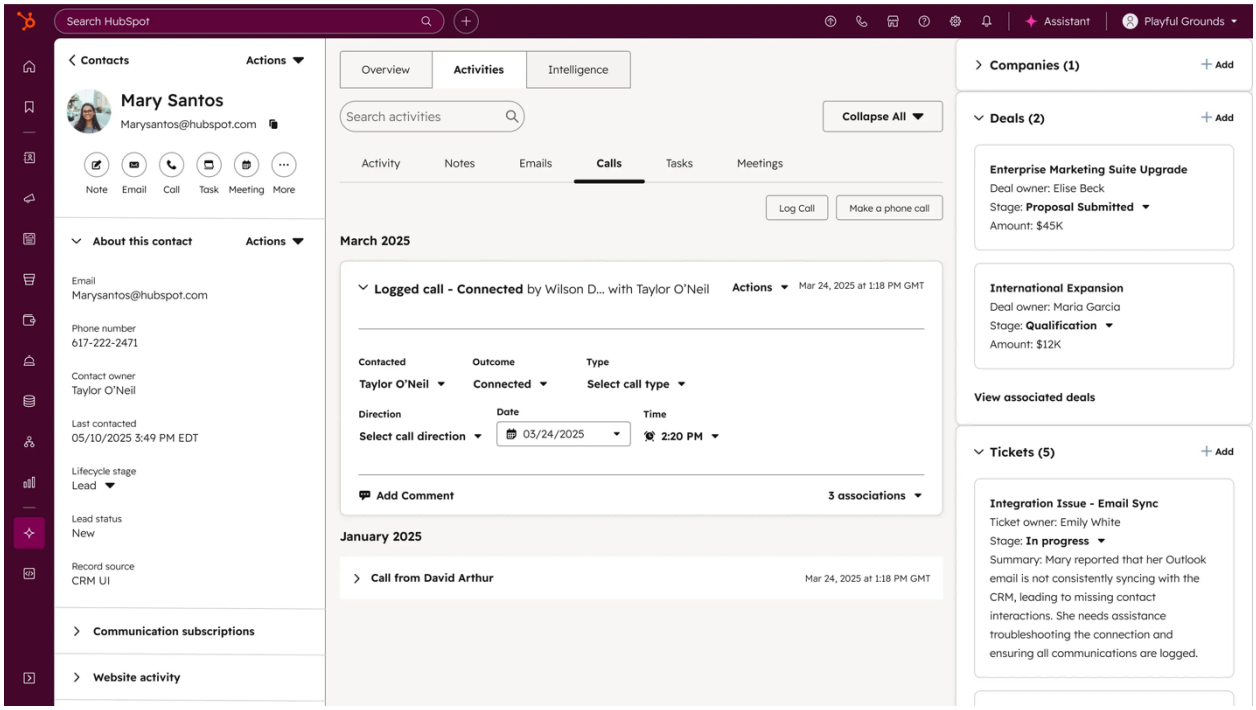


Рис. 2.4 Приклад інтерфейсу платформи Help Scout

Переваги Help Scout:

– простота використання;

– спільна робота команди підтримки;

– внутрішні нотатки для операторів;

– зручна робота з email-зверненнями;

– менша складність впровадження;

– наявність базових функцій для підтримки клієнтів [9].

Недоліками Help Scout є обмежені можливості складної автоматизації, аналітики та інтеграції з бізнес-процесами інтернет-магазину. Для великих магазинів із високим навантаженням функціональності системи може бути недостатньо [9].

2.5 Порівняльний аналіз програмних засобів

Розглянуті програмні засоби мають спільну мету — підвищення ефективності клієнтської підтримки. Проте кожна система має власний акцент. Zendesk і Freshdesk є потужними helpdesk-платформами з широкою

функціональністю. Intercom більше орієнтований на чат-комунікацію та автоматизацію. Help Scout підходить для простішої командної роботи зі зверненнями, особливо через електронну пошту [6-9].

У межах кваліфікаційної роботи розроблюється web-застосунок ElectroCare, який орієнтований саме на базові потреби інтернет-магазину електроніки. Його головна перевага полягає в тому, що система не містить надлишкового функціоналу, а зосереджена на найбільш необхідних процесах: створенні звернень, категоризації, контролі статусів, історії повідомлень, шаблонах відповідей, базі знань і звітності.

Зведені результати порівняння програмних засобів наведено в таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз програмних засобів для автоматизації підтримки клієнтів

	Zendesk	Freshdesk	Intercom	Help Scout	ElectroCare
Система тікетів	+	+	+	+	+
Категоризація звернень	+	+	+	частково	+
Контроль статусів	+	+	+	+	+
База знань	+	+	+	+	+
Чат з клієнтом	+	+	+	частково	частково
Аналітика та звіти	+	+	+	частково	+
Автоматизація процесів	+	+	+	частково	+
Інтеграція з CRM	+	+	+	частково	-
Smart Triage score для звернення	-	-	-	-	+

Продовження таблиці 2.1

Порівняльний аналіз програмних засобів для автоматизації підтримки клієнтів

	Zendesk	Freshdesk	Intercom	Help Scout	ElectroCare
Автоматичне призначення оператора на звернення	-	-	-	-	+
SLA Radar для швидкого виявлення прострочених звернень	частково	частково	частково	-	+
Простота впровадження	середня	середня	середня	висока	висока
Орієнтація на малий бізнес	частково	+	частково	+	+
Орієнтація на інтернет-магазин електроніки	частково	частково	частково	частково	+

За результатами аналізу можна зробити висновок, що існуючі програмні засоби мають широкий набір можливостей для організації клієнтської підтримки, однак не завжди повністю відповідають потребам невеликого або середнього інтернет-магазину електроніки. Більшість таких систем є комерційними, потребують налаштування та можуть містити функціональність, яка не використовується в межах конкретного бізнес-процесу [6-10].

3 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ПІДТРИМКИ КЛІЄНТІВ ІНТЕРНЕТ-МАГАЗИНУ ЕЛЕКТРОНІКИ

3.1 Логічна архітектура

Для реалізації автоматизованої системи підтримки клієнтів інтернет-магазину електроніки доцільно використовувати клієнт-серверну архітектуру. Такий підхід дозволяє розділити логіку роботи системи на окремі частини: клієнтський інтерфейс, серверну логіку та базу даних. Клієнтська частина відповідає за взаємодію користувача з системою, серверна частина — за обробку звернень, авторизацію, маршрутизацію запитів і роботу з даними, а база даних — за збереження інформації про користувачів, звернення, повідомлення, категорії, статуси та шаблони відповідей [11-20].

У межах розробки web-застосунку для підтримки клієнтів інтернет-магазину електроніки передбачається використання таких засобів реалізації:

– мова програмування Java;
– фреймворк Spring Boot;
– REST API для взаємодії клієнтської та серверної частин;
– система керування базами даних PostgreSQL;
– бібліотека React для створення клієнтського інтерфейсу;
– мова TypeScript для підвищення надійності frontend-частини [11-20].

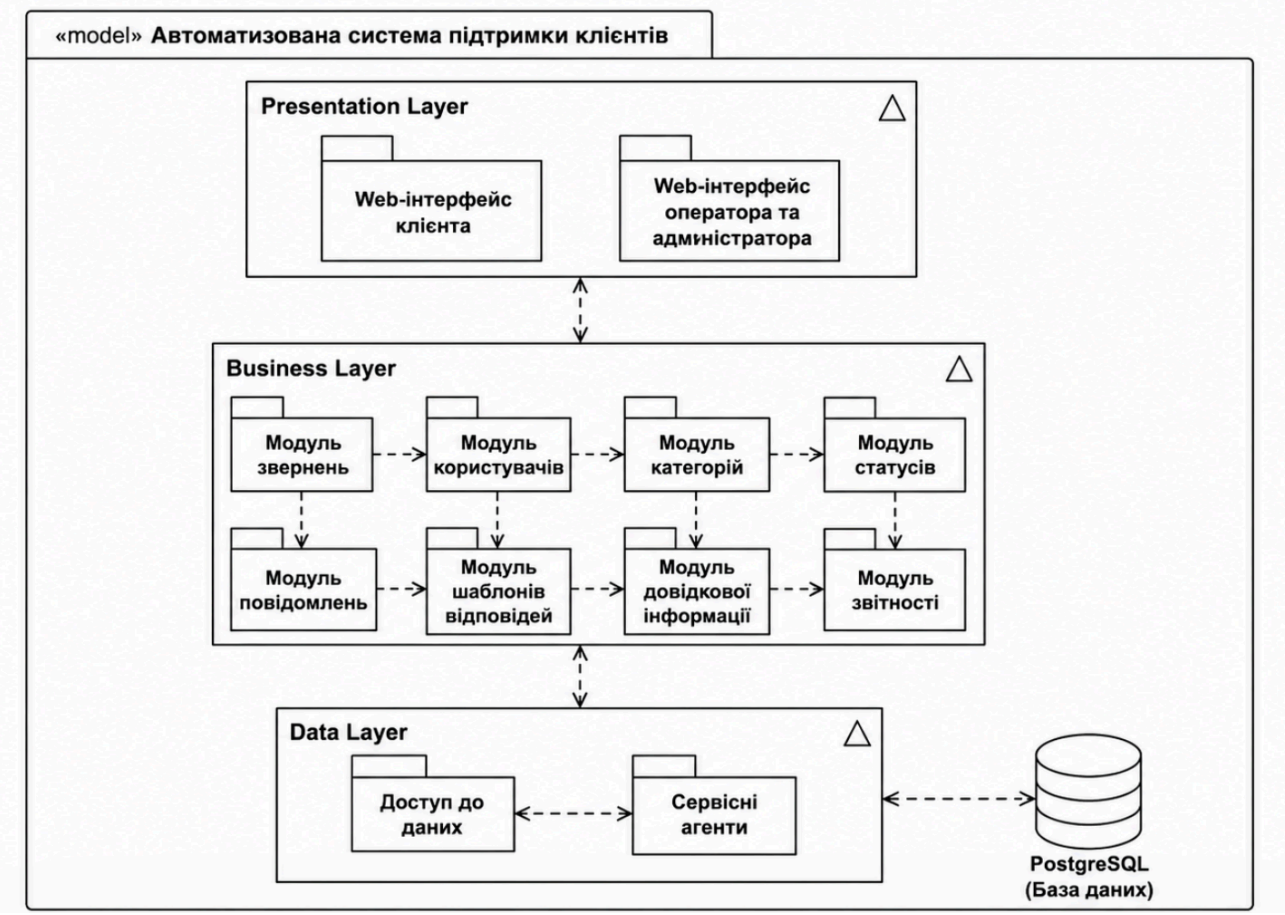


Рис. 3.1 Логічна архітектура автоматизованої системи підтримки клієнтів інтернет-магазину електроніки

3.2 Мова програмування Java

Серверну частину системи доцільно реалізувати мовою програмування Java. Java є однією з найбільш поширених мов для створення серверних, корпоративних та web-орієнтованих систем. Вона підтримує об’єктно-орієнтований підхід, має розвинену екосистему бібліотек і фреймворків, а також використовується для розробки застосунків, які потребують стабільної роботи, масштабованості та безпечної обробки даних. У документації Oracle Java SE зазначається, що Java використовується для розробки та розгортання застосунків на серверах і настільних системах, а також орієнтована на продуктивність, безпеку та переносимість програмного забезпечення. [11].

Для даного проєкту Java є доцільним вибором, оскільки система підтримки клієнтів повинна виконувати обробку різних типів запитів, працювати з користувачами, статусами, повідомленнями, категоріями звернень і шаблонами відповідей. Крім того, серверна частина має забезпечувати стабільну взаємодію з базою даних та клієнтським інтерфейсом [11].

Для спрощення створення серверної частини використовується Spring Boot. Spring Boot дозволяє швидше розгортати web-застосунок, налаштовувати маршрути, контролери, сервіси, роботу з базою даних і зовнішніми запитами. У документації Spring Boot представлено як середовище, яке містить готові підходи для створення Spring-застосунків і спрощує початкове налаштування проєкту [12; 13; 18].

3.3 Spring Boot та Spring Framework

У системі підтримки клієнтів Spring Boot може використовуватися для реалізації таких функціональних можливостей: [12; 13; 18].

– реєстрація та авторизація користувачів;
– створення звернень клієнтами;
– перегляд списку звернень оператором;
– зміна статусів звернень;
– збереження повідомлень у межах діалогу;
– робота з категоріями звернень;
– керування шаблонами відповідей;
– формування звітних даних для адміністратора.

3.4 REST API та формат обміну даними JSON

Для взаємодії frontend- та backend-частин доцільно використовувати REST API. Такий підхід дозволяє клієнтській частині надсилати HTTP-запити до серверної частини та отримувати структуровані відповіді. Наприклад, клієнт

створює звернення через web-форму, після чого frontend надсилає запит на сервер, а сервер зберігає інформацію в базі даних і повертає номер звернення та його поточний статус. Spring має окремі матеріали щодо створення REST-сервісів, які описують підхід до побудови HTTP-орієнтованої взаємодії між частинами застосунку. [19].

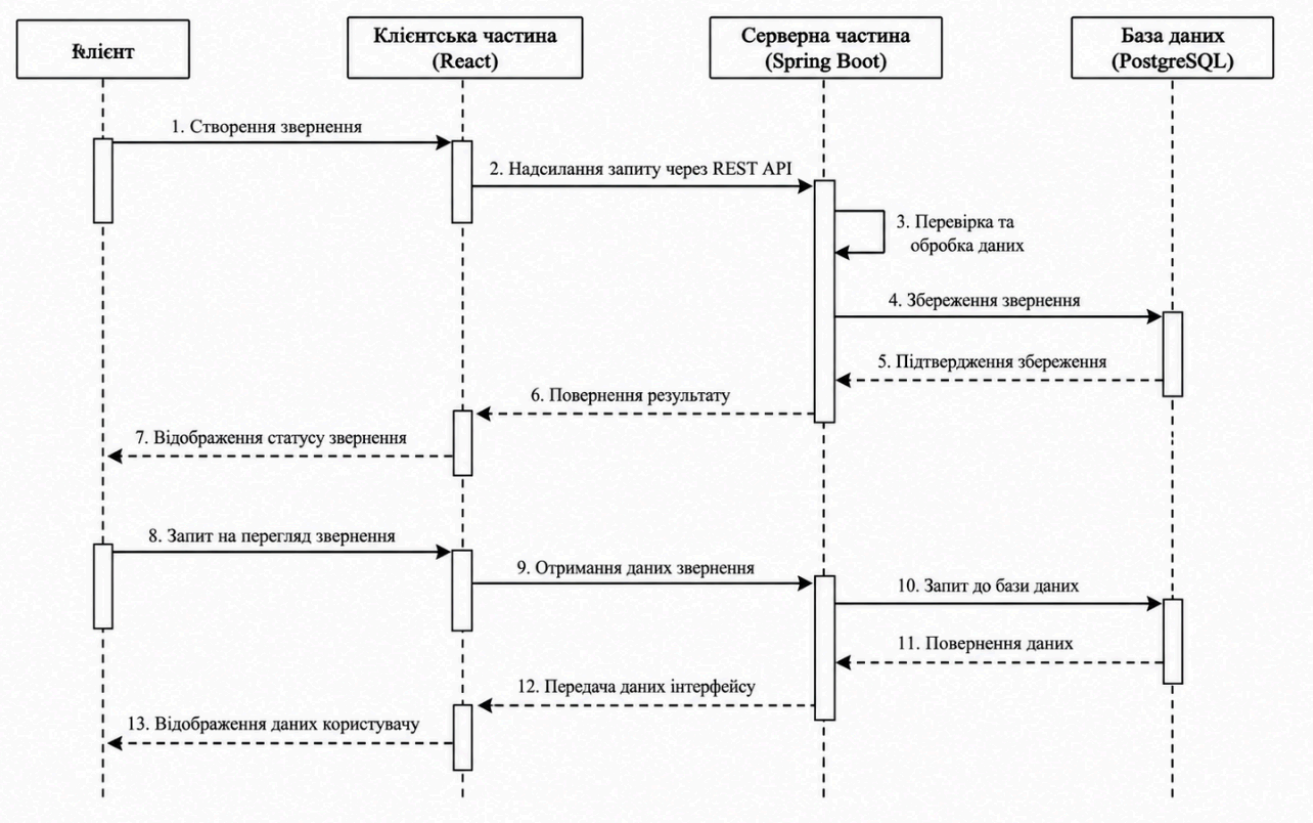


Рис. 3.2 Діаграма взаємодії клієнтської частини, серверної частини та бази даних

3.5 PostgreSQL

Для збереження даних у системі використовується PostgreSQL. Це об’єктно-реляційна система керування базами даних з відкритим кодом, яка підтримує SQL і може використовуватися для збереження складних структурованих даних. На офіційному сайті PostgreSQL зазначено, що ця система має репутацію надійної, функціональної та продуктивної бази даних. [14; 15].

У межах розроблюваної системи PostgreSQL може зберігати такі основні сутності:

– користувачі системи;

– ролі користувачів;

– звернення клієнтів;

– категорії звернень;

– повідомлення в межах звернення;

– статуси опрацювання;

– шаблони відповідей;

– довідкові матеріали;

– історія змін і взаємодій.

Використання PostgreSQL є доцільним, оскільки інформація в системі має чітку структуру і пов’язана між собою. Наприклад, одне звернення належить конкретному клієнту, має певну категорію, статус, список повідомлень і відповідального оператора. Реляційна модель бази даних дозволяє зручно описати такі зв’язки та забезпечити цілісність даних [14; 15].

3.6 React та TypeScript

Для створення клієнтського інтерфейсу доцільно використати React. React — це бібліотека для побудови інтерфейсів користувача на основі компонентного підходу. В офіційній документації зазначено, що React-застосунки складаються з ізольованих частин інтерфейсу, які називаються компонентами. [16].

У системі підтримки клієнтів React може застосовуватися для створення таких елементів інтерфейсу:

– форма створення звернення;

– сторінка перегляду статусу звернення;

– список звернень для оператора;

– картка окремого звернення;

– інтерфейс повідомлень між клієнтом і оператором;

– панель адміністратора;

– сторінка керування категоріями та шаблонами;

– довідковий розділ.

Перевагою React є можливість розділення інтерфейсу на окремі компоненти. Наприклад, компонент форми звернення, компонент списку звернень, компонент повідомлення або компонент статусу можуть розроблятися та повторно використовуватися незалежно один від одного. Це спрощує підтримку коду і робить frontend-частину більш структурованою [16].

Для підвищення надійності клієнтської частини може використовуватися TypeScript. TypeScript розширює JavaScript шляхом додавання типів, що дозволяє виявляти частину помилок ще на етапі розробки, а не під час виконання програми. В офіційній документації TypeScript зазначено, що мова додає типізацію до JavaScript і покращує процес розробки завдяки ранньому виявленню помилок та підтримці редакторів коду. [17].

У даній системі TypeScript є корисним для опису структури даних, які надходять від серверної частини. Наприклад, можна чітко визначити типи для користувача, звернення, повідомлення, категорії або статусу. Це зменшує ризик помилок під час роботи з API та полегшує подальше розширення системи [17].

Загальна логіка роботи застосунку може бути представлена таким чином: клієнт або оператор взаємодіє з web-інтерфейсом, frontend-частина надсилає запит до серверної частини через REST API, backend обробляє запит, виконує необхідну бізнес-логіку та звертається до PostgreSQL для збереження або отримання даних. Після цього сервер повертає відповідь клієнтській частині, а React відображає оновлену інформацію користувачу [16; 19].

У результаті вибір Java, Spring Boot, PostgreSQL, React і TypeScript є обґрунтованим для реалізації автоматизованої системи підтримки клієнтів інтернет-магазину електроніки. Такий набір технологій дозволяє створити структурований web-застосунок із розділенням відповідальності між клієнтською частиною, серверною логікою та базою даних. Це забезпечує зручність розробки, можливість подальшого масштабування та адаптацію системи під потреби конкретного інтернет-магазину [11-20].

4 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ ELECTROCARE

Проєктування web-застосунку ElectroCare виконувалося з урахуванням того, що система повинна не просто зберігати звернення клієнтів, а забезпечувати повний цикл їх опрацювання: від моменту створення запиту клієнтом до його аналізу, призначення оператору, контролю SLA, ведення переписки, фіксації внутрішніх дій та завершення обробки. Тобто система розглядається не як проста форма зворотного зв’язку, а як невелика helpdesk-платформа, адаптована під задачі інтернет-магазину електроніки [4; 5].

Під час проєктування основна увага була приділена таким аспектам:

– розмежуванню прав доступу між клієнтом, оператором і адміністратором;

– побудові зрозумілої логічної структури системи;

– моделюванню основних процесів роботи зі зверненнями;

– визначенню доменних сутностей системи;

– проєктуванню бази даних для збереження користувачів, звернень, повідомлень, журналу дій і бази знань;

– опису компонентів клієнтської та серверної частин;

– підготовці системи до подальшого тестування.

Окремо в системі передбачено модуль Smart Triage Radar, який виконує первинну інтелектуальну маршрутизацію звернення без використання зовнішнього AI-сервісу. Умовно кажучи, система не просто приймає звернення, а одразу намагається “зрозуміти”, наскільки воно термінове, який пріоритет має отримати, який SLA-дедлайн потрібно встановити, яку дію варто порекомендувати оператору та яку статтю бази знань можна запропонувати для вирішення проблеми.

4.1 Логічна структура системи

Web-застосунок ElectroCare побудований за клієнт-серверною архітектурою. Такий підхід є доцільним для системи підтримки клієнтів, оскільки дозволяє відокремити користувацький інтерфейс від серверної логіки та бази даних. На практиці це означає, що клієнт, оператор і адміністратор взаємодіють із системою через web-інтерфейс, а всі основні операції з авторизацією, зверненнями, повідомленнями, SLA, triage-аналізом і журналом дій виконуються на серверній частині [19].

Логічну структуру системи можна поділити на декілька основних рівнів. Першим є рівень клієнтського інтерфейсу. Він реалізований на React із використанням Vite, React Router та Axios. Саме цей рівень відповідає за відображення сторінок системи: “Огляд”, “Мої задачі”, “Деталі звернення”, “База знань”, “Профіль” та “Користувачі”. Клієнтська частина також враховує роль користувача. Наприклад, звичайний клієнт не бачить сторінку керування користувачами, а адміністратор має доступ до ширшого набору функцій [16; 20].

Другим є рівень API та маршрутизації запитів. Взаємодія між frontend і backend здійснюється через REST API. Клієнтська частина надсилає HTTP-запити до серверної частини, отримує відповіді у форматі JSON і відображає ці дані в інтерфейсі. Наприклад, при відкритті сторінки “Мої задачі” frontend звертається до відповідного API, отримує список звернень, які належать клієнту або призначені оператору, і відображає їх користувачу [19].

Третім є рівень бізнес-логіки. Це центральна частина системи, де реалізовано основні правила роботи. Саме тут обробляються звернення, призначаються оператори, змінюються статуси, додаються повідомлення, визначається SLA-дедлайн, формується triage-score і записуються події до журналу дій. Цей рівень фактично відповідає за те, щоб система працювала не просто як набір сторінок, а як цілісний програмний продукт.

Четвертим є рівень доступу до даних. Він реалізований за допомогою Spring Data JPA. Завдяки цьому серверна частина може працювати з таблицями бази даних

через об’єктну модель. Це спрощує роботу з користувачами, зверненнями, повідомленнями, статтями бази знань і журналом дій [12; 18].

П’ятим є рівень збереження даних. Для цього використовується PostgreSQL. У базі даних зберігаються користувачі, звернення, повідомлення, журнал дій за зверненнями та статті бази знань. Для керування змінами структури бази даних використовується Liquibase, зокрема міграція 002-enhance-support-platform.yaml, яка розширює систему новими полями профілю, SLA, Smart Triage та журналом дій [14; 15].

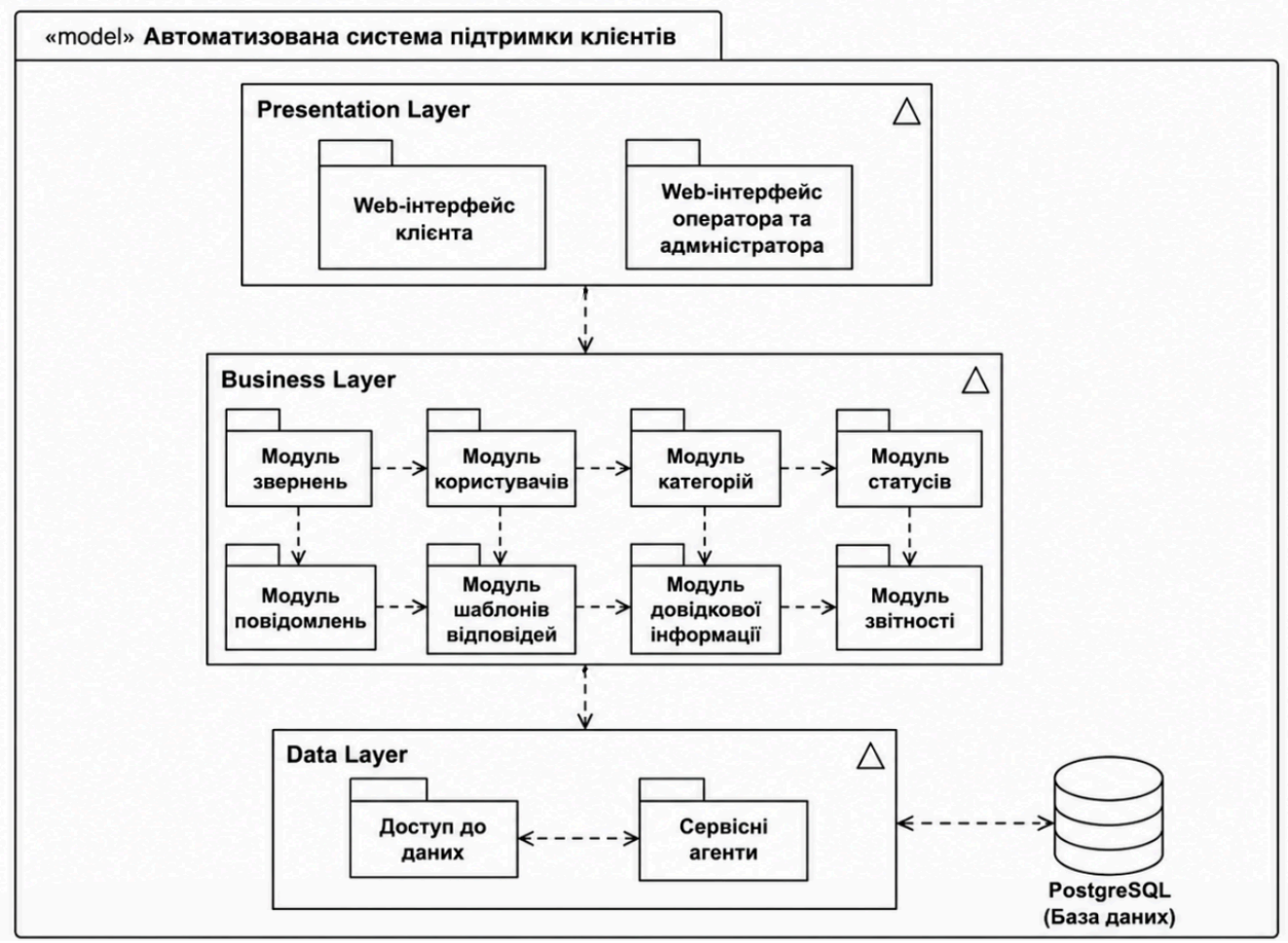


Рис. 4.1 Логічна структура web-застосунку ElectroCare

Після моделювання логічної структури можна зробити висновок, що система має чіткий поділ на клієнтський, серверний, бізнесовий та інформаційний рівні. Такий поділ дозволяє простіше підтримувати застосунок, додавати нові функції та

тестувати окремі модулі. Наприклад, модуль Smart Triage або SLA можна розширити без повної зміни клієнтської частини чи структури всього проєкту [19].

4.2 Діаграма варіантів використання web-застосунку ElectroCare

Діаграма варіантів використання дозволяє показати, які користувачі взаємодіють із системою та які основні функції їм доступні. У web-застосунку ElectroCare передбачено три основні ролі: CUSTOMER**,** OPERATOR та ADMIN [4; 5].

Роль CUSTOMER відповідає клієнту інтернет-магазину. Клієнт може створювати звернення, переглядати власні звернення, читати відповіді оператора, додавати повідомлення, відстежувати статус звернення, переглядати базу знань і редагувати власний профіль. У контексті цієї системи клієнт не просто відправляє повідомлення, а бачить повний цикл обробки свого запиту.

Роль OPERATOR відповідає працівнику служби підтримки. Оператор має більше можливостей, ніж клієнт, оскільки він не тільки переглядає звернення, а й безпосередньо їх опрацьовує. Оператор може переглядати загальний реєстр звернень, відкривати сторінку “Мої задачі”, брати неназначені звернення в роботу, переглядати Smart Triage Radar, додавати відповіді клієнту, створювати внутрішні нотатки, змінювати статуси, аналізувати SLA і переглядати журнал дій.

Роль ADMIN має найбільші права в системі. Адміністратор повинен мати можливість виконувати частину дій оператора, а також додатково керувати користувачами. До його функцій належать перегляд списку користувачів, зміна ролі, активація або деактивація облікового запису, редагування відділу та посади користувача. Також адміністратор може контролювати загальну роботу системи, звернення, навантаження операторів і якість обробки клієнтських запитів.

Тут важливо правильно показати, що адміністратор не є окремим “слабким” користувачем із двома функціями. Навпаки, він має найширший набір прав: частина його можливостей дублює функції оператора, а частина є суто

адміністративною. Саме тому на діаграмі варіантів використання адміністратор повинен бути пов’язаний із більшою кількістю прецедентів, ніж інші ролі.

Рис. 4.2 Діаграма варіантів використання web-застосунку ElectroCare

За результатами моделювання варіантів використання визначено три групи функцій: клієнтські, операторські та адміністративні. Клієнт працює переважно зі своїми зверненнями, оператор виконує їх обробку, а адміністратор має розширені права та може керувати як користувачами, так і частиною службових процесів системи.

4.3 Діаграма доменної моделі web-застосунку ElectroCare

Доменна модель відображає основні сутності предметної галузі та зв’язки між ними. Для web-застосунку ElectroCare головними сутностями є користувачі, звернення, повідомлення, журнал дій та статті бази знань [4; 5].

Сутність User описує користувача системи. У системі користувач може мати одну з трьох ролей: ADMIN, OPERATOR або CUSTOMER. Крім базових даних, таких як email, пароль і повне ім’я, профіль користувача містить телефон, посаду, відділ, колір профілю та ознаку активності. Це дозволяє зробити систему більш наближеною до реального середовища служби підтримки, де оператори можуть належати до різних відділів, наприклад гарантійного, доставки, продажів або оплати.

Сутність SupportTicket є центральною в системі. Вона описує звернення клієнта. Звернення містить тему, опис, номер замовлення, категорію, статус, пріоритет, джерело, призначеного оператора, SLA-дедлайн, резюме вирішення, оцінку задоволеності клієнта, triage-score, рекомендовану дію та рекомендовану статтю бази знань. Тобто звернення містить не лише текст клієнтського запиту, а й службову інформацію, необхідну для якісної обробки.

Сутність TicketMessage використовується для збереження повідомлень у межах звернення. Повідомлення може бути написане клієнтом, оператором або адміністратором. Також передбачено поле internal_note, яке дозволяє відрізняти звичайні повідомлення від внутрішніх нотаток. Це важливо, тому що оператор може залишати службові коментарі, які не повинні бути видимі клієнту.

Сутність TicketActivityLog відповідає за журнал дій. Вона фіксує, хто і яку дію виконав зі зверненням. Наприклад, оператор узяв звернення в роботу, змінив статус, додав внутрішню нотатку або відповів клієнту. Це робить процес обробки звернень прозорим і дозволяє в майбутньому аналізувати роботу операторів.

Сутність KnowledgeArticle описує статтю бази знань. Вона містить назву, категорію, короткий опис, повний текст, ознаку активності та дати створення й оновлення. Статті бази знань використовуються як для самостійного ознайомлення клієнтів, так і для рекомендацій Smart Triage Radar.

Між сутностями існують такі основні зв’язки:

– один користувач може створити багато звернень;

– один оператор може бути призначений на багато звернень;

– одне звернення містить багато повідомлень;

– один користувач може написати багато повідомлень;

– одне звернення має багато записів журналу дій;

– один користувач може виконати багато дій, які фіксуються в журналі;

– стаття бази знань може бути рекомендована для звернення через поле recommended_article_id.

Окремо варто зазначити, що статус звернення може бути реалізований як окремий довідник або як перерахований тип. Якщо статус моделюється як сутність TicketStatus, то між TicketStatus і SupportTicket повинна бути асоціація “один до багатьох”: один статус може відповідати багатьом зверненням, а кожне звернення має один поточний статус.

Побудована доменна модель, яка зображена на рисунку 4.3, показує, що основою системи є звернення клієнта, навколо якого групуються повідомлення, журнал дій, SLA-дані, triage-аналіз і рекомендовані матеріали бази знань.

Така структура дозволяє зберігати не тільки сам запит, а й повний контекст його опрацювання [4; 5].

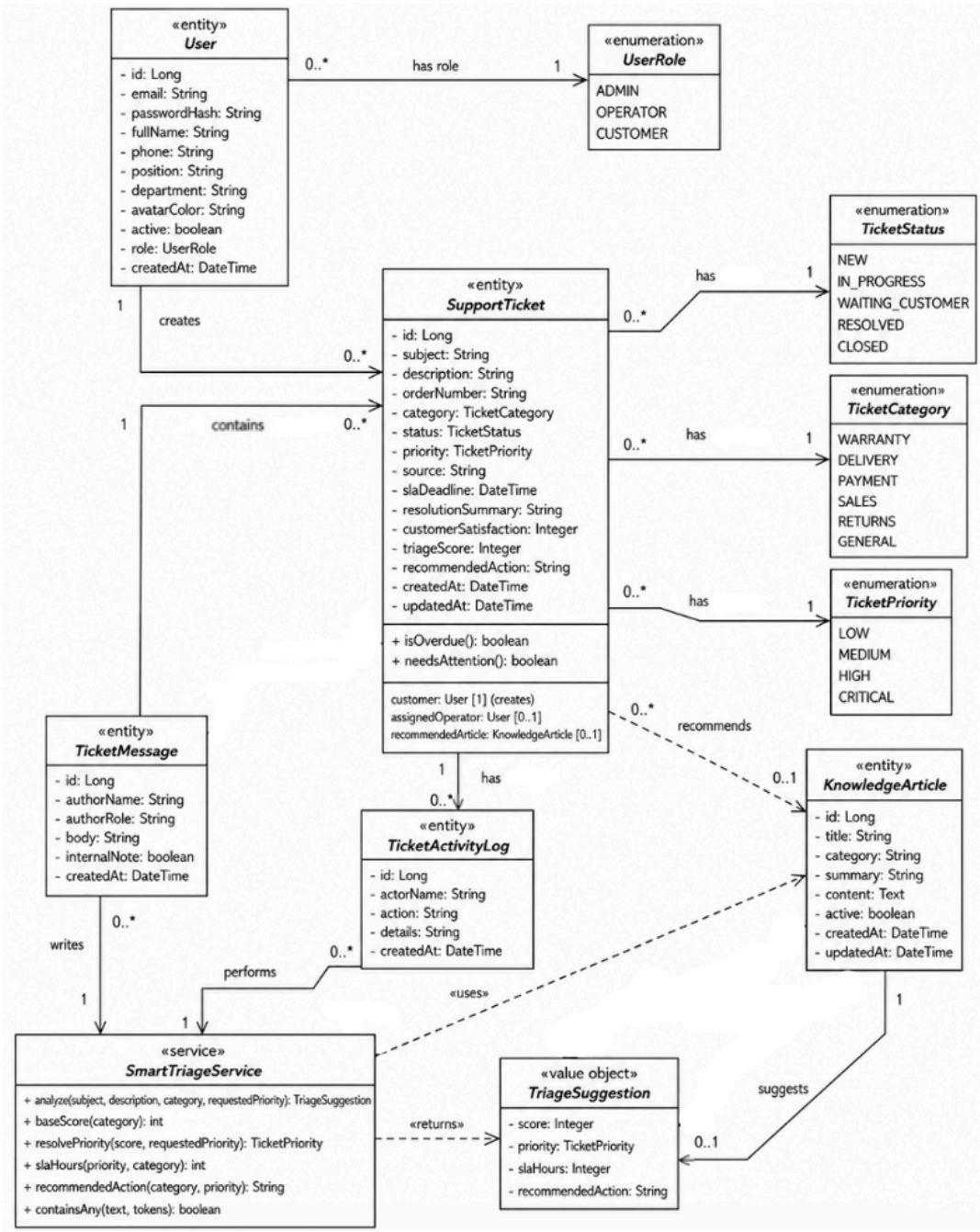


Рис. 4.3 Діаграма доменної моделі web-застосунку ElectroCare

4.4 Діаграми діяльності

Діаграми діяльності використовуються для моделювання поведінки системи в межах конкретних бізнес-процесів. У роботі доцільно змоделювати не всі можливі сценарії, а саме ті, які найбільш повно показують логіку ElectroCare: створення звернення клієнтом, обробку звернення оператором, контроль SLA та керування користувачами адміністратором.

4.4.1 Діаграма діяльності процесу створення звернення клієнтом

Процес створення звернення починається з авторизації клієнта в системі. Після входу клієнт переходить до створення нового звернення, заповнює тему, опис проблеми, категорію та за потреби номер замовлення. Система перевіряє коректність введених даних. Якщо обов'язкові поля не заповнені, користувачу відображається повідомлення про помилку.

Якщо дані введено коректно, система створює нове звернення. На цьому етапі спрацьовує модуль Smart Triage Radar. Він аналізує категорію, опис і ключові слова звернення, після чого визначає triage-score, пріоритет, SLA-дедлайн, рекомендовану дію та відповідну статтю бази знань. Наприклад, якщо клієнт пише про несправність товару або гарантійний випадок, система може встановити вищий пріоритет і рекомендувати оператору перевірити гарантійні умови.

Після цього звернення зберігається в базі даних, а клієнт отримує підтвердження про створення звернення. З цього моменту звернення стає доступним у списку "Мої задачі" або в загальному реєстрі звернень для оператора.

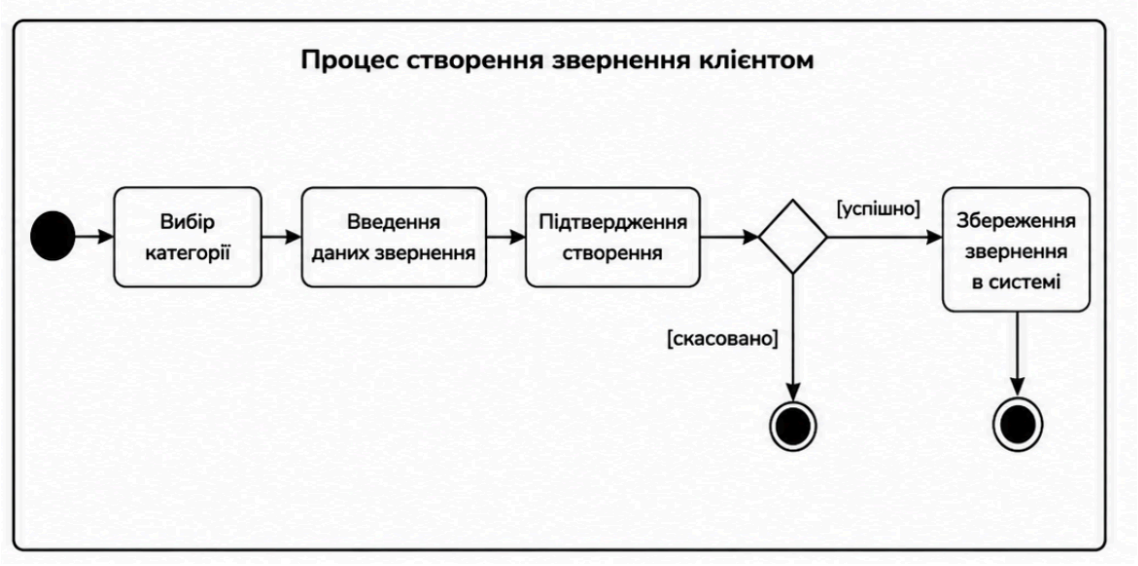


Рис. 4.4 Діаграма діяльності процесу створення звернення клієнтом

Змодельований процес показує, що створення звернення в ElectroCare не обмежується простим збереженням тексту. Система одразу виконує первинний

аналіз звернення, визначає його важливість і готує додаткову службову інформацію для оператора.

4.4.2 Діаграма діяльності процесу обробки звернення оператором

Процес обробки звернення оператором починається з відкриття сторінки “Мої задачі” або загального реєстру звернень. Якщо звернення ще не призначене оператору, працівник підтримки може взяти його в роботу. Після цього система фіксує призначення звернення та записує відповідну дію до журналу TicketActivityLog.

Оператор відкриває детальну сторінку звернення, де бачить тему, опис, категорію, статус, пріоритет, SLA-дедлайн, призначеного оператора, історію повідомлень, внутрішні нотатки, журнал дій і блок Smart Triage Radar. Саме цей блок допомагає оператору швидше зрозуміти, що робити далі. Наприклад, система може рекомендувати перевірити номер замовлення, запропонувати статтю бази знань або звернути увагу на наближення SLA-дедлайну.

Після аналізу звернення оператор може виконати одну з дій: надіслати відповідь клієнту, додати внутрішню нотатку, змінити статус, оновити пріоритет або закрити звернення після вирішення проблеми. Усі важливі дії зберігаються в журналі, що дозволяє відстежувати історію роботи над зверненням.

Рис. 4.5 Діаграма діяльності процесу обробки звернення оператором

Діаграма діяльності процесу обробки звернення показує, що оператор працює не “вручну всліпу”, а отримує структуровану інформацію: статус, SLA, triage-score, рекомендації, історію діалогу та журнал дій. Це підвищує якість обробки звернення та зменшує ризик пропустити важливу деталь.

4.4.3 Діаграма діяльності процесу керування користувачами адміністратором

Адміністратор у системі ElectroCare має розширені можливості керування користувачами. Процес починається з відкриття сторінки “Користувачі”, яка доступна тільки користувачу з роллю ADMIN. На цій сторінці адміністратор бачить список користувачів, їх email, ім’я, роль, активність, відділ і посаду.

Адміністратор може змінити роль користувача, наприклад із CUSTOMER на OPERATOR, якщо користувач повинен отримати доступ до обробки звернень. Також він може активувати або деактивувати обліковий запис, оновити відділ чи посаду. Це корисно для організації роботи служби підтримки, де оператори можуть бути розподілені за напрямами: гарантія, доставка, оплата, продажі.

Після зміни даних система зберігає оновлену інформацію в базі даних. У більш розширеній логіці така дія також може фіксуватися в журналі або адміністративному логуванні.

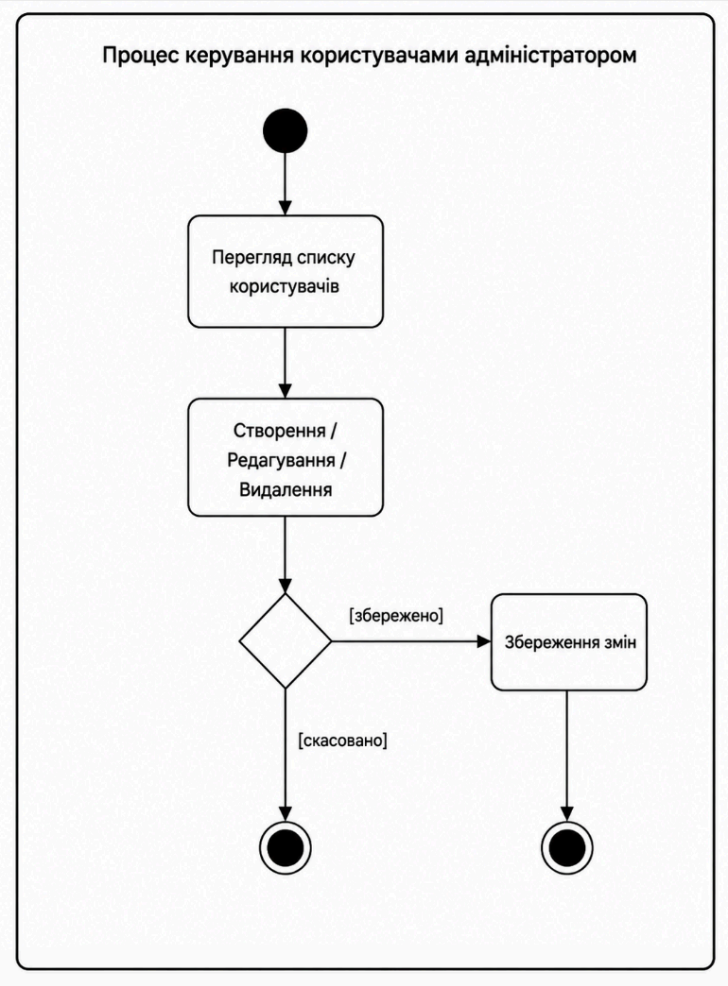


Рис. 4.6 Діаграма діяльності процесу керування користувачами адміністратором

Діаграма діяльності процесу керування користувачами демонструє, що адміністратор забезпечує не тільки контроль системи, а й підтримку її організаційної структури. Він керує ролями, активністю та службовими даними користувачів, що дозволяє адаптувати систему під реальну роботу інтернет-магазину.

4.5 Діаграма класів web-застосунку ElectroCare

Діаграма класів використовується для опису внутрішньої структури програмного забезпечення. У web-застосунку ElectroCare класи можна умовно поділити на декілька груп: сутності, контролери, сервіси та репозиторії [11; 12; 18].

До основних сутностей належать User, SupportTicket, TicketMessage, TicketActivityLog та KnowledgeArticle. Саме ці класи відповідають основним таблицям бази даних і описують інформацію, з якою працює система.

Клас User містить дані про користувача: email, пароль, ім’я, телефон, посаду, відділ, колір профілю, активність і роль. Роль визначає, які сторінки та функції доступні користувачу.

Клас SupportTicket описує звернення клієнта. Він містить інформацію про клієнта, призначеного оператора, тему, опис, номер замовлення, категорію, статус, пріоритет, джерело, SLA-дедлайн, triage-score, рекомендовану дію та рекомендовану статтю бази знань.

Клас TicketMessage відповідає за повідомлення у зверненні. Він зберігає автора, текст повідомлення, роль автора, ознаку внутрішньої нотатки та дату створення.

Клас TicketActivityLog фіксує події, які відбуваються зі зверненням. Наприклад, зміна статусу, призначення оператора або додавання відповіді.

Клас KnowledgeArticle описує статтю бази знань, яка може використовуватися для самостійного ознайомлення клієнтів або як рекомендація для оператора.

Окрему роль у системі відіграють сервісні класи. TicketService відповідає за логіку створення, оновлення та обробки звернень. SmartTriageService аналізує звернення і формує triage-рекомендації. SlaService відповідає за контроль SLA-дедлайнів. UserService реалізує роботу з користувачами, а KnowledgeArticleService — з базою знань. ActivityLogService відповідає за запис дій у журнал.

Контролери забезпечують зв’язок між frontend і backend. Наприклад, TicketController приймає запити щодо звернень, AuthController відповідає за авторизацію, UserAdminController — за адміністративне керування користувачами, а KnowledgeArticleController — за роботу з базою знань.

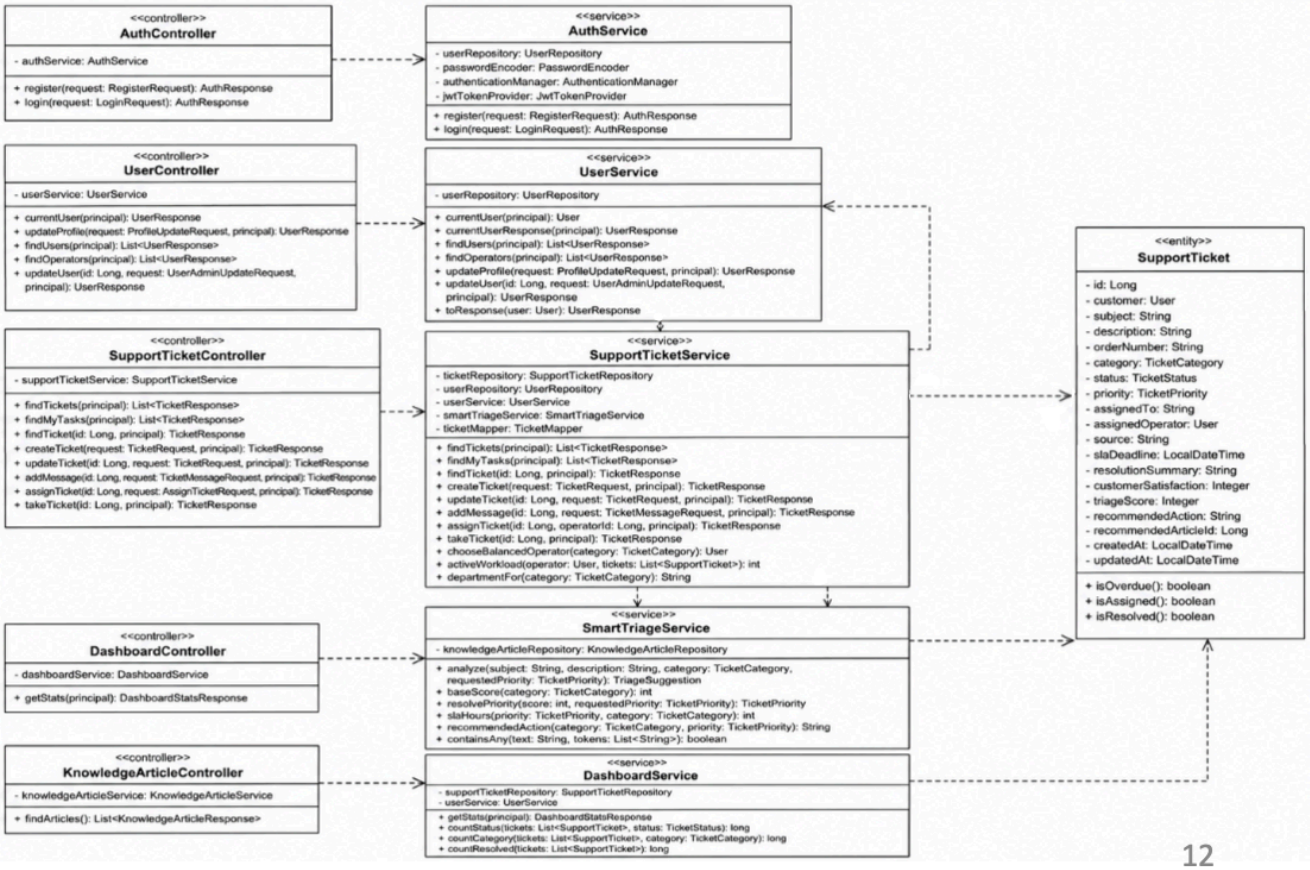


Рис. 4.8 Діаграма класів web-застосунку ElectroCare

Діаграма класів показує, що система має модульну структуру. Сутності відповідають за збереження даних, сервіси — за бізнес-логіку, контролери — за приймання запитів, а репозиторії — за доступ до бази даних. Такий підхід відповідає типовій структурі Spring Boot-застосунку і спрощує подальший розвиток системи [11; 12; 18].

4.6 Діаграма компонентів web-застосунку ElectroCare

Діаграма компонентів дозволяє показати, з яких програмних частин складається система та як вони взаємодіють між собою. Для ElectroCare доцільно виділити frontend-компоненти, backend-модулі та інфраструктурні компоненти [11-20].

До frontend-компонентів належать сторінки:

Інфраструктурний рівень містить PostgreSQL, Liquibase, Docker Compose для запуску бази даних, Spring Security і JWT-механізм доступу. Взаємодія між frontend і backend здійснюється через REST API, а backend працює з базою даних через Spring Data JPA [12-15; 19].

Діаграма компонентів, продемонстрована на рисунку 4.9, показує, що система поділена на окремі функціональні модулі. Це зручно, бо кожен модуль відповідає за конкретну частину логіки. Наприклад, Smart Triage Module можна доопрацьовувати незалежно від модуля профілю користувача, а Knowledge Base Module — незалежно від Dashboard Analytics Module [11-20].

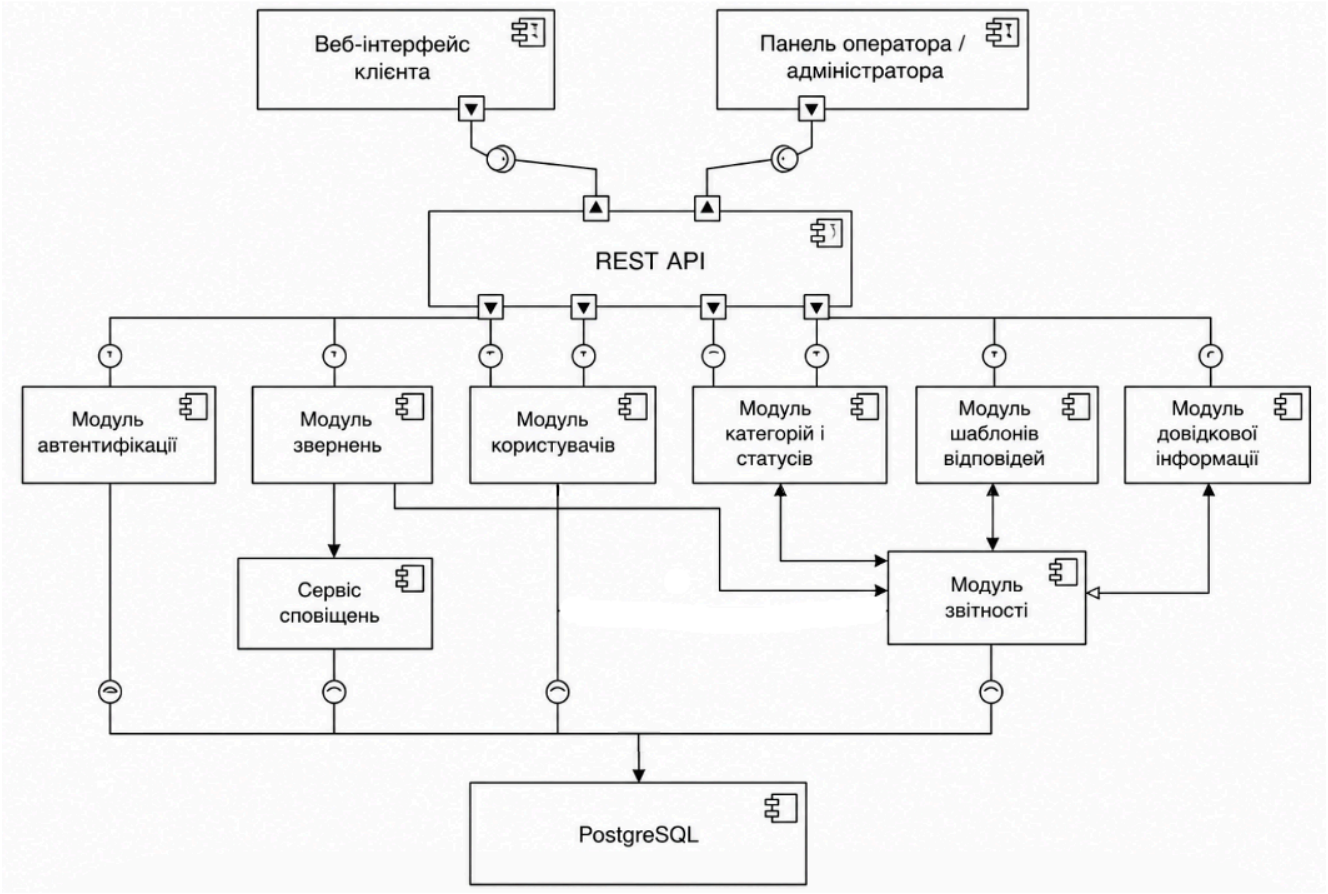


Рис. 4.9 Діаграма компонентів web-застосунку ElectroCare

4.7 Діаграма розгортання web-застосунку ElectroCare

Діаграма розгортання показує фізичне розміщення компонентів системи та способи їх взаємодії. Для web-застосунку ElectroCare можна виділити такі основні вузли: [19].

– клієнтський пристрій із web-браузером;

– frontend-застосунок;

– сервер застосунку Spring Boot;

– сервер бази даних PostgreSQL;

– Docker Compose-середовище для запуску PostgreSQL.

Користувач взаємодіє із системою через браузер. Клієнтська частина надсилає HTTP-запити до backend через REST API. Серверна частина обробляє ці запити, виконує бізнес-логіку та звертається до PostgreSQL для отримання або збереження даних. Для підключення до бази даних використовується JDBC через Spring Data JPA [14; 15; 19].

У середовищі розробки PostgreSQL може запускатися через Docker Compose. Це спрощує підготовку бази даних і дозволяє швидко відтворити середовище на іншому комп’ютері. Структура бази даних керується Liquibase-міграціями, що дозволяє відстежувати зміни схеми БД.

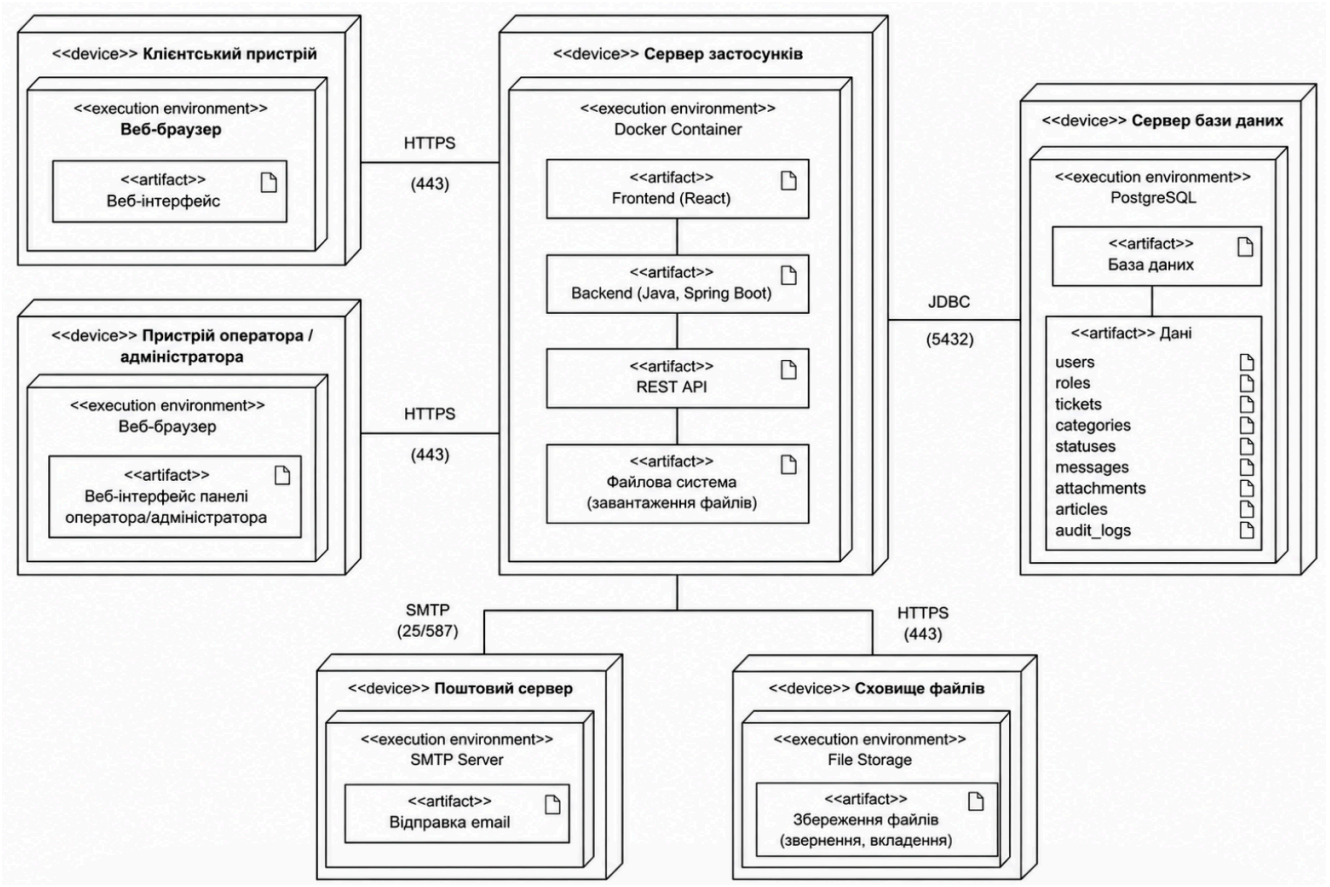


Рис. 4.10 Діаграма розгортання web-застосунку ElectroCare

Діаграма розгортання показує, що система має типову web-структуру: браузер користувача, клієнтська частина, сервер застосунку та база даних. Такий варіант є зрозумілим, практичним і достатнім для бакалаврського проєкту, а також дозволяє в майбутньому перенести систему на повноцінний сервер або хмарну інфраструктуру [19].

4.8 Проєктування бази даних

База даних web-застосунку ElectroCare спроєктована з урахуванням основних процесів роботи системи: збереження користувачів, створення звернень, призначення операторів, ведення переписки, фіксації дій і використання бази знань [14; 15].

Основними таблицями бази даних є:

– USERS;

– SUPPORT_TICKETS;

– TICKET_MESSAGES;

– TICKET_ACTIVITY_LOGS;

– KNOWLEDGE_ARTICLES.

Таблиця USERS зберігає інформацію про користувачів системи. До неї входять поля: id, email, password_hash, full_name, phone, position, department, avatar_color, active, role, created_at. Поле email є унікальним, оскільки один email не може належати кільком користувачам. Поле role визначає роль користувача в системі: ADMIN, OPERATOR або CUSTOMER.

Таблиця SUPPORT_TICKETS зберігає звернення клієнтів. Вона містить посилання на клієнта через customer_id і на призначеного оператора через assigned_operator_id. Крім цього, таблиця містить тему, опис, номер замовлення, категорію, статус, пріоритет, джерело, SLA-дедлайн, резюме вирішення, оцінку задоволеності, triage-score, рекомендовану дію та рекомендовану статтю бази знань.

Таблиця TICKET_MESSAGES містить повідомлення в межах звернення. Кожне повідомлення пов’язане з конкретним зверненням і конкретним автором. Поле internal_note дозволяє визначити, чи є повідомлення внутрішньою нотаткою оператора.

Таблиця TICKET_ACTIVITY_LOGS використовується для збереження журналу дій. Вона дозволяє бачити, які операції виконувалися зі зверненням, хто їх виконував і коли саме. Це важливо для контролю роботи служби підтримки.

Таблиця KNOWLEDGE_ARTICLES містить статті бази знань. Вона зберігає назву, категорію, короткий опис, повний зміст статті, активність та дати створення й оновлення.

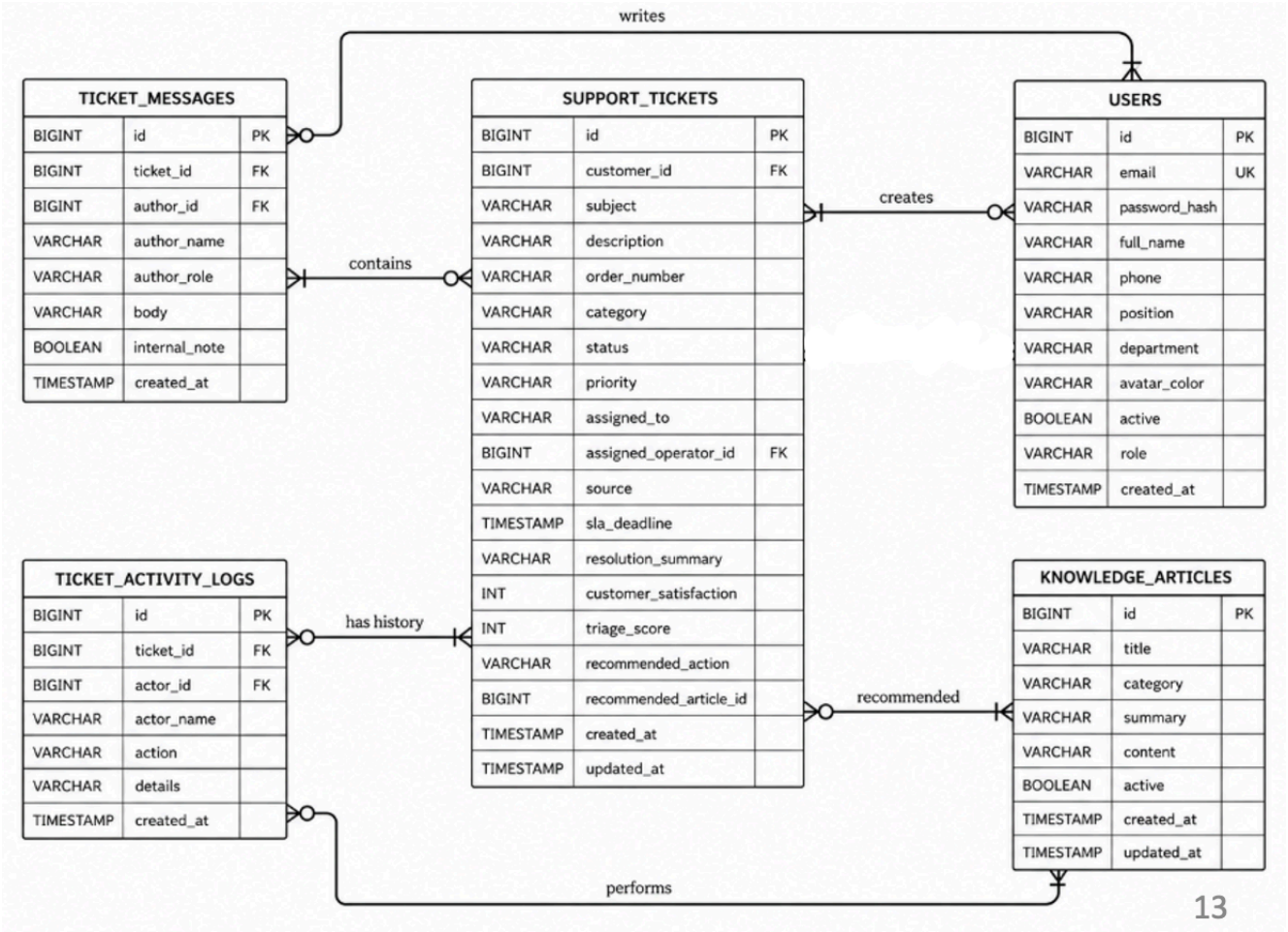


Рис. 4.11 Схема бази даних web-застосунку ElectroCare

Структура бази даних дозволяє забезпечити повний цикл роботи зі зверненням. Кожне звернення пов’язане з клієнтом, може бути призначене оператору, містить повідомлення, має журнал дій і може отримати рекомендовану статтю бази знань. Саме така структура робить систему не просто сховищем звернень, а повноцінним інструментом для організації підтримки клієнтів інтернет-магазину електроніки [14; 15].

5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ ELECTROCARE

Реалізація web-застосунку **ElectroCare** виконувалася з урахуванням задачі автоматизації підтримки клієнтів інтернет-магазину електроніки. На цьому етапі основна увага приділялася не тільки створенню стандартних сторінок для роботи зі зверненнями, а й реалізації допоміжної логіки, яка робить систему більш корисною для оператора підтримки. Тобто система не просто приймає звернення і зберігає його в базі даних, а додатково аналізує його, визначає пріоритет, контролює SLA-дедлайни, веде журнал дій та допомагає розподіляти задачі між операторами [4; 5].

У процесі реалізації було створено клієнт-серверний web-застосунок, у якому серверна частина відповідає за бізнес-логіку, авторизацію, роботу з базою даних, аналіз звернень і контроль прав доступу, а клієнтська частина забезпечує зручний інтерфейс для адміністратора, оператора та клієнта. Такий підхід дозволив розділити відповідальність між частинами системи й зробити проєкт більш зрозумілим для подальшого супроводу [11-20].

5.1 Загальна характеристика реалізованого web-застосунку

Web-застосунок ElectroCare призначений для централізованої підтримки клієнтів інтернет-магазину електроніки. У системі реалізовано три основні ролі користувачів: адміністратор**,** оператор підтримки та клієнт. Кожна роль має власний набір доступних сторінок і функцій, що дозволяє зробити роботу системи більш безпечною та логічно розділеною.

Клієнт у системі може створювати звернення, переглядати власні задачі, бачити історію комунікації з оператором, отримувати відповіді та користуватися базою знань. Для клієнта важливо, щоб система була простою: створити звернення,

побачити статус, отримати відповідь. Тому клієнтський функціонал не перевантажений зайвими елементами.

Оператор підтримки працює вже з більш складним набором можливостей. Він бачить призначені йому звернення, може переглядати термінові задачі, SLA-прострочки, відповідати клієнтам, додавати внутрішні нотатки, змінювати статуси та аналізувати рекомендації Smart Triage Radar. Тобто оператор отримує не просто список заявок, а повноцінне робоче місце для щоденної обробки звернень.

Адміністратор має найширші права. Він може переглядати всі задачі, вручну призначати або переназначати операторів, контролювати проблемні звернення, керувати користувачами, змінювати ролі, відділи та активність акаунтів. Такий підхід зроблений тому, що в реальній службі підтримки адміністратор або керівник повинен бачити загальну картину, а не тільки свої особисті задачі.

Особливістю реалізованого застосунку є наявність декількох модулів автоматизації. Найважливішими з них є Smart Triage Radar**,** Operator Load Balancer**,** SLA Radar та Activity Log. Вони дозволяють системі не просто зберігати дані, а допомагати в роботі оператора: визначати терміновість, рекомендувати наступний крок, автоматично призначати звернення та фіксувати всі важливі дії.

5.2 Реалізація серверної частини

Серверна частина web-застосунку реалізована мовою Java з використанням фреймворку Spring Boot. Такий вибір є доцільним для системи підтримки клієнтів, оскільки Spring Boot дозволяє швидко створювати REST API, працювати з базою даних, налаштовувати авторизацію та розділяти логіку застосунку на контролери, сервіси, репозиторії та сутності [11-13; 18; 19].

Для реалізації безпеки використано Spring Security та JWT-авторизацію. Після входу в систему користувач отримує токен, який надалі використовується для доступу до захищених API. Це дозволяє перевіряти роль користувача та обмежувати доступ до окремих функцій. Наприклад, сторінка керування

користувачами доступна тільки адміністратору, а клієнт не може переглядати чужі звернення [12; 18; 19].

Робота з базою даних реалізована за допомогою Spring Data JPA. Це дозволило працювати з таблицями бази даних через Java-сутності. Для збереження даних використано PostgreSQL, оскільки система має працювати зі структурованими даними: користувачами, зверненнями, повідомленнями, журналом дій і статтями бази знань [14; 15; 18].

Для керування змінами структури бази даних використано Liquibase. У проєкті додано міграцію 002-enhance-support-platform.yaml, яка розширює систему новими полями та таблицями. Зокрема, були додані поля для профілю користувача, SLA, Smart Triage, призначення оператора та журналу дій. Це зручно, тому що структура бази даних не змінюється вручну хаотично, а контролюється через міграції [14; 15].

Серверна частина містить декілька основних модулів:

– модуль авторизації та ролей;
– модуль роботи з користувачами;
– модуль звернень;
– модуль повідомлень і внутрішніх нотаток;
– модуль Smart Triage Radar;
– модуль Operator Load Balancer;
– модуль SLA Radar;
– модуль журналу дій;
– модуль бази знань;
– модуль аналітики для dashboard.

Модуль авторизації відповідає за вхід користувача, перевірку email і пароля, формування JWT-токена та повернення інформації про роль користувача. Саме на основі цієї ролі frontend надалі будує навігацію та приховує або показує певні сторінки [12; 18].

Модуль звернень є центральним у серверній частині. Він відповідає за створення звернення, отримання списку звернень, перегляд деталей, зміну статусу,

додавання повідомлень, внутрішніх нотаток, призначення оператора та оновлення службової інформації. Тут же відбувається запуск Smart Triage Radar і логіки автоматичного призначення оператора [4; 5].

Окремо було реалізовано API для адміністратора. Через нього адміністратор може переглядати список користувачів, змінювати роль, активність, відділ і посаду користувача. Це важливо, тому що система має не просто обслуговувати клієнтів, а й дозволяти адміністратору керувати командою підтримки.

5.3 Реалізація клієнтської частини

Клієнтська частина web-застосунку реалізована з використанням React**,** Vite, React Router та Axios. React використовується для створення компонентного інтерфейсу, Vite — для швидкої збірки та запуску проєкту, React Router — для переходів між сторінками, а Axios — для виконання HTTP-запитів до backend [16; 17; 20].

Основна логіка клієнтської частини побудована навколо ролей користувачів. Після входу в систему frontend отримує інформацію про користувача та його роль. Далі залежно від ролі формується доступна навігація. Наприклад, клієнт бачить свої звернення та базу знань, оператор — “Мої задачі”, “Огляд” і деталі звернень, а адміністратор — додатково сторінки “Всі задачі” та “Користувачі” [16; 20].

Такий підхід зробив інтерфейс більш зрозумілим. Користувач не бачить зайвих сторінок, до яких не має доступу. Це одночасно покращує зручність і зменшує ризик помилкових дій. Наприклад, клієнту не потрібно бачити адмінську сторінку користувачів, а оператору не потрібно мати можливість змінювати ролі інших користувачів.

Клієнтська частина містить такі основні сторінки:

– огляд**;**
– всі задачі для адміністратора;
– мої задачі для оператора;
– мої задачі для клієнта;

– деталі звернення**;**
– база знань**;**
– профіль**;**
– користувачі**.**

У процесі реалізації dashboard був оновлений так, щоб аналітика була компактною, без надмірно великих блоків. Це зроблено для того, щоб оператор або адміністратор міг швидко побачити загальну ситуацію: кількість звернень, пріоритети, SLA, triage-score та поточне навантаження [16; 20].

5.4 Опис основних сторінок web-застосунку

5.4.1 Сторінка “Огляд”

Сторінка **“**Огляд**”** використовується як загальний реєстр звернень. На ній відображаються звернення клієнтів, фільтри, компактна аналітика, SLA-стан, пріоритети та Smart Triage score. Така сторінка корисна для швидкого перегляду загальної ситуації в системі [4; 5].

На сторінці можна побачити, які звернення мають високий пріоритет, які ще не призначені оператору, які скоро можуть порушити SLA, а які вже прострочені. Для системи підтримки це важливо, бо оператор або адміністратор не повинен вручну відкривати кожне звернення, щоб зрозуміти, де є проблема.

Фільтри дозволяють швидко знайти потрібні звернення за статусом, категорією, пріоритетом або іншими параметрами. У реальному використанні це суттєво економить час, особливо якщо в системі є багато звернень.

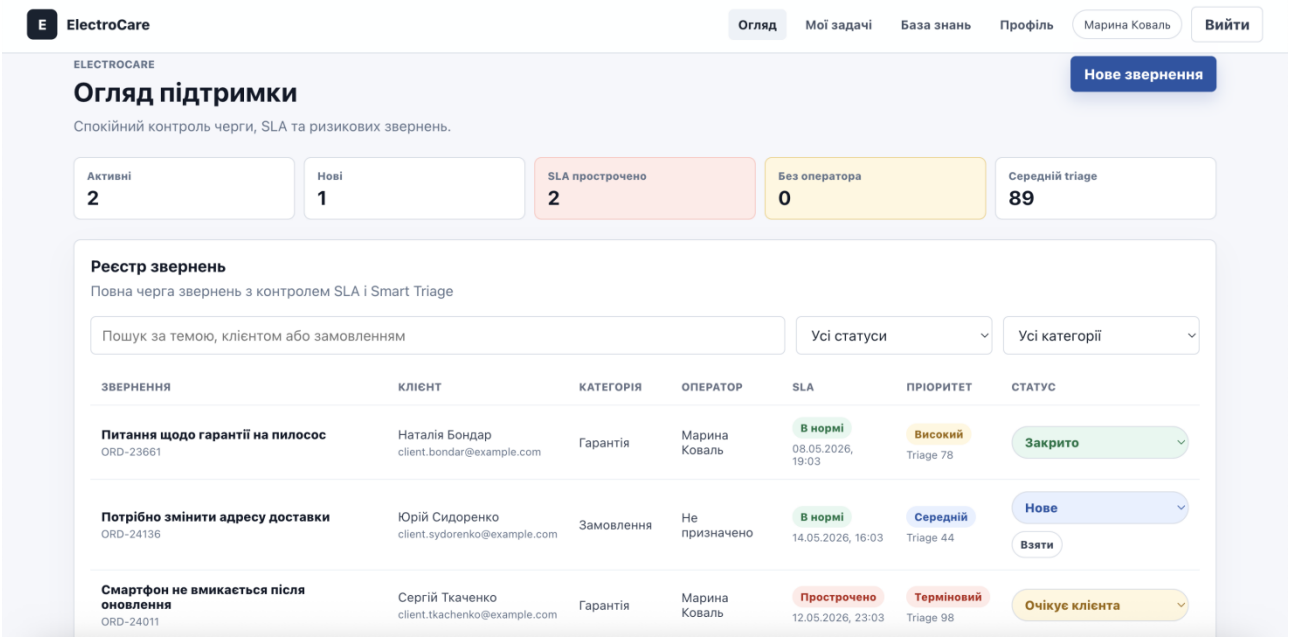


Рис. 5.1 Сторінка “Огляд” web-застосунку ElectroCare

5.4.2 Сторінка “Всі задачі” для адміністратора

Сторінка **“**Всі задачі**”** доступна адміністратору та використовується для контролю всіх звернень у системі. Якщо оператор бачить переважно власну чергу, то адміністратор бачить повну картину. Це потрібно для ручного призначення, переназначення операторів і перевірки проблемних звернень [4; 5].

На цій сторінці адміністратор може визначити, які задачі залишилися без відповідального оператора, які звернення мають прострочений SLA, які мають високий triage-score, а також які оператори можуть бути перевантажені. Якщо автоматичне призначення не підходить або ситуація потребує втручання, адміністратор може вручну змінити відповідального оператора.

Така сторінка фактично виконує роль контрольної панелі керівника служби підтримки. Вона дозволяє не просто реагувати на окремі звернення, а керувати процесом обробки в цілому.

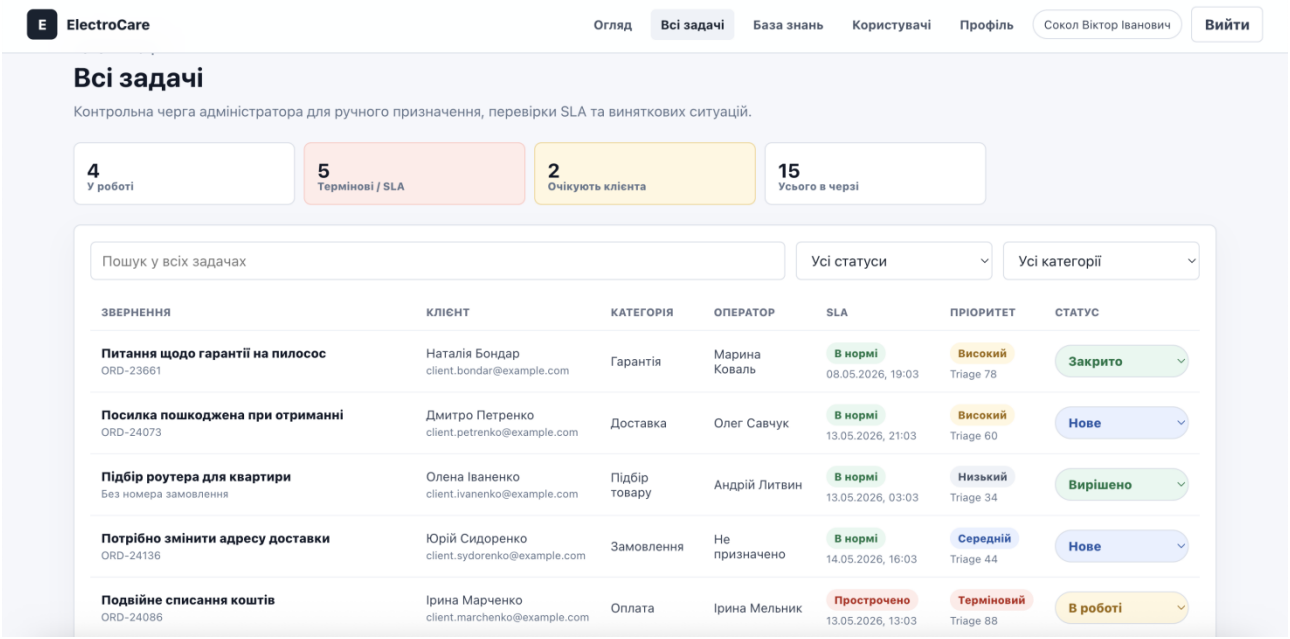


Рис. 5.2 Сторінка “Всі задачі” для адміністратора

5.4.3 Сторінка “Мої задачі” для оператора

Сторінка “Мої задачі” для оператора відображає персональну чергу звернень, які призначені конкретному працівнику підтримки. На цій сторінці оператор бачить свої активні задачі, термінові звернення, SLA-прострочки та може швидко перейти до детальної сторінки звернення [4; 5].

Ця сторінка потрібна для щоденної роботи оператора. Вона прибирає зайвий інформаційний шум, тому що оператор бачить не весь потік звернень, а саме ті задачі, за які він відповідає. Якщо звернення ще не призначене, оператор може взяти його в роботу, якщо має відповідні права.

Також на сторінці передбачено швидкі дії. Це дозволяє оператору не витрачати зайвий час на пошук потрібної функції, а швидко перейти до обробки звернення, відповіді клієнту або зміни статусу.

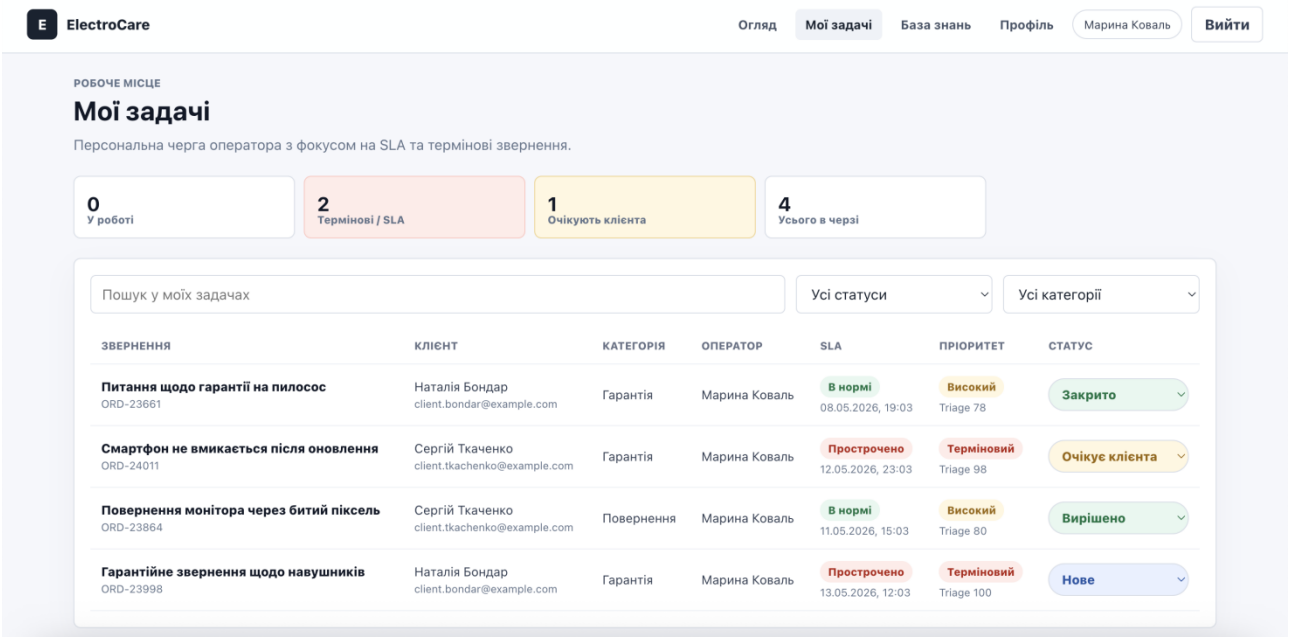


Рис. 5.3 Сторінка “Мої задачі” для оператора

5.4.4 Сторінка “Мої задачі” для клієнта

Для клієнта сторінка **“**Мої задачі**”** має інше значення. Вона відображає список його власних звернень та історію комунікації зі службою підтримки. Клієнт може переглянути, які звернення вже створені, який у них статус, чи є відповідь оператора та на якому етапі знаходиться обробка [4; 5].

Це вирішує типову проблему, коли клієнт не розуміє, чи отримала служба підтримки його повідомлення. У системі ElectroCare звернення має статус, дату створення, історію повідомлень і відповідь оператора. Тому клієнт може самостійно перевірити стан свого запиту без повторного звернення.

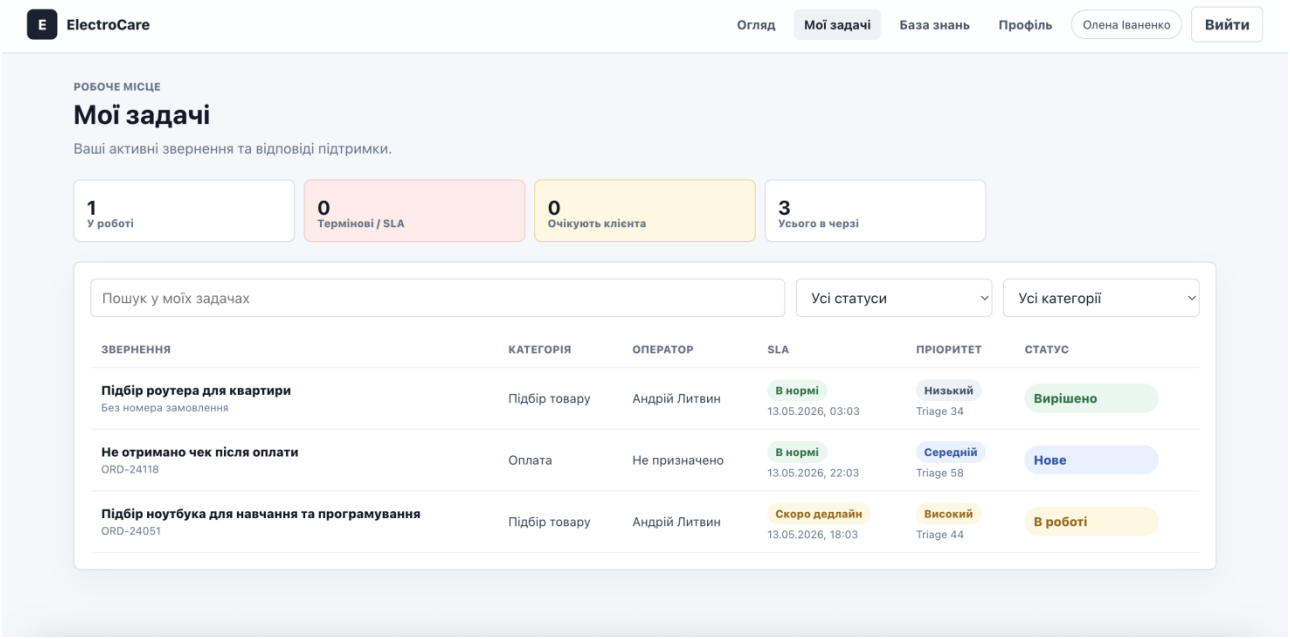


Рис. 5.4 Сторінка “Мої задачі” для клієнта

5.4.5 Сторінка “Деталі звернення”

Сторінка **“**Деталі звернення**”** є однією з головних сторінок системи. Саме тут відбувається основна робота зі зверненням. На сторінці відображаються тема, опис, категорія, статус, пріоритет, призначений оператор, SLA, журнал дій, переписка з клієнтом, внутрішні нотатки та рекомендація Smart Triage [4; 5].

Для оператора ця сторінка є робочим простором. Він бачить не тільки текст звернення, а й службову інформацію, яка допомагає швидше прийняти рішення. Наприклад, якщо triage-score високий, оператор розуміє, що звернення потребує швидкої реакції. Якщо SLA-дедлайн скоро буде порушено, система також показує це як важливий сигнал.

На цій сторінці оператор може відповісти клієнту або додати внутрішню нотатку. Внутрішні нотатки потрібні для службової комунікації, наприклад, коли оператор хоче залишити коментар для іншого працівника або зафіксувати деталі перевірки, які не повинні бути видимі клієнту.

Також на сторінці відображається Activity Log. Він показує історію важливих дій: створення звернення, автоматичне призначення, зміну статусу, додавання

повідомлення, внутрішню нотатку або переназначення оператора. Це зручно, бо можна побачити, що саме відбувалося зі зверненням і хто виконував дії.

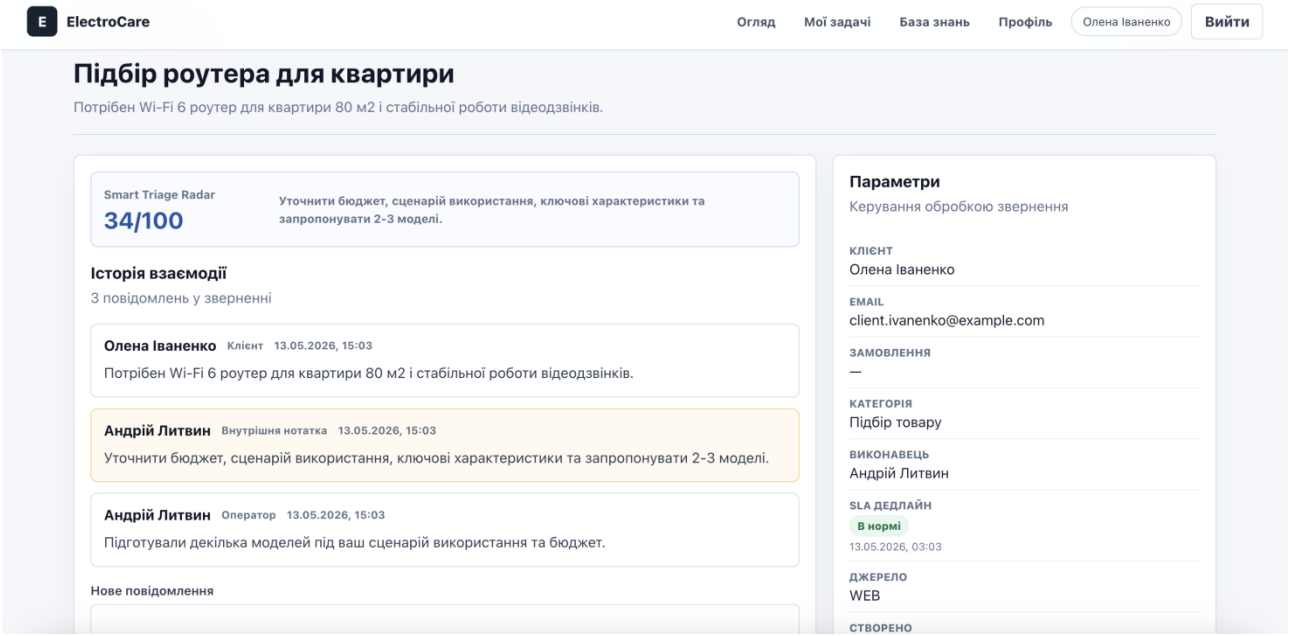


Рис. 5.5 Сторінка деталей звернення

5.4.6 Сторінка “База знань”

Сторінка “База знань” містить довідкові матеріали та інструкції для операторів. У системі передбачено статті з таких напрямів: гарантія, доставка, оплата, повернення, підбір електроніки та інші типові ситуації, які часто виникають у клієнтів інтернет-магазину [4; 5].

База знань важлива, тому що оператор не повинен кожного разу формулювати відповідь з нуля. Якщо є типова ситуація, він може орієнтуватися на готову інструкцію. Це підвищує швидкість відповіді та зменшує кількість помилок у комунікації [4; 5].

Крім того, база знань пов’язана з модулем Smart Triage Radar. Система може рекомендувати відповідну статтю бази знань для конкретного звернення. Наприклад, якщо клієнт питає про повернення товару, система може запропонувати статтю з правилами повернення.

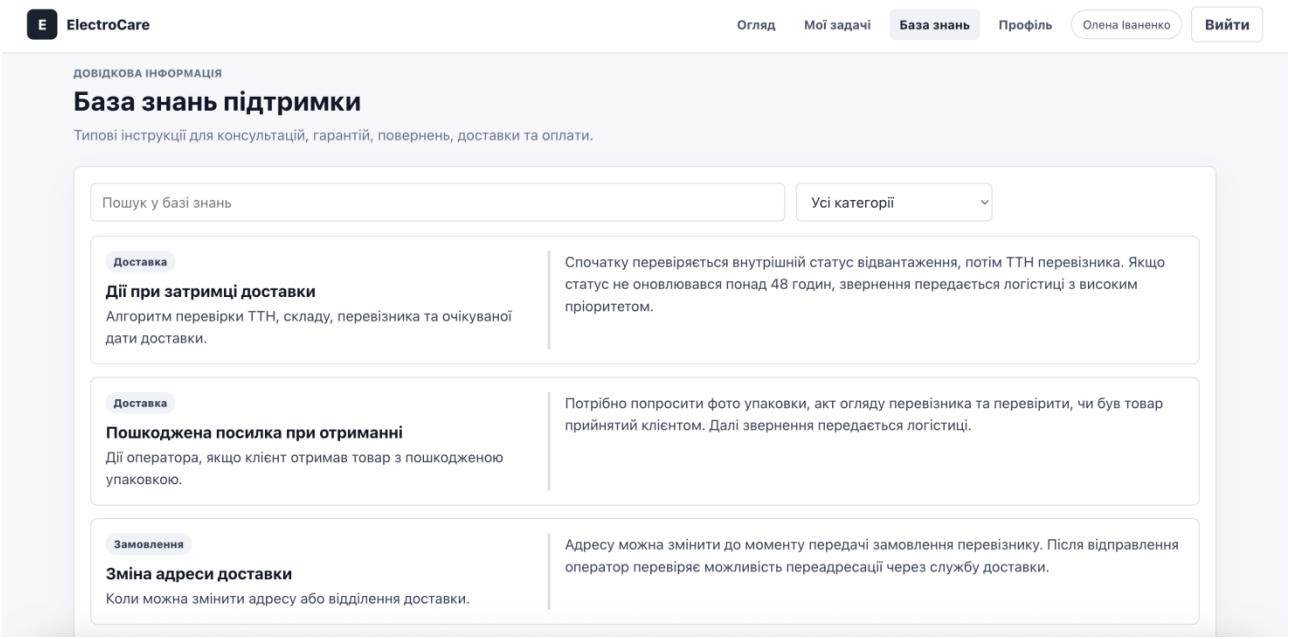


Рис. 5.6 Сторінка бази знань

5.4.7 Сторінка “Профіль”

Сторінка “Профіль” дозволяє користувачу редагувати власні дані. У профілі можна змінити ім’я, телефон, відділ, посаду та колір профілю. Для клієнта це базова персональна інформація, а для оператора — ще й службові дані, які можуть бути корисні для організації роботи підтримки.

Наприклад, оператор може належати до відділу доставки, гарантії, оплати або продажів. Ці дані можуть використовуватися для більш логічного розподілу звернень між операторами. Тобто профіль у системі не просто декоративний, а має практичне значення.

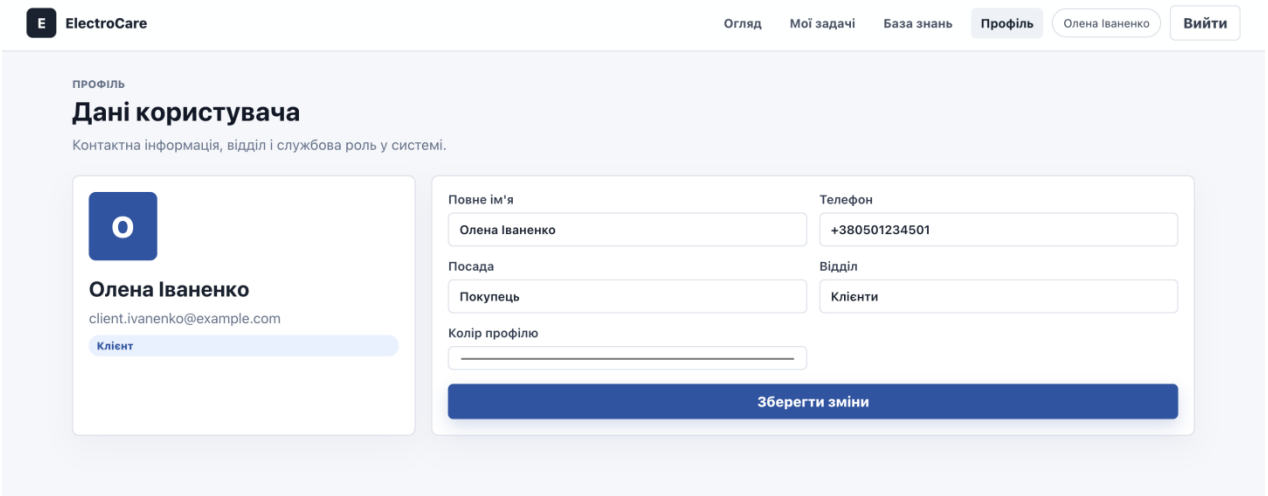


Рис. 5.7 Сторінка профілю користувача

5.4.8 Сторінка “Користувачі”

Сторінка “Користувачі” доступна тільки адміністратору. Вона призначена для керування ролями, відділами та активністю акаунтів. Адміністратор може переглядати список користувачів, змінювати роль, активувати або деактивувати обліковий запис, редагувати службові дані [4; 5].

Ця сторінка потрібна для підтримки актуальної структури користувачів системи. Наприклад, якщо працівник переходить з відділу доставки до відділу гарантії, адміністратор може змінити його дані. Якщо користувач більше не працює із системою, його акаунт можна деактивувати.

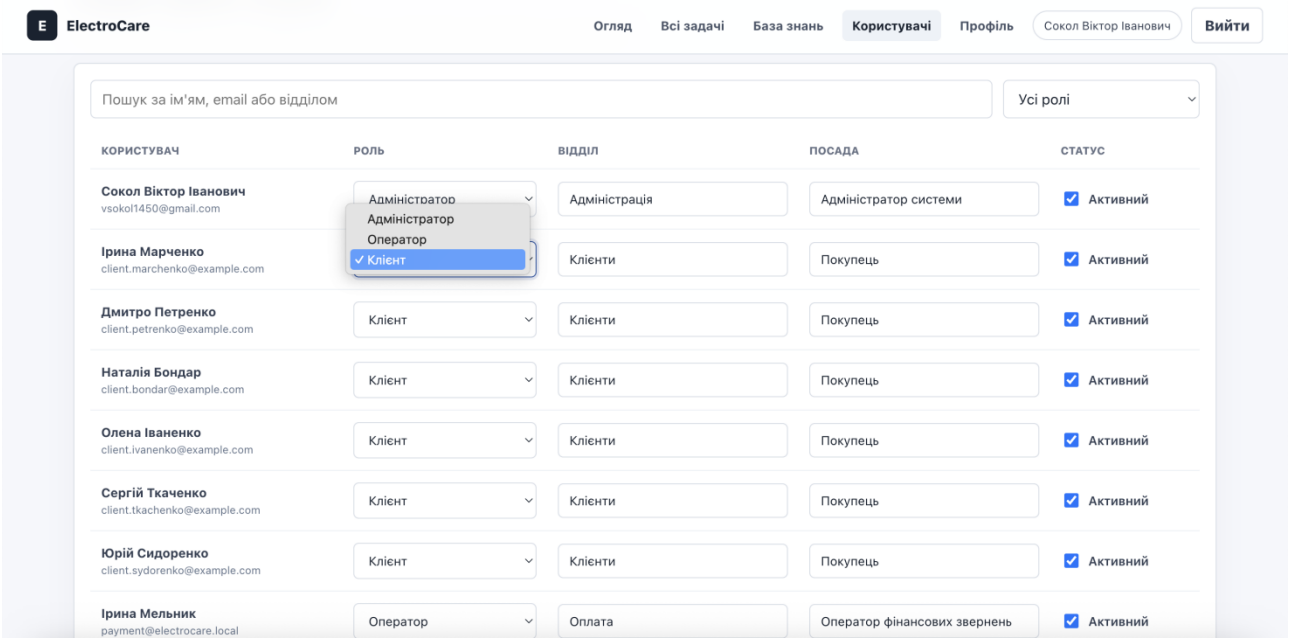


Рис. 5.8 Сторінка керування користувачами

5.5 Реалізація унікальних функцій системи

5.5.1 Smart Triage Radar

Однією з основних особливостей ElectroCare є модуль Smart Triage Radar. Його задача полягає в первинному аналізі нового звернення. Система аналізує категорію, опис і ключові слова, після чого формує оцінку терміновості — triage-score [4; 5].

Triage-score має значення від 0 до 100. Чим вищий показник, тим більш терміновим є звернення. Наприклад, звернення про гарантійний випадок, несправність товару або проблему з оплатою може отримати вищий бал, ніж загальне питання про характеристики товару.

На основі аналізу система також визначає підсумковий пріоритет, SLA-дедлайн, рекомендовану дію для оператора та відповідну статтю бази знань. Це не повноцінний зовнішній штучний інтелект, але для задач бакалаврського проєкту така логіка виглядає практичною: вона не замінює оператора, а допомагає йому швидше зрозуміти, з чого почати обробку звернення.

5.5.2 Operator Load Balancer

Модуль Operator Load Balancer відповідає за автоматичне призначення звернення оператору. Під час призначення система може враховувати категорію звернення та поточну завантаженість операторів [4; 5].

Наприклад, якщо звернення належить до категорії доставки, його логічно передати оператору відповідного відділу. Якщо в одного оператора вже багато активних задач, система може обрати іншого менш завантаженого працівника. Така логіка дозволяє більш рівномірно розподіляти навантаження.

Водночас система не забирає контроль в адміністратора. Адміністратор може вручну переназначити звернення, якщо бачить, що автоматичний вибір потрібно скоригувати. Це важливо, бо в реальній роботі не всі ситуації можна ідеально автоматизувати.

5.5.3 SLA Radar

Модуль SLA Radar використовується для контролю строків обробки звернень. Кожне звернення може мати SLA-дедлайн, тобто час, до якого його потрібно опрацювати або надати відповідь [4; 5].

SLA Radar дозволяє визначити три основні стани:

– звернення в нормі;
– дедлайн скоро настане;

– SLA вже прострочено.

Ця інформація відображається в інтерфейсі, щоб оператор або адміністратор міг швидко побачити проблемні звернення. Для служби підтримки це дуже важливо, тому що прострочені звернення напряму впливають на якість сервісу та задоволеність клієнтів.

5.5.4 Activity Log

Activity Log — це журнал дій за зверненням. Він фіксує всі важливі події, які відбуваються в системі. До таких подій належать створення звернення, автоматичне призначення оператора, зміна статусу, додавання повідомлення, внутрішня нотатка, переназначення оператора та інші службові дії [4; 5].

Журнал дій потрібен для прозорості. Якщо виникає питання, хто змінив статус або коли оператор взяв звернення в роботу, це можна побачити в Activity Log. У звичайній ручній підтримці така інформація часто губиться, а тут вона зберігається системно.

5.5.5 Рольовий доступ

У системі реалізовано рольовий доступ. Це означає, що користувачі з різними ролями бачать різні сторінки та мають різні права. Клієнт не може переглядати чужі звернення або керувати користувачами. Оператор не може змінювати ролі інших користувачів. Адміністратор має доступ до ширшого набору функцій [12; 18].

Рольовий доступ реалізований і на рівні frontend, і на рівні backend. На frontend приховуються недоступні сторінки, а backend додатково перевіряє права при виконанні API-запитів. Це правильно, бо приховати кнопку в інтерфейсі недостатньо: сервер також повинен перевіряти, чи має користувач право виконувати дію.

5.6 Тестування програмного забезпечення

Тестування web-застосунку ElectroCare проводилося для перевірки коректності основних функцій системи. Оскільки застосунок має серверну та клієнтську частини, перевірка виконувалася на декількох рівнях: backend-тести, збірка frontend, перевірка авторизації, перевірка рольового доступу та ручне проходження основних сценаріїв [11-20].

Для backend-тестування використовувалися JUnit 5**,** Mockito **та** Spring Boot Test**.** Ці інструменти дозволяють перевіряти роботу сервісів, контролерів і логіки системи без необхідності вручну виконувати кожен сценарій через інтерфейс [12; 18].

Тестування охоплювало такі основні сценарії:

– авторизація адміністратора, оператора та клієнта;

– створення нового звернення;

– автоматичне визначення triage-score;

– формування пріоритету та SLA-дедлайну;

– автоматичне призначення оператора;

– перегляд персональної черги оператора;

– перегляд власних звернень клієнтом;

– додавання відповіді;

– додавання внутрішньої нотатки;

– зміна статусу звернення;

– перевірка журналу дій;

– керування користувачами адміністратором.

Окремо була перевірена збірка клієнтської частини командою npm run build. Це потрібно для того, щоб переконатися, що frontend не має помилок збірки та може бути підготовлений до розгортання.

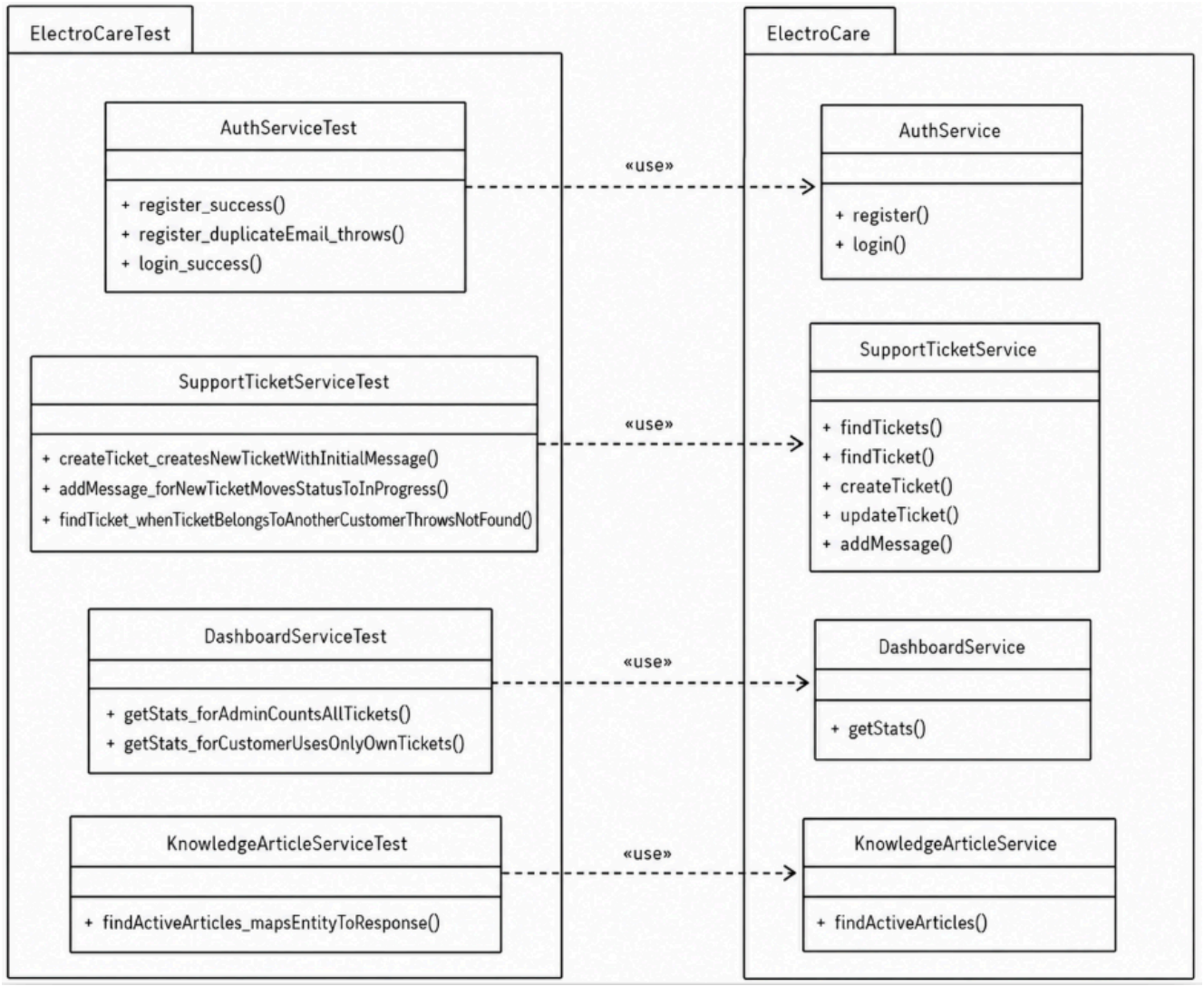


Рис. 5.9 Схема тестування web-застосунку ElectroCare

5.7 Тест-кейси web-застосунку ElectroCare

Для перевірки функціональності web-застосунку було сформовано набір тест-кейсів. Вони охоплюють основні сценарії роботи адміністратора, оператора та клієнта.

Таблиця 5.1

Тест-кейси web-застосунку ElectroCare

ID	**Сценарій**	**Дії**	**Очікуваний результат**	**Фактичний результат**	**Статус**
TC-01	Авторизація адміністратора	Ввести email і пароль адміністратора	Адміністратор входить у систему та бачить доступні адмінські сторінки	Відповідає очікуванню	Пройдено
TC-02	Авторизація оператора	Ввести email і пароль оператора	Оператор входить у систему та бачить сторінку “Мої задачі”	Відповідає очікуванню	Пройдено
TC-03	Авторизація клієнта	Ввести email і пароль клієнта	Клієнт входить у систему та бачить власні звернення	Відповідає очікуванню	Пройдено
TC-04	Створення звернення клієнтом	Заповнити тему, опис, категорію та надіслати форму	Звернення створюється та зберігається в системі	Відповідає очікуванню	Пройдено
TC-05	Робота Smart Triage Radar	Створити звернення з ключовими словами термінової проблеми	Система визначає triage-score, пріоритет, SLA та рекомендовану дію	Відповідає очікуванню	Пройдено
TC-06	Автоматичне призначення оператора	Створити звернення певної категорії	Система призначає звернення відповідному оператору з урахуванням категорії та навантаження	Відповідає очікуванню	Пройдено
TC-07	Перегляд “Мої задачі” оператором	Відкрити сторінку “Мої задачі” під роллю OPERATOR	Відображаються звернення, призначені цьому оператору	Відповідає очікуванню	Пройдено
TC-08	Перегляд “Мої задачі” клієнтом	Відкрити сторінку “Мої задачі” під роллю CUSTOMER	Відображаються тільки власні звернення клієнта	Відповідає очікуванню	Пройдено
TC-09	Додавання відповіді оператором	Відкрити звернення та надіслати відповідь клієнту	Повідомлення додається до історії звернення	Відповідає очікуванню	Пройдено

Продовження таблиці 5.1

Тест-кейси web-застосунку ElectroCare

ID	**Сценарій**	**Дії**	**Очікуваний результат**	**Фактичний результат**	**Статус**
TC-10	Додавання внутрішньої нотатки	Оператор додає internal note	Нотатка зберігається як внутрішня та не є звичайною відповіддю клієнту	Відповідає очікуванню	Пройдено
TC-11	Зміна статусу звернення	Оператор змінює статус звернення	Статус оновлюється, дія фіксується в журналі	Відповідає очікуванню	Пройдено
TC-12	Перевірка SLA Radar	Відкрити звернення з наближенням або порушенням дедлайну	Система відображає відповідний SLA-стан	Відповідає очікуванню	Пройдено
TC-13	Перегляд журналу дій	Відкрити деталі звернення	Відображаються основні події за зверненням	Відповідає очікуванню	Пройдено
TC-14	Керування користувачами	Адміністратор змінює роль або активність користувача	Дані користувача оновлюються	Відповідає очікуванню	Пройдено
TC-15	Збірка frontend	Виконати команду npm run build	Клієнтська частина успішно збирається	Відповідає очікуванню	Пройдено

5.8 Результати тестування

За результатами тестування було підтверджено працездатність основних функцій web-застосунку ElectroCare. Backend-тести були виконані командою ./mvnw test. Результат виконання: 9 tests, 0 failures. Це означає, що автоматизовані тести пройшли успішно і критичних помилок у перевірених сервісних сценаріях не виявлено [12; 18].

Клієнтська частина була перевірена командою npm run build. Збірка виконана успішно, що підтверджує коректність структури frontend-проєкту та відсутність помилок, які блокують підготовку застосунку до запуску [16; 17].

Також було перевірено логін через Vite proxy. Це важливо для локальної розробки, оскільки frontend і backend працюють як окремі частини, але повинні коректно взаємодіяти між собою через API [19; 20].

Таблиця 5.2

Результати тестування web-застосунку ElectroCare

Вид перевірки	**Інструмент або спосіб перевірки**	**Результат**
Backend-тестування	Maven, JUnit 5, Mockito, Spring Boot Test	11 tests, 0 failures
Збірка клієнтської частини	npm run build	Успішно
Перевірка авторизації	Ручна перевірка через web-інтерфейс	Успішно
Перевірка рольового доступу	Ручна перевірка для ADMIN, OPERATOR, CUSTOMER	Успішно
Перевірка створення звернення	Ручна перевірка	Успішно
Перевірка Smart Triage Radar	Ручна перевірка на тестових зверненнях	Успішно
Перевірка Operator Load Balancer	Ручна перевірка призначення операторів	Успішно
Перевірка SLA Radar	Ручна перевірка дедлайнів	Успішно
Перевірка Activity Log	Ручна перевірка журналу дій	Успішно
Перевірка frontend-backend взаємодії	Vite proxy	Успішно

.

ВИСНОВКИ

1. Проведено аналіз предметної галузі підтримки клієнтів інтернет-магазину електроніки та визначено основні проблеми, які виникають під час ручної або частково автоматизованої обробки звернень. До таких проблем належать розрізнені канали комунікації, втрата історії взаємодії з клієнтом, складність контролю статусів, нерівномірне навантаження на операторів, дублювання типових відповідей та відсутність зручного контролю строків обробки звернень. [1-5].
2. Проведено огляд існуючих програмних засобів для автоматизації клієнтської підтримки, зокрема Zendesk, Freshdesk, Intercom та Help Scout. У результаті аналізу встановлено, що такі системи мають широкий набір функцій для роботи зі зверненнями, однак часто є комерційними, складними для налаштування або надлишковими для невеликого чи середнього інтернет-магазину електроніки. Це підтвердило доцільність розробки власного web-застосунку, адаптованого під конкретні задачі предметної галузі. [6-9].
3. Визначено функціональні та нефункціональні вимоги до web-застосунку ElectroCare. До основних функціональних вимог віднесено створення та обробку звернень, розмежування ролей адміністратора, оператора і клієнта, призначення звернень на операторів, контроль статусів, ведення переписки, використання внутрішніх нотаток, роботу з базою знань, журналом дій та аналітикою. До нефункціональних вимог віднесено швидке завантаження сторінок, захист доступу відповідно до ролей, структуроване збереження даних, підтримку одночасної роботи користувачів і можливість подальшого розширення системи.
4. Спроєктовано web-застосунок ElectroCare для автоматизації підтримки клієнтів інтернет-магазину електроніки. У процесі проєктування було побудовано логічну структуру системи, діаграму варіантів використання, діаграму доменної моделі, діаграми діяльності, діаграму класів, діаграму компонентів, діаграму розгортання та ERD-діаграму бази даних. Це

дозволило формалізувати структуру системи, основні ролі користувачів, бізнес-процеси, модулі програмного забезпечення та взаємозв'язки між основними сутностями.

5. Обрано та обґрунтовано засоби реалізації програмного забезпечення. Для серверної частини використано Java, Spring Boot, Spring Security, Spring Data JPA, JWT, Liquibase та PostgreSQL. Для клієнтської частини використано React, Vite, React Router та Axios. Такий стек технологій дозволив реалізувати клієнт-серверну архітектуру, REST API, рольовий доступ, роботу з базою даних, міграції структури БД та зручний web-інтерфейс для різних категорій користувачів. [11-20].
6. Розроблено web-застосунок ElectroCare, який забезпечує централізовану роботу зі зверненнями клієнтів. У системі реалізовано сторінки "Огляд", "Всі задачі", "Мої задачі", "Деталі звернення", "База знань", "Профіль" та "Користувачі". Адміністратор отримує можливість контролювати всі задачі, керувати користувачами, ролями, відділами та активністю акаунтів. Оператор може працювати з персональною чергою звернень, відповідати клієнтам, додавати внутрішні нотатки та контролювати SLA. Клієнт може створювати звернення, переглядати власні задачі та історію комунікації.
7. Реалізовано додаткові функції автоматизації, які підвищують практичну цінність системи. Модуль Smart Triage Radar аналізує нове звернення за категорією, описом і ключовими словами, після чого визначає оцінку терміновості, підсумковий пріоритет, SLA-дедлайн і рекомендовану дію для оператора. Модуль Operator Load Balancer виконує автоматичне призначення звернення оператору з урахуванням категорії та поточного навантаження. Модуль SLA Radar дозволяє визначати звернення, які перебувають у нормі, наближаються до дедлайну або вже прострочені. Activity Log фіксує основні дії за зверненням: створення, автоназначення, зміну статусу, повідомлення, внутрішні нотатки та переназначення.
8. Проведено тестування розробленого web-застосунку ElectroCare. Backend-тести виконано за допомогою Maven, JUnit 5, Mockito та Spring Boot Test.

Результат виконання тестів становить 11 tests, 0 failures. Клієнтську частину перевірено командою npm run build, збірка виконана успішно. Також проведено ручну перевірку авторизації, рольового доступу, створення звернення, роботи Smart Triage Radar, Operator Load Balancer, SLA Radar, Activity Log, сторінок “Мої задачі”, “Всі задачі”, “Деталі звернення”, “База знань”, “Профіль” та “Користувачі”. Отримані результати підтвердили працездатність основних функцій системи. [12; 18].

9. Результати виконаної роботи можуть бути використані як основа для подальшого розвитку системи підтримки клієнтів інтернет-магазину електроніки. У майбутньому web-застосунок ElectroCare можна розширити інтеграцією з онлайн-чатом, Telegram-ботом, CRM-системою, email-сповіщеннями, розширеною аналітикою роботи операторів та AI-модулем для автоматичного формування відповідей на типові звернення.

Результати дослідження апробовано та опубліковано у тезах доповідей на науково-практичних конференціях, що відбулися на базі Державного університету інформаційно-комунікаційних технологій:

- II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Автоматизована система підтримки із telegram-ботом і нейромережею для відповідей.
- Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, Київ, ДУІКТ. Актуальність системи підтримки клієнтів інтернет-магазину електроніки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Краус К. М., Краус Н.М., Манжура О. В. Електронна комерція та Інтернет-торгівля : навчально-методичний посібник. Київ : Аграр Медіа Груп, 2021. 454 с. URL: https://elibrary.kubg.edu.ua/37044/1/Kraus_Elektronna_komertsiia_2021.pdf (дата звернення: 01.05.2026).
2. Що таке e-commerce, приклади, інструменти, як створити // Kyivstar Business Hub. URL: https://hub.kyivstar.ua/articles/shho-take-e-commerce (дата звернення: 01.05.2026).
3. E-Commerce — що таке електронна комерція? // WEZOM. URL: https://wezom.com.ua/ua/blog/elektronna-komertsiya (дата звернення: 01.05.2026).
4. Топ 10: Helpdesk системи підтримки клієнтів для України // LiveBusiness. URL: https://www.livebusiness.com.ua/ua/tags/helpdesk/ (дата звернення: 01.05.2026).
5. Як обрати CRM-систему для інтернет-магазину // Detector Media. URL: https://detector.media/infospace/article/221557/2024-01-10-yak-obraty-crm-systemu-dlya-internet-magazynu/ (дата звернення: 01.05.2026).
6. Zendesk Ticketing System // Zendesk. URL: https://www.zendesk.com/service/ticketing-system/ (дата звернення: 01.05.2026).
7. Freshdesk Support Desk // Freshworks. URL: https://www.freshworks.com/freshdesk/ (дата звернення: 01.05.2026).
8. Intercom Features Explained // Intercom Help Center. URL: https://www.intercom.com/help/en/articles/591233-intercom-features-explained (дата звернення: 01.05.2026).
9. What is a Shared Inbox? // Help Scout Docs. URL: https://docs.helpscout.com/article/1581-what-is-shared-inbox (дата звернення: 01.05.2026).

10. Service Hub Software // HubSpot. URL: https://www.hubspot.com/products/service (дата звернення: 01.05.2026).
11. Java SE Documentation // Oracle. URL: https://docs.oracle.com/en/java/javase/21/ (дата звернення: 01.05.2026).
12. Spring Boot Documentation // Spring. URL: https://docs.spring.io/spring-boot/documentation.html (дата звернення: 01.05.2026).
13. Spring Boot // Spring. URL: https://spring.io/projects/spring-boot (дата звернення: 01.05.2026).
14. PostgreSQL Documentation // PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 01.05.2026).
15. PostgreSQL: The World's Most Advanced Open Source Relational Database // PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 01.05.2026).
16. React Documentation // React. URL: https://react.dev/ (дата звернення: 01.05.2026).
17. TypeScript Documentation // TypeScript. URL: https://www.typescriptlang.org/docs/ (дата звернення: 01.05.2026).
18. Spring Framework Documentation // Spring. URL: https://docs.spring.io/spring-framework/reference/ (дата звернення: 01.05.2026).
19. RESTful Web Services // Spring Guides. URL: https://spring.io/guides/gs/rest-service/ (дата звернення: 01.05.2026).
20. React Router Documentation // React Router. URL: https://reactrouter.com/ (дата звернення: 01.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

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

Автоматизована система підтримки клієнтів інтернет-магазину електроніки

Виконав студент 4 курсу
групи ПД-42
Сокол Віктор Іванович
Керівник роботи
ст. викладач Гаманюк Ігор Михайлович

Київ – 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі підтримки клієнтів інтернет-магазину електроніки.
2. Дослідити існуючі програмні засоби для автоматизації клієнтської підтримки.
3. Визначити основні проблеми обробки звернень клієнтів.
4. Сформувати функціональні та нефункціональні вимоги до системи.
5. Спроєктувати структуру базу даних.
6. Реалізувати web-застосунок ElectroCare з ролями адміністратора, оператора та клієнта.
7. Провести тестування розробленого програмного забезпечення.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення підтримки клієнтів шляхом впровадження інтернет-магазину електроніки.

Об'єкт дослідження: процеси підтримки клієнтів інтернет-магазину електроніки.

Предмет дослідження: засоби, методи та технології реалізації автоматизації підтримки клієнтів інтернет-магазину електроніки.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1.Реєстрація та авторизація користувачів з ролями.
2.Створення клієнтом звернення із вибором категорії та описом проблеми.
3.Категоризація звернень мінімум за типами.
4.Відображення статусу звернення.
5.Фільтрація звернень оператором за статусом, категорією, датою та клієнтом.
6.Використання шаблонів відповідей для типових запитів.
7.Формування звітів за кількістю звернень, статусами, категоріями та навантаженням операторів.

Нефункціональні вимоги

1.Час завантаження основних сторінок системи — не більше 3 секунд.
2.Час відповіді сервера на стандартний запит — до 1 секунди.
3.Система повинна підтримувати одночасну роботу не менше 50 користувачів.
4.Дані користувачів і звернень повинні зберігатися в базі даних PostgreSQL.
5.Доступ до функцій системи має обмежуватися відповідно до ролі користувача.
6.Інтерфейс повинен бути адаптований для екранів від 1366×768 px і вище.
7.Система повинна забезпечувати збереження 10 000+ звернень без втрати продуктивності.

АНАЛІЗ АНАЛОГІВ

Показник	Zendesk	Freshdesk	Intercom	Help Scout	Electro Care
Система тікетів	+	+	+	+	+
Категоризація звернень	+	+	+	базова категоризація без глибокої логіки	+
Контроль статусів	+	+	+	+	+
База знань	+	+	+	+	+
Чат з клієнтом	+	+	+	переважно через shared inbox	+
Аналітика та звіти	+	+	+	базові звіти без розширеної аналітики	+
Автоматизація процесів	+	+	+	обмежені автоматичні сценарії	+
Smart Triage score для звернення	-	-	-	-	+
Автоматичне призначення оператора на звернення	-	-	-	-	+
SLA Radar для швидкого виявлення прострочених звернень	SLA є, але без окремого radar-модуля	SLA є, але без окремого radar-модуля	SLA/пріоритети є, але без окремого radar-модуля	-	+
Орієнтація на малий бізнес	потребує складнішого налаштування	+	більше орієнтований на активні продажі та чат	+	+
Орієнтація на інтернет-магазин електроніки	потребує адаптації під магазин	потребує адаптації під магазин	потребує адаптації під магазин	потребує адаптації під магазин	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Реєстрація та авторизація користувачів з ролями.
2. Створення клієнтом звернення із вибором категорії та описом проблеми.
3. Категоризація звернень за типами.
4. Відображення статусу звернення.
5. Фільтрація звернень оператором за статусом, категорією, датою та клієнтом.
6. Використання шаблонів відповідей для типових запитів.
7. Формування звітів за кількістю звернень, статусами, категоріями та навантаженням операторів.

Нефункціональні вимоги

1. Час завантаження сторінок системи — не більше 3 секунд.
2. Час відповіді сервера на стандартний запит — до 1 секунди.
3. Система повинна підтримувати одночасну роботу не менше 50 користувачів.
4. Дані користувачів і звернень повинні зберігатися в базі даних PostgreSQL.
5. Доступ до функцій системи має обмежуватися відповідно до ролі користувача.
6. Інтерфейс повинен бути адаптований для екранів від 1366×768 px і вище.
7. Система повинна забезпечувати збереження 10 000+ звернень без втрати продуктивності.

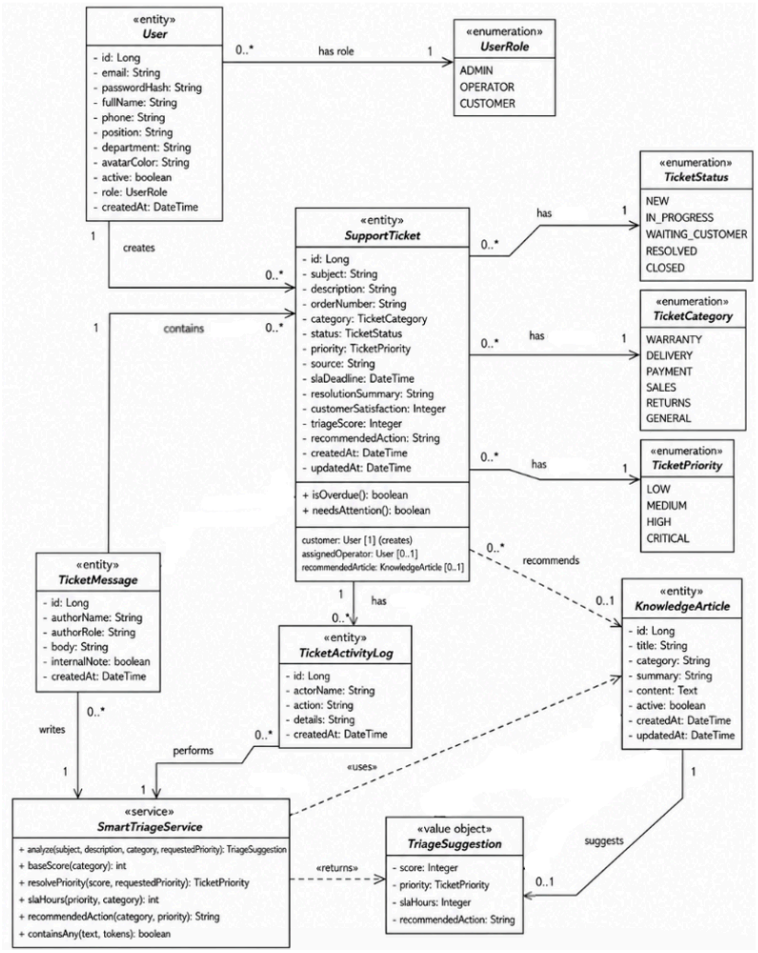
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

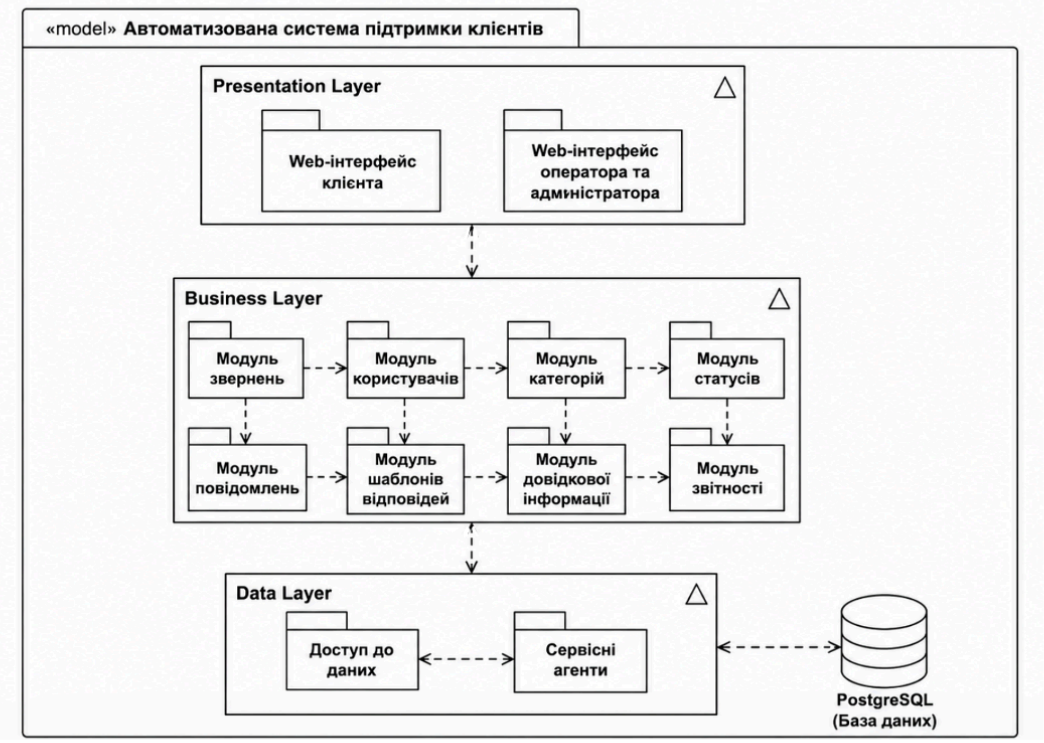
7

ДІАГРАМА ДОМЕННОЇ МОДЕЛІ



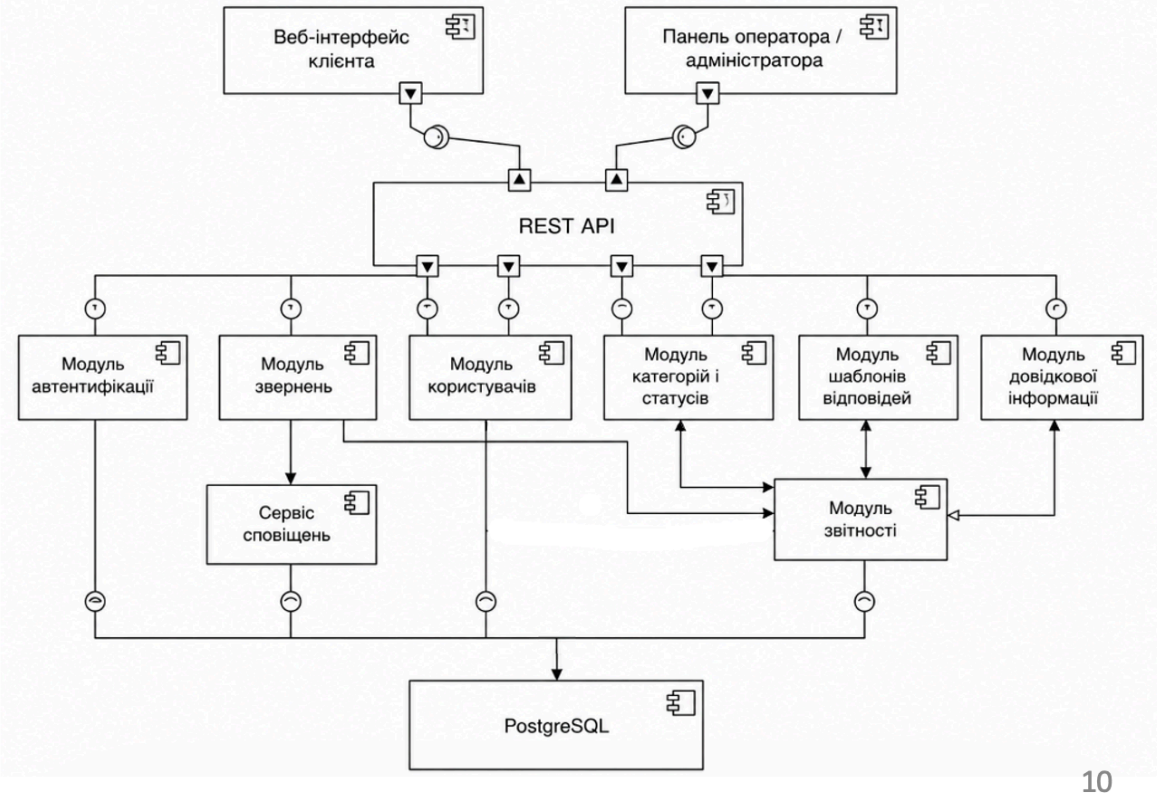
8

ЛОГІЧНА АРХІТЕКТУРА СИСТЕМИ



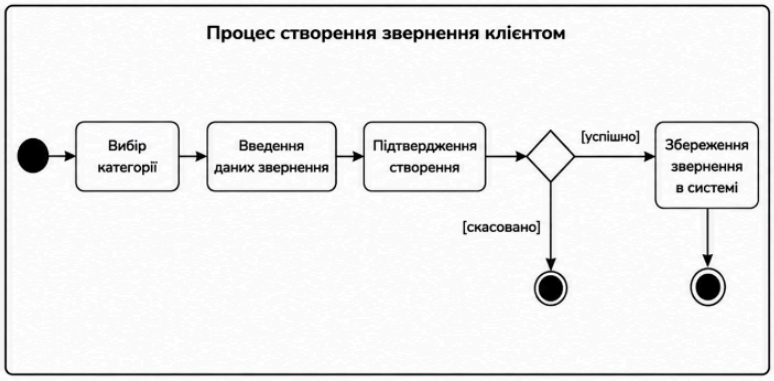
9

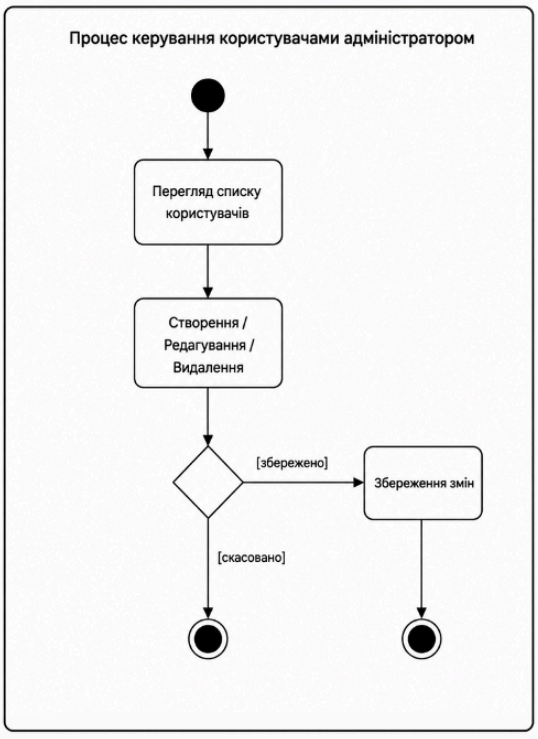
ДІАГРАМА КОМПОНЕНТІВ



10

ДІАГРАМИ ДІЯЛЬНОСТІ





11

ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ

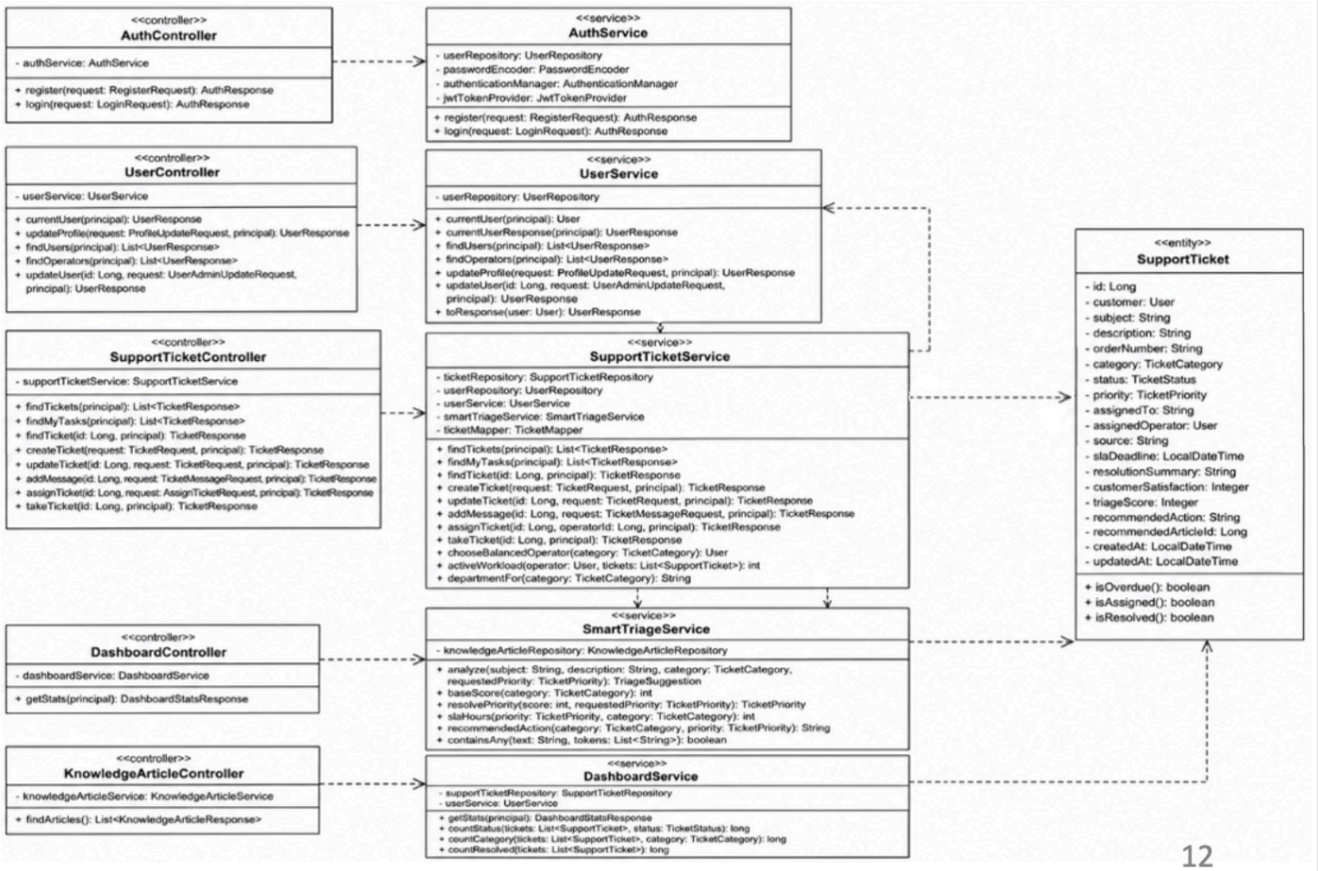
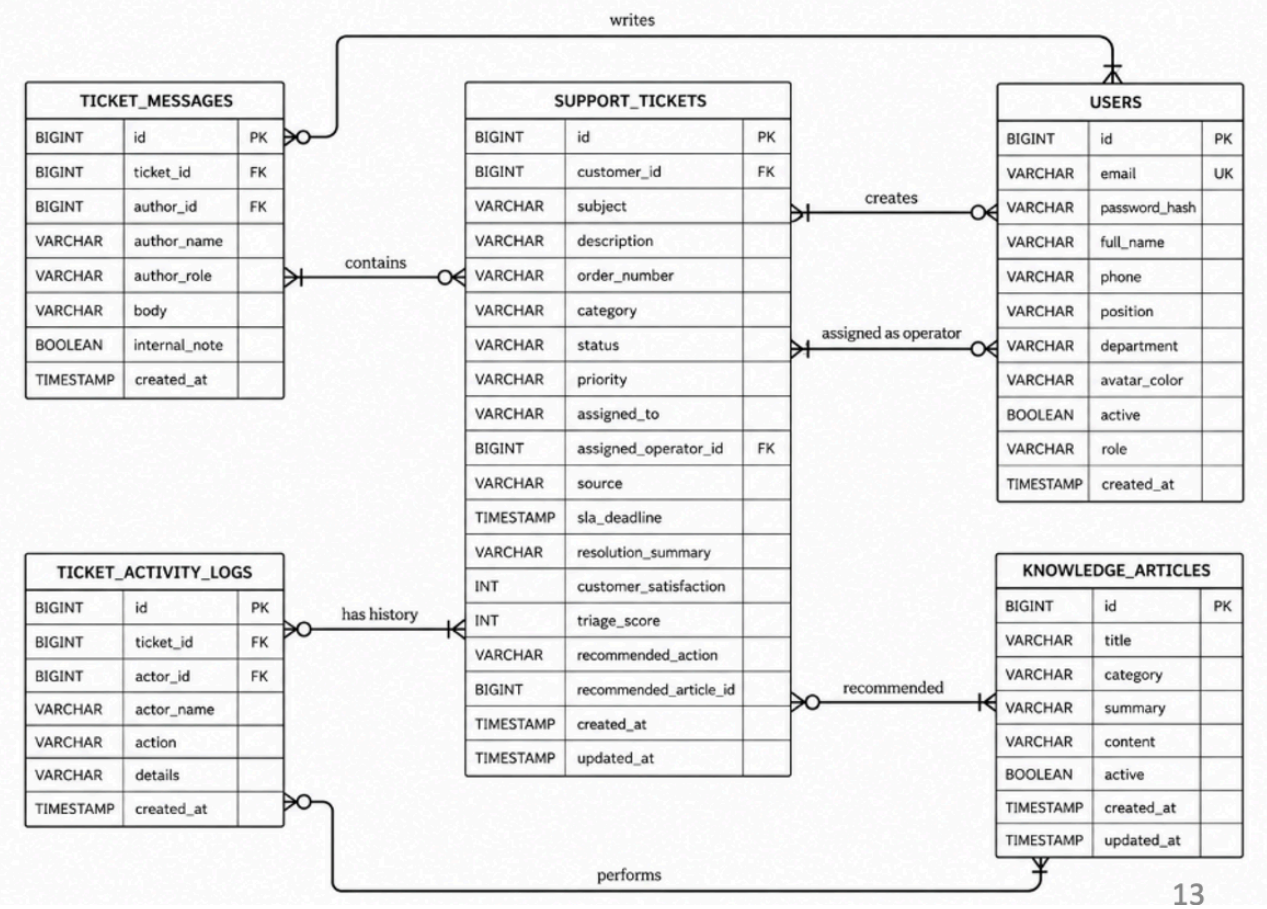
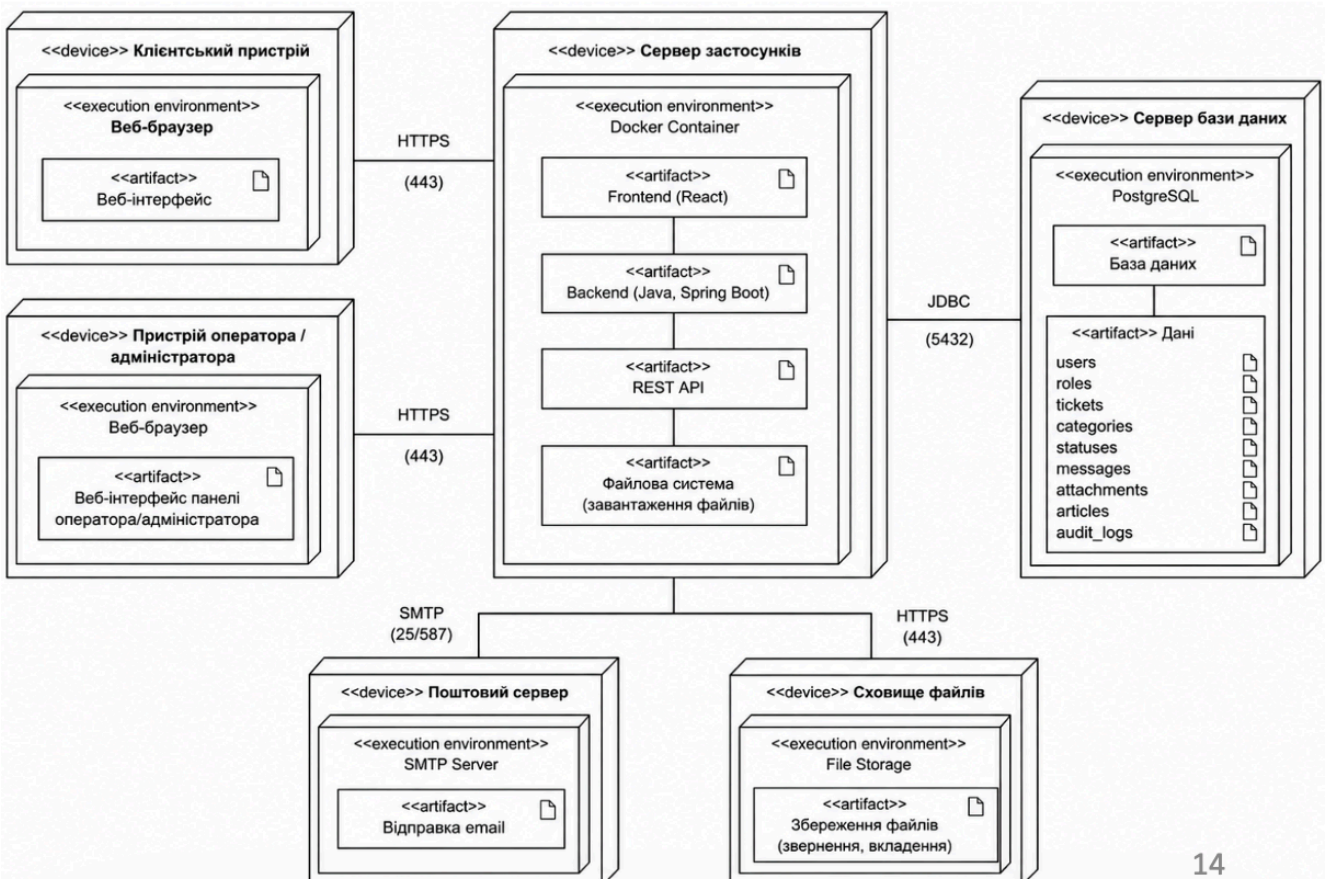


СХЕМА БАЗИ ДАНИХ



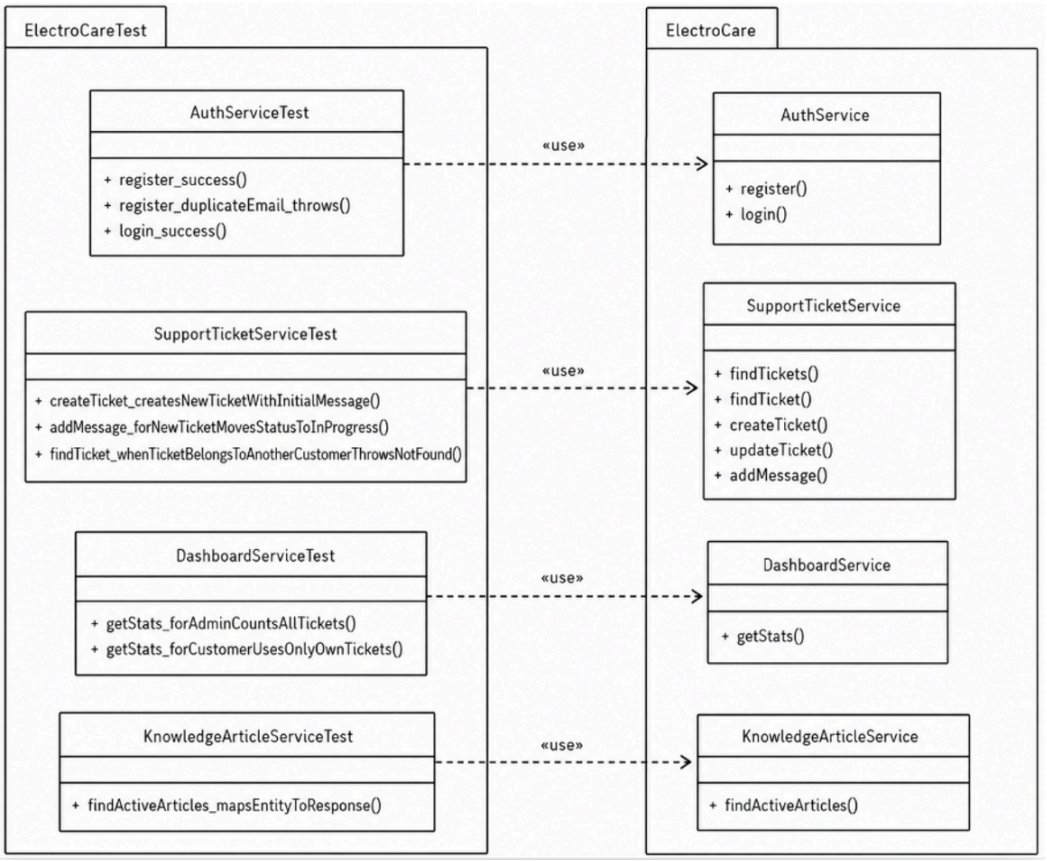
13

ДІАГРАМА РОЗГОРТАННЯ



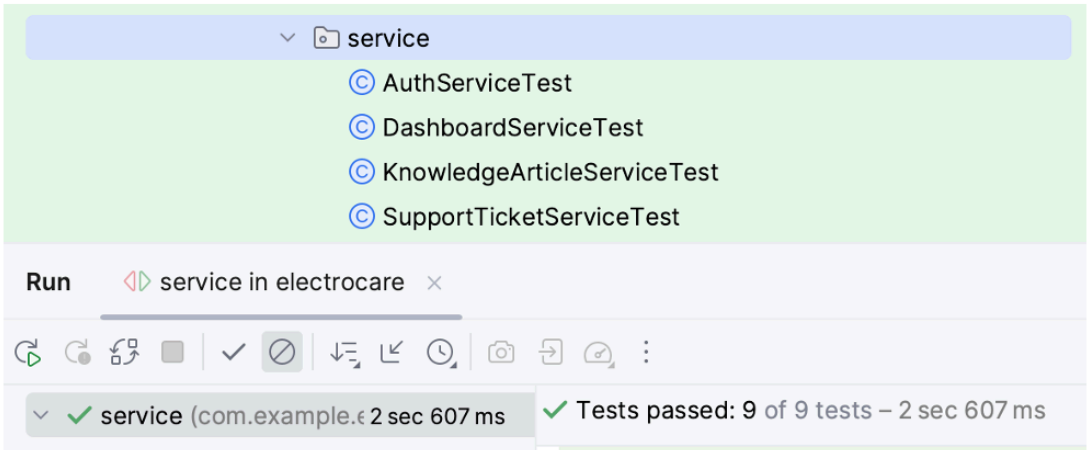
14

СХЕМА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



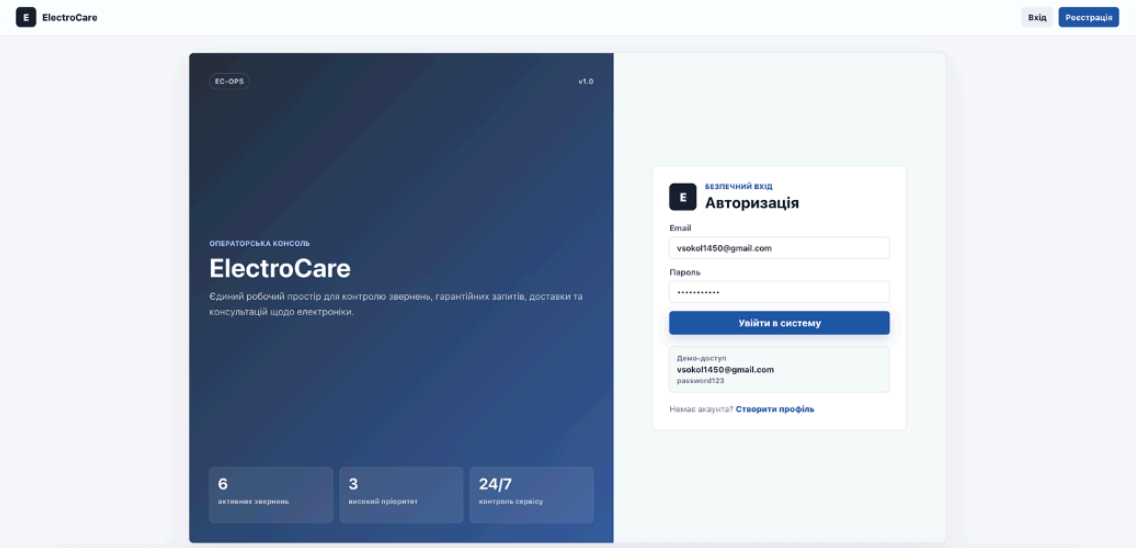
15

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



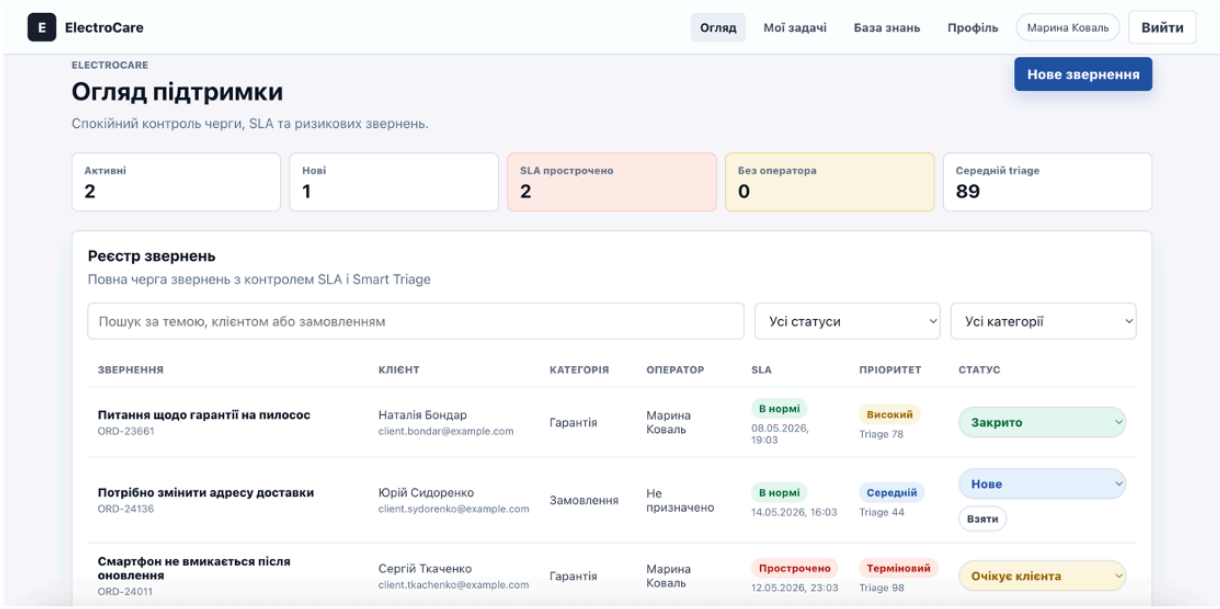
16

ЕКРАННІ ФОРМИ



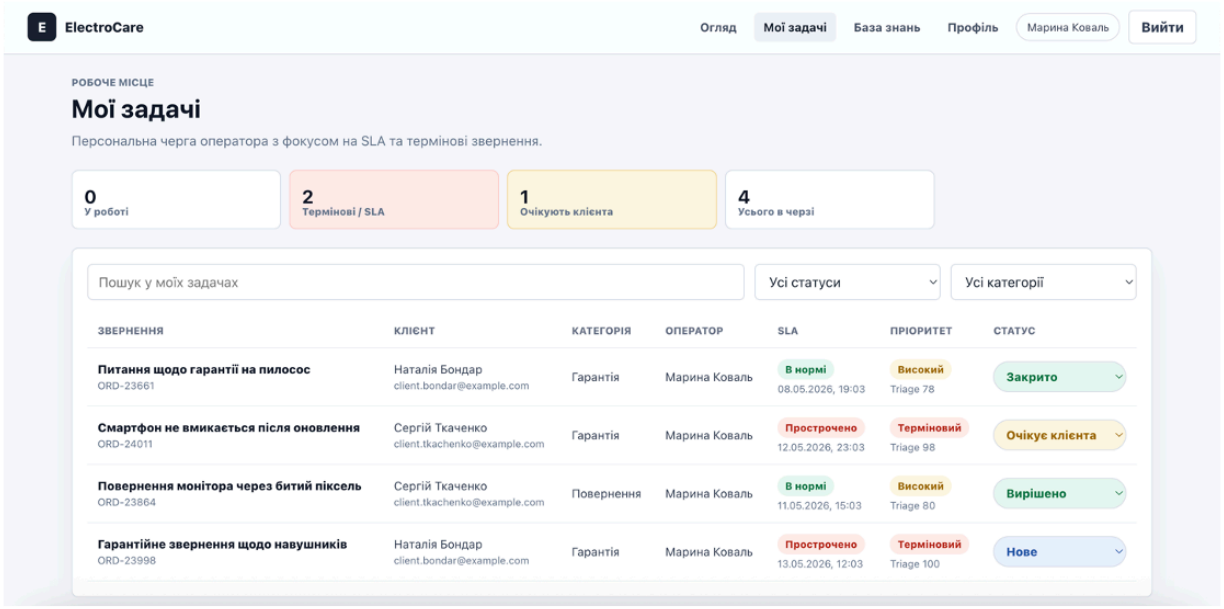
Сторінка входу

ЕКРАННІ ФОРМИ



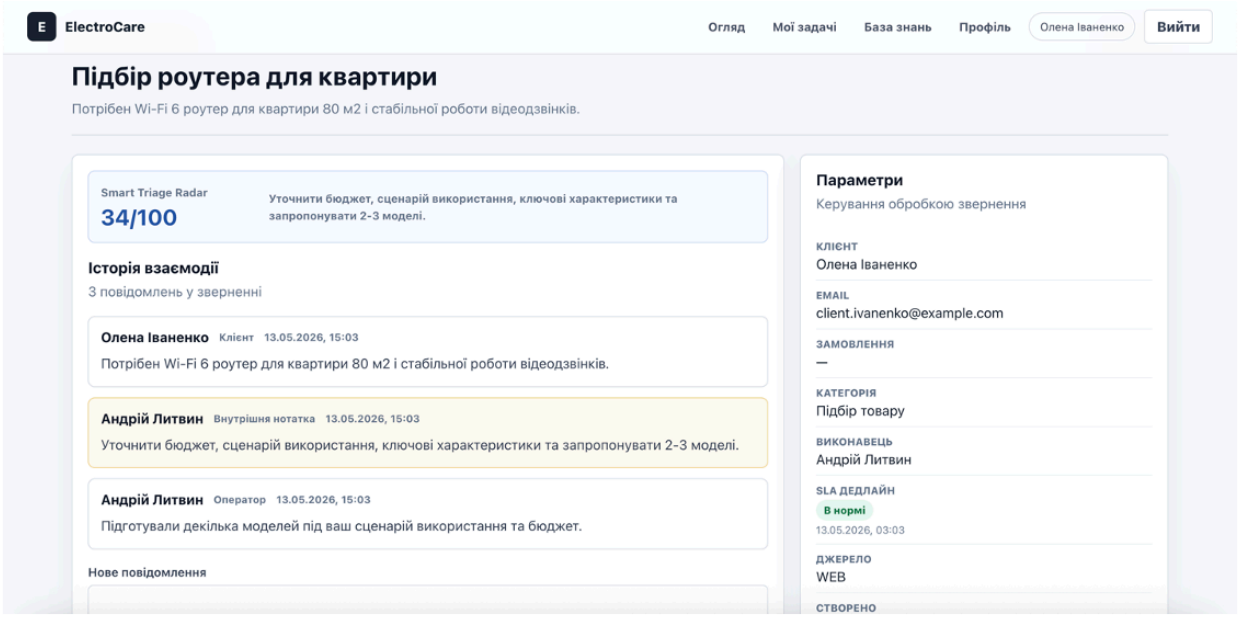
Сторінка адміністратора - Огляд

ЕКРАННІ ФОРМИ



Сторінка оператора – Мої задачі

ЕКРАННІ ФОРМИ



Сторінка деталей звернення

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Сокол В.І., Соляник Л.О. Автоматизована система підтримки із telegram-ботом і нейромережею для відповідей. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.491-493.

2. Сокол В.І., Гаманюк І.М. Актуальність системи підтримки клієнтів інтернет-магазину електроніки. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку)*.

22

ВИСНОВКИ

1. Здійснено аналіз предметної галузі підтримки клієнтів інтернет-магазину електроніки для визначення основних процесів обробки звернень, контролю статусів, SLA та взаємодії клієнта з оператором.
2. Проаналізовано програмні засоби Zendesk, Freshdesk, Intercom та Help Scout для порівняння їхніх можливостей із розробленою системою та визначення функцій, які доцільно реалізувати в ElectroCare.
3. Визначено проблеми ручної обробки звернень для обґрунтування необхідності автоматизації: втрата історії повідомлень, складність контролю статусів, нерівномірне навантаження операторів і ризик прострочення SLA.
4. Спроєктовано логічну структуру, базу даних та за допомогою UML-діаграм відпрацьовано модель прецедентів, доменну модель, основні бізнес-процеси, модулі системи та зв’язки між сутностями
5. Реалізовано web-застосунок ElectroCare для централізованої роботи зі зверненнями клієнтів, автоматичного призначення операторів, контролю SLA, ведення журналу дій та використання Smart Triage Radar.
6. Проведено тестування програмного забезпечення для перевірки коректності авторизації, рольового доступу, створення й обробки звернень, роботи SLA, Smart Triage Radar та журналу дій. За результатами тестування підтверджено працездатність основних функцій системи: backend-тести виконано успішно — 9 tests, 0 failures, frontend-збірку завершено без помилок.

23

ДОДАТОК В. ЛИСТИНГИ ОСНОВНИХ МОДУЛІВ

Лістинг 1. Головний клас запуску системи ElectroCareApplication.java

```
package com.example.electrocare;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApp
lication;

@SpringBootApplication
public class ElectroCareApplication {

   public static void main(String[]. args) {

SpringApplication.run(ElectroCareApplication.class,
args);
   }
}
```

Лістинг 2. Модель звернення клієнта SupportTicket.java

```
package com.example.electrocare.entity;

import
com.example.electrocare.entity.enums.TicketCategory;
import
com.example.electrocare.entity.enums.TicketPriority;
import
com.example.electrocare.entity.enums.TicketStatus;
import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "support_tickets")
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class SupportTicket {

   @Id
   @GeneratedValue(strategy =
GenerationType.IDENTITY)
   private Long id;

   private String subject;

   @Column(columnDefinition = "TEXT")
   private String description;

   private String orderNumber;

   @Enumerated(EnumType.STRING)
   private TicketCategory category;

   @Enumerated(EnumType.STRING)
   private TicketStatus status;

   @Enumerated(EnumType.STRING)
   private TicketPriority priority;

   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "customer_id")
   private User customer;

   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "assigned_operator_id")
   private User assignedOperator;

   private String assignedTo;
   private String source;
   private LocalDateTime slaDeadline;

   private Integer triageScore;

   @Column(columnDefinition = "TEXT")
   private String recommendedAction;

   private Long recommendedArticleId;

   @Column(columnDefinition = "TEXT")
   private String resolutionSummary;

   private Integer customerSatisfaction;

   @OneToMany(mappedBy = "ticket", cascade =
CascadeType.ALL, orphanRemoval = true)
   @Builder.Default
   private List<TicketMessage> messages = new
ArrayList<>();

   @OneToMany(mappedBy = "ticket", cascade =
CascadeType.ALL, orphanRemoval = true)
   @Builder.Default
   private List<TicketActivityLog> activityLogs = new
ArrayList<>();

   private LocalDateTime createdAt;
   private LocalDateTime updatedAt;

   @PrePersist
   public void onCreate() {
      createdAt = LocalDateTime.now();
      updatedAt = LocalDateTime.now();

      if (status == null) {
         status = TicketStatus.NEW;
      }
   }
```

```
    @PreUpdate
    public void onUpdate() {
        updatedAt = LocalDateTime.now();
    }
}
```

Лістинг 3. Контролер звернень SupportTicketController.java

```
package com.example.electrocare.controller;

import com.example.electrocare.dto.request.AssignTicketRequest;
import com.example.electrocare.dto.request.TicketMessageRequest;
import com.example.electrocare.dto.request.TicketRequest;
import com.example.electrocare.dto.response.TicketResponse;
import com.example.electrocare.service.SupportTicketService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.security.Principal;
import java.util.List;

@RestController
@RequestMapping("/api/tickets")
@RequiredArgsConstructor
public class SupportTicketController {

    private final SupportTicketService ticketService;

    @GetMapping
    public List<TicketResponse> findTickets(Principal principal) {
        return ticketService.findTickets(principal);
    }

    @GetMapping("/my")
    public List<TicketResponse> findMyTasks(Principal principal) {
        return ticketService.findMyTasks(principal);
    }

    @GetMapping("/{id}")
    public TicketResponse findTicket(@PathVariable Long id, Principal principal) {
        return ticketService.findTicket(id, principal);
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public TicketResponse createTicket(@Valid @RequestBody TicketRequest request,
                                       Principal principal) {
        return ticketService.createTicket(request, principal);
    }

    @PostMapping("/{id}/messages")
    @ResponseStatus(HttpStatus.CREATED)
    public TicketResponse addMessage(@PathVariable Long id,
                                     @Valid @RequestBody TicketMessageRequest request,
                                     Principal principal) {
        return ticketService.addMessage(id, request, principal);
    }

    @PostMapping("/{id}/assign")
    public TicketResponse assignTicket(@PathVariable Long id,
                                       @Valid @RequestBody AssignTicketRequest request,
                                       Principal principal) {
        return ticketService.assignTicket(id, request.getOperatorId(), principal);
    }

    @PostMapping("/{id}/take")
    public TicketResponse takeTicket(@PathVariable Long id, Principal principal) {
        return ticketService.takeTicket(id, principal);
    }
}
```

Лістинг 4. Основна бізнес-логіка звернень SupportTicketService.java

```
@Service
@RequiredArgsConstructor
public class SupportTicketService {

    private final SupportTicketRepository ticketRepository;
    private final UserRepository userRepository;
    private final UserService userService;
    private final SmartTriageService smartTriageService;
    private final TicketMapper ticketMapper;

    @Transactional
    public TicketResponse createTicket(TicketRequest request, Principal principal) {
        User customer = userService.currentUser(principal);

        TriageSuggestion triage = smartTriageService.analyze(
                request.getSubject(),
                request.getDescription(),
                request.getCategory(),
                request.getPriority()
        );

        User manualOperator = findOperator(request.getAssignedOperatorId());
        User assignedOperator = manualOperator == null
                ? chooseBalancedOperator(request.getCategory())
                : manualOperator;
```

```
        SupportTicket ticket = SupportTicket.builder()
                .customer(customer)
                .subject(request.getSubject())
                .description(request.getDescription())
                .orderNumber(request.getOrderNumber())
                .category(request.getCategory())
                .priority(triage.priority())
                .status(TicketStatus.NEW)
                .assignedOperator(assignedOperator)
                .assignedTo(assignedOperator == null ? null :
assignedOperator.getFullName())
                .source(request.getSource())
                .slaDeadline(triage.slaDeadline())
                .triageScore(triage.score())

.recommendedAction(triage.recommendedAction())

.recommendedArticleId(triage.recommendedArticleId())
                .build();

        SupportTicket savedTicket =
ticketRepository.save(ticket);

        addSystemLog(savedTicket, customer,
"CREATED", "Клієнт створив нове звернення");

        if (assignedOperator != null) {
            addSystemLog(
                    savedTicket,
                    customer,
                    manualOperator == null ?
"AUTO_ASSIGNED" : "ASSIGNED",
                    "Звернення призначено оператору " +
assignedOperator.getFullName()
            );
        }

        return ticketMapper.toResponse(savedTicket);
    }

    @Transactional(readOnly = true)
    public List<TicketResponse> findTickets(Principal
principal) {
        User user = userService.currentUser(principal);

        List<SupportTicket> tickets = switch
(user.getRole()) {
            case ADMIN ->
ticketRepository.findAllByOrderByUpdatedAtDesc();
            case OPERATOR ->
ticketRepository.findByAssignedOperatorOrderByUpdat
edAtDesc(user);
            case CUSTOMER ->
ticketRepository.findByCustomerOrderByUpdatedAtDes
c(user);
        };

        return tickets.stream()
                .map(ticketMapper::toResponse)
                .toList();
    }

    @Transactional(readOnly = true)
    public List<TicketResponse> findMyTasks(Principal
principal) {
        User user = userService.currentUser(principal);

        List<SupportTicket> tickets = user.getRole() ==
UserRole.OPERATOR
                ?
ticketRepository.findByAssignedOperatorOrderByUpdat
edAtDesc(user)
                :
ticketRepository.findByCustomerOrderByUpdatedAtDes
c(user);

        return tickets.stream()
                .map(ticketMapper::toResponse)
                .toList();
    }

    @Transactional
    public TicketResponse assignTicket(Long ticketId,
Long operatorId, Principal principal) {
        User admin = userService.currentUser(principal);

        if (admin.getRole() != UserRole.ADMIN) {
            throw new AccessDeniedException("Тільки
адміністратор може вручну призначати операторів");
        }

        SupportTicket ticket = findTicketEntity(ticketId);
        User operator = findOperator(operatorId);

        ticket.setAssignedOperator(operator);
        ticket.setAssignedTo(operator.getFullName());

        if (ticket.getStatus() == TicketStatus.NEW) {
            ticket.setStatus(TicketStatus.IN_PROGRESS);
        }

        addSystemLog(ticket, admin, "ASSIGNED",
                "Адміністратор призначив звернення
оператору " + operator.getFullName());

        return
ticketMapper.toResponse(ticketRepository.save(ticket));
    }

    private User chooseBalancedOperator(TicketCategory
category) {
        List<User> operators = userRepository

.findByRoleAndActiveTrueOrderByFullNameAsc(User
Role.OPERATOR);

        String department = departmentFor(category);

        List<User> suitableOperators = operators.stream()
                .filter(operator -> department.isBlank()
                        ||
department.equalsIgnoreCase(operator.getDepartment())
)
                .toList();
```

```
        List<User> candidates =
suitableOperators.isEmpty() ? operators :
suitableOperators;
        List<SupportTicket> tickets =
ticketRepository.findAllByOrderByUpdatedAtDesc();

        return candidates.stream()
                .min(Comparator.comparingLong(operator ->
activeWorkload(operator, tickets)))
                .orElse(null);
    }

    private long activeWorkload(User operator,
List<SupportTicket> tickets) {
        return tickets.stream()
                .filter(ticket -> ticket.getAssignedOperator() !=
null)
                .filter(ticket ->
ticket.getAssignedOperator().getId().equals(operator.getI
d()))
                .filter(ticket -> ticket.getStatus() !=
TicketStatus.RESOLVED)
                .filter(ticket -> ticket.getStatus() !=
TicketStatus.CLOSED)
                .count();
    }

    private String departmentFor(TicketCategory
category) {
        return switch (category) {
            case WARRANTY, RETURN -> "Сервіс та
гарантія";
            case DELIVERY, ORDER -> "Доставка";
            case PAYMENT -> "Оплата";
            case PRODUCT_SELECTION ->
"Консультації";
            case OTHER -> "";
        };
    }
}
```

Лістинг 5. Інтелектуальна оцінка звернення SmartTriageService.java

```
        package com.example.electrocare.service;

        import
com.example.electrocare.dto.response.TriageSuggestion;
        import
com.example.electrocare.entity.KnowledgeArticle;
        import
com.example.electrocare.entity.enums.TicketCategory;
        import
com.example.electrocare.entity.enums.TicketPriority;
        import
com.example.electrocare.repository.KnowledgeArticleR
epository;
        import lombok.RequiredArgsConstructor;
        import org.springframework.stereotype.Service;

        import java.time.LocalDateTime;
        import java.util.Locale;

@Service
@RequiredArgsConstructor
public class SmartTriageService {

    private final KnowledgeArticleRepository
articleRepository;

    public TriageSuggestion analyze(String subject,
                                    String description,
                                    TicketCategory category,
                                    TicketPriority requestedPriority) {
        String text = ((subject == null ? "" : subject) + " "
                + (description == null ? "" : description))
                .toLowerCase(Locale.ROOT);

        int score = baseScore(category);

        if (containsAny(text, "не працює", "злам",
"дефект", "гарант", "повернен")) {
            score += 20;
        }

        if (containsAny(text, "терміново", "не отримав",
"затрим", "помилка оплати")) {
            score += 15;
        }

        if (requestedPriority == TicketPriority.HIGH) {
            score += 10;
        } else if (requestedPriority ==
TicketPriority.URGENT) {
            score += 20;
        }

        score = Math.min(100, Math.max(10, score));

        TicketPriority priority = resolvePriority(score,
requestedPriority);
        LocalDateTime deadline =
LocalDateTime.now().plusHours(slaHours(priority,
category));

        Long articleId = articleRepository

.findFirstByCategoryAndActiveTrueOrderByTitleAsc(ca
tegory)
                .map(KnowledgeArticle::getId)
                .orElse(null);

        return new TriageSuggestion(
                score,
                priority,
                deadline,
                recommendedAction(category, priority),
                articleId
        );
    }

    private int baseScore(TicketCategory category) {
        return switch (category) {
            case WARRANTY, PAYMENT -> 58;
```

```
            case RETURN, DELIVERY -> 50;
            case ORDER -> 44;
            case PRODUCT_SELECTION -> 34;
            case OTHER -> 30;
        };
    }

    private TicketPriority resolvePriority(int score,
TicketPriority requestedPriority) {
        if (requestedPriority == TicketPriority.URGENT ||
score >= 82) {
            return TicketPriority.URGENT;
        }
        if (requestedPriority == TicketPriority.HIGH ||
score >= 65) {
            return TicketPriority.HIGH;
        }
        if (score >= 38) {
            return TicketPriority.MEDIUM;
        }
        return TicketPriority.LOW;
    }

    private long slaHours(TicketPriority priority,
TicketCategory category) {
        if (priority == TicketPriority.URGENT) {
            return 4;
        }
        if (priority == TicketPriority.HIGH
                || category == TicketCategory.WARRANTY
                || category == TicketCategory.PAYMENT) {
            return 8;
        }
        if (priority == TicketPriority.MEDIUM) {
            return 24;
        }
        return 48;
    }

    private String recommendedAction(TicketCategory
category, TicketPriority priority) {
        String prefix = priority == TicketPriority.URGENT
                ? "Обробити першочергово. "
                : "";

        return prefix + switch (category) {
            case WARRANTY -> "Запросити номер
замовлення, серійний номер та фото дефекту.";
            case RETURN -> "Перевірити строк
повернення, комплектацію та стан товару.";
            case DELIVERY -> "Перевірити ТТН, статус
перевізника та дату останнього оновлення.";
            case PAYMENT -> "Перевірити платіжний
шлюз та запропонувати альтернативну оплату.";
            case ORDER -> "Звірити номер замовлення,
оплату та стан комплектації.";
            case PRODUCT_SELECTION -> "Уточнити
бюджет, сценарій використання та потрібні
характеристики.";
            case OTHER -> "Уточнити деталі звернення і
визначити відповідальний відділ.";
        };
    }

    private boolean containsAny(String text, String...
tokens) {
        for (String token : tokens) {
            if (text.contains(token)) {
                return true;
            }
        }

        return false;
    }
}
```

Лістинг 6. Аналітика панелі керування DashboardService.java

```
        package com.example.electrocare.service;

        import
com.example.electrocare.dto.response.DashboardStatsRe
sponse;
        import
com.example.electrocare.dto.response.StatsItemRespons
e;
        import
com.example.electrocare.entity.SupportTicket;
        import com.example.electrocare.entity.User;
        import
com.example.electrocare.entity.enums.TicketCategory;
        import
com.example.electrocare.entity.enums.TicketPriority;
        import
com.example.electrocare.entity.enums.TicketStatus;
        import
com.example.electrocare.entity.enums.UserRole;
        import
com.example.electrocare.repository.SupportTicketRepos
itory;
        import lombok.RequiredArgsConstructor;
        import org.springframework.stereotype.Service;

        import java.security.Principal;
        import java.time.LocalDateTime;
        import java.util.Arrays;
        import java.util.List;
        import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class DashboardService {

    private final SupportTicketRepository
ticketRepository;
    private final UserService userService;

    public DashboardStatsResponse getStats(Principal
principal) {
        User user = userService.currentUser(principal);

        List<SupportTicket> tickets = user.getRole() ==
UserRole.ADMIN
                ?
```

```
ticketRepository.findAllByOrderByUpdatedAtDesc()
            : user.getRole() == UserRole.OPERATOR
            ?
ticketRepository.findByAssignedOperatorOrderByUpdat
edAtDesc(user)
            :
ticketRepository.findByCustomerOrderByUpdatedAtDes
c(user);

        LocalDateTime now = LocalDateTime.now();

        return new DashboardStatsResponse(
                tickets.size(),
                countStatus(tickets, TicketStatus.NEW),
                countStatus(tickets,
TicketStatus.IN_PROGRESS),
                countStatus(tickets,
TicketStatus.WAITING_CUSTOMER),
                countResolved(tickets),
                tickets.stream()
                        .filter(ticket -> ticket.getPriority() ==
TicketPriority.HIGH
                                || ticket.getPriority() ==
TicketPriority.URGENT)
                        .count(),
                tickets.stream()
                        .filter(ticket -> ticket.getSlaDeadline() !=
null)
                        .filter(ticket ->
ticket.getSlaDeadline().isBefore(now))
                        .filter(ticket -> ticket.getStatus() !=
TicketStatus.RESOLVED)
                        .filter(ticket -> ticket.getStatus() !=
TicketStatus.CLOSED)
                        .count(),
                tickets.stream()
                        .filter(ticket ->
ticket.getAssignedOperator() == null)
                        .filter(ticket -> ticket.getStatus() !=
TicketStatus.RESOLVED)
                        .filter(ticket -> ticket.getStatus() !=
TicketStatus.CLOSED)
                        .count(),
                tickets.stream()
                        .mapToInt(ticket ->
ticket.getTriageScore() == null
                                ? 0
                                : ticket.getTriageScore())
                        .average()
                        .orElse(0),
                Arrays.stream(TicketStatus.values())
                        .map(status -> new
StatsItemResponse(status.name(), countStatus(tickets,
status)))
                        .toList(),
                Arrays.stream(TicketCategory.values())
                        .map(category -> new
StatsItemResponse(category.name(),
countCategory(tickets, category)))
                        .toList(),
                tickets.stream()
                        .collect(Collectors.groupingBy(
                                ticket ->
ticket.getAssignedOperator() == null
                                        ? "Не призначено"
                                        :
ticket.getAssignedOperator().getFullName(),
                                Collectors.counting()
                        ))
                        .entrySet()
                        .stream()
                        .map(entry -> new
StatsItemResponse(entry.getKey(), entry.getValue()))
                        .toList()
        );
    }

    private long countStatus(List<SupportTicket> tickets,
TicketStatus status) {
        return tickets.stream()
                .filter(ticket -> ticket.getStatus() == status)
                .count();
    }

    private long countCategory(List<SupportTicket>
tickets, TicketCategory category) {
        return tickets.stream()
                .filter(ticket -> ticket.getCategory() ==
category)
                .count();
    }

    private long countResolved(List<SupportTicket>
tickets) {
        return tickets.stream()
                .filter(ticket -> ticket.getStatus() ==
TicketStatus.RESOLVED
                        || ticket.getStatus() ==
TicketStatus.CLOSED)
                .count();
    }
}
```

Лістинг 7. Ролі користувачів UserRole.java

```
        package com.example.electrocare.entity.enums;

public enum UserRole {
    ADMIN,
    OPERATOR,
    CUSTOMER
}
```