

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Веборієнтована CRM-система для інтернет-магазину одягу з аналітикою продажів, обліком клієнтів і замовлень, реалізованої на основі React та MongoDB

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»



(підпис)

Чалій Катерини Русланівни

(прізвище, ім'я, по батькові)

Науковий керівник



(підпис)

Омельчук Антон Анатолійович,
канд. тех. наук

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми



(підпис)

Ніжегородцев Владислав
Олександрович, канд. пед.
наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «12» травня 2026 № 20

Завідувач кафедри



(підпис)

Філоненко Михайло
Миколайович, канд. фіз.-мат.
наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Реєстрація № 42

27.05.2026

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту

(до захисту, на доопрацювання)

12.05.2026

(дата)



(підпис)

Результат захисту

5 (відмінно)

(оцінка)

08.06.2026

(дата)



(підпис)

Голова комісії:



(підпис)

О.С. Дюбанов

Члени комісії:



(підпис)

М.М. Філоненко



(підпис)

В.О. Ніжегородцев



(підпис)

А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		08.06.2016 Омельчук А.А. Дюбанов О.С.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		Філоненко М.М. Ніжегородцев В.О.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		Омельчук А.А.
4. Рецензент	від 0 до 5	5		Суховерхівка Л.П.
РАЗОМ БАЛІВ	від 0 до 50	48		

Оцінювання кваліфікаційної роботи науковим керівником – 47 балів.

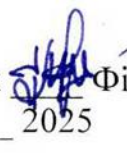
ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
95	A	5 (відмінно)	08.06.2016	Дюбанов О.С. Філоненко М.М. Ніжегородцев В.О. Омельчук А.А. Суховерхівка Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри  Філоненко М.М.
« 01 » грудня 2025

Завдання на кваліфікаційну роботу
здобувачу вищої освіти

Чалій Катерині Русланівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Веборієнтована CRM-система для інтернет-магазину одягу з аналітикою продажів, обліком клієнтів і замовлень, реалізованої на основі React та MongoDB
Затверджена наказом ректора від «28» листопада 2025 № 1526
2. Строк здачі закінченої роботи «02» травня 2026
3. Мета роботи створення веборієнтованої CRM-системи для управління клієнтами, замовленнями та аналітики продажів інтернет-магазину одягу
Об'єкт дослідження процеси управління клієнтами, замовленнями та взаємодією з користувачами в інтернет-магазині
Предмет дослідження методи та програмні засоби розробки веборієнтованих CRM-систем із використанням сучасних вебтехнологій
Орієнтовна база дослідження розроблена CRM-система із серверною частиною, клієнтським застосунком і тестовим середовищем
4. Вихідні дані до роботи сучасні вебтехнології (Node.js, Express, React, MongoDB), вимоги до функціоналу CRM-систем, принципи побудови клієнт-серверних застосунків, дані тестового середовища (crm-store-demo)
5. Перелік графічного матеріалу схеми та діаграми архітектури CRM-системи і бізнес-процесів; моделі бази даних; фрагменти програмного коду; скріншоти інтерфейсу користувача та роботи системи; таблиці порівняльного аналізу технологій і результатів тестування
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
вступ; теоретико-методологічні основи розробки CRM-систем (аналіз бізнес-процесів, огляд технологій, обґрунтування СУБД); проектування архітектури та моделей даних системи; програмна реалізація та тестування CRM-системи; висновки; список використаних джерел; додатки

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	11.09.2025	11.09.2025
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.01.2026	26.01.2026
3	II розділ (аналіз експериментальної частини про методики досліджень)	18.03.2026	18.03.2026
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	Висновки	28.04.2026	28.04.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	01.05.2026	01.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи

10. Керівник освітньо-професійної програми

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Омельчук А.А.
(прізвище, ініціали)


(підпис)

Ніжегородцев В.О.
(прізвище, ініціали)


(підпис)

Чалій К.Р.
(прізвище, ініціали)

АНОТАЦІЯ

Чалій К.Р.

Веборієнтована CRM-система для інтернет-магазину одягу з аналітикою продажів, обліком клієнтів і замовлень, реалізованої на основі React та MongoDB

Мета: розробка CRM для автоматизації бізнес-процесів інтернет-магазину.

Завдання: аналіз предметної області, проектування архітектури та реалізація модулів на стеку MERN.

Предмет: програмні засоби розробки вебсистем управління. Об'єкт: управління клієнтами та продажами в e-commerce.

База: середовище розробленої системи (backend, frontend, demo store).

За результатами сформульовано підходи до автоматизації обліку; впроваджено патерн Snapshot для цілісності історичних даних.

Результати підвищують оперативність та ефективність роботи малого бізнесу.

Кваліфікаційна робота містить 56 сторінок, 8 таблиць, 14 рисунків, список використаних джерел з 45 найменувань, 2 додатки.

Ключові слова: CRM-система, MERN-стек, React, MongoDB.

Рік виконання та захисту кваліфікаційної роботи - 2026.

ANNOTATION

Chalii K.R.

Web-oriented CRM system for an online clothing store with sales analytics, customer and order management, implemented using React and MongoDB

Purpose: development of a CRM for online store automation.

Tasks: domain analysis, architecture design, and MERN module implementation.

Subject: web management system development tools. Object: customer and sales management in e-commerce.

Base: the developed system environment (backend, frontend, demo store).

Results: automation approaches formulated; Snapshot pattern implemented for historical data integrity.

Results enhance the operational efficiency of small businesses.

The qualification work contains 56 pages, 8 tables, 14 figures, a list of references with 45 items, and 2 appendix.

Keywords: CRM system, MERN stack, React, MongoDB.

The year of completion and defense of the qualification work - 2026.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	3
ВСТУП.....	4
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ CRM-СИСТЕМ.....	6
1.1. Аналіз бізнес-процесів та вимог до управління клієнтами в e-commerce	6
1.2. Огляд сучасних технологій веброзробки та стеку MERN.....	10
1.3. Обґрунтування вибору системи управління базами даних.....	14
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЕЙ ДАНИХ СИСТЕМИ.....	18
2.1. Загальна архітектура та взаємодія компонентів CRM-платформи.....	18
2.2. Проєктування логічної структури бази даних та опис основних моделей	22
2.3. Розробка алгоритмів бізнес-логіки та механізмів контролю доступу ...	28
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ CRM-СИСТЕМИ.....	32
3.1. Розробка серверної частини та інтерфейсів програмування API	32
3.2. Реалізація клієнтської частини та логіки управління станом	36
3.3. Тестування інтеграції із зовнішніми джерелами та аналіз функціоналу	41
ВИСНОВКИ	45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	47
ДОДАТКИ	51

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface, інтерфейс програмування застосунків

Bcrypt – алгоритм хешування паролів

BSON – Binary JSON

CRM – Customer Relationship Management, система управління взаємовідносинами з клієнтами

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

MERN – MongoDB, Express, React, Node.js

RBAC – Role-Based Access Control (управління доступом на основі ролей)

REST API – Representational State Transfer Application Programming Interface

SPA – Single Page Application, односторінковий вебзастосунок

Vite – інструмент збірки фронтенд-застосунків

ВСТУП

Сектор онлайн-ритейлу одягу, що активно розвивається в соціальних мережах, характеризується високою динамікою обробки запитів та значним навантаженням на комунікаційні канали. У таких умовах ефективність роботи магазину залежить не лише від асортименту, а й від швидкості обробки звернень, точності ведення клієнтської бази та узгодженості внутрішніх процесів. Використання розрізнених інструментів, таких як електронні таблиці або месенджери, часто призводить до втрати даних, дублювання записів і помилок під час оформлення замовлень.

Існуючі CRM-рішення здебільшого орієнтовані на великі компанії або мають надлишковий функціонал, який ускладнює їх використання в малому бізнесі. У результаті виникає потреба в системах, адаптованих під конкретні бізнес-процеси інтернет-магазинів, зокрема у сфері онлайн-продажу одягу. Це обумовлює актуальність розробки спеціалізованої системи, яка об'єднує облік клієнтів, замовлень, звернень і аналітики в єдиному середовищі.

Метою кваліфікаційної роботи є створення веборієнтованої CRM-системи для інтернет-магазину одягу, яка забезпечує управління клієнтами, замовленнями та зверненнями, а також підтримує базову аналітику продажів.

Для досягнення поставленої мети визначено такі завдання:

- проаналізувати бізнес-процеси інтернет-магазину одягу та сформулювати вимоги до системи;
- обґрунтувати вибір технологічного стеку для реалізації клієнтської та серверної частин;
- спроектувати архітектуру системи з виділенням компонентів `crm-backend`, `crm-frontend` та `crm-store-demo`;
- розробити структуру бази даних і основні моделі системи;
- реалізувати функціональні модулі авторизації, роботи з клієнтами, замовленнями, зверненнями та нотатками;
- виконати тестування інтеграції із зовнішнім джерелом даних.

Об'єктом дослідження є процеси управління клієнтами, замовленнями та взаємодією з користувачами в інтернет-магазині.

Предметом дослідження є методи та програмні засоби розробки веборієнтованих CRM-систем із використанням сучасних вебтехнологій, зокрема React, Node.js, Express, MongoDB, REST API та механізмів автентифікації і контролю доступу.

У роботі застосовано методи аналізу предметної області, структурного проєктування, моделювання даних і функціонального тестування. Базою дослідження є розроблена CRM-система, яка включає серверну частину, клієнтський застосунок та тестове середовище для інтеграції із зовнішнім джерелом.

Практична значущість роботи полягає у створенні прикладного інструменту для автоматизації роботи інтернет-магазину одягу. Реалізована система забезпечує централізоване зберігання даних, обробку замовлень і звернень, а також контроль доступу до функціоналу залежно від ролі користувача. Окрему увагу приділено збереженню історичної цілісності даних, що досягається через використання snapshot-підходу в моделі замовлення.

Елементи новизни полягають у поєднанні в межах одного вебзастосунку декількох взаємопов'язаних модулів - клієнтів, замовлень, звернень, нотаток та аналітики - із реалізацією узгодженої бізнес-логіки та механізмів контролю доступу.

У межах роботи виконано проєктування архітектури системи, розробку структури бази даних, реалізацію серверної та клієнтської частин, а також тестування інтеграції із зовнішнім середовищем.

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ CRM-СИСТЕМ

1.1. Аналіз бізнес-процесів та вимог до управління клієнтами в e-commerce

Розробку CRM-системи доцільно розпочинати не з вибору технологій, а з аналізу бізнес-процесів, які вона має підтримувати. Від результатів залежать структура модулів, логіка роботи із замовленнями та спосіб взаємодії з клієнтом.

Електронна комерція давно вийшла за межі простого продажу товарів через вебсайт. На сьогодні e-commerce трансформувалася у складний цифровий простір, де паралельно виконуються пошук товарів, фінансові транзакції, логістичні операції та комунікація з покупцем. Наукові джерела її розглядають як форму організації бізнес-процесів із використанням інформаційних технологій і мережевих сервісів [7]. У межах електронної комерції виділяють різні моделі взаємодії, зокрема B2C (business-to-consumer), B2B (business-to-business) та C2C (consumer-to-consumer). Для інтернет-магазину одягу характерною є модель B2C, у якій основний акцент робиться на індивідуальній роботі з клієнтом і персоналізації обслуговування.

Центром цієї системи є клієнт. Саме тому CRM-рішення поступово стали ключовим інструментом у сфері e-commerce [11, 19]. Вони вже не обмежуються роллю «бази контактів». Сучасна CRM функціонує як інтегрована платформа, що консолідує відомості про замовлення, історію взаємодії, клієнтські звернення та внутрішні бізнес-операції. Зазначена стратегія цілком корелює з принципами клієнтоорієнтованості. Згідно з цією концепцією, ключовим фактором успіху бізнесу є не лише залучення нової аудиторії, а й утримання існуючих клієнтів через побудову довготривалих відносин [39].

Для інтернет-магазину одягу потрібно враховувати специфіку поведінки покупця. Клієнт може кілька разів переглядати асортимент, уточнювати характеристики товару, звертатися щодо наявності потрібного розміру або кольору, оформлювати замовлення, змінювати дані доставки, а потім знову повертатися до магазину для повторної покупки - усе це формує безперервний

потік інформації. Якщо такі дані не зібрані в одному середовищі, менеджер змушений працювати вручну, шукати попередню інформацію в різних джерелах і витрачати час на повторні уточнення. Таким чином зростає ймовірність помилок і погіршується якість обслуговування клієнтів.

Аналіз діяльності інтернет-магазину дає змогу структурувати основні бізнес-процеси, які повинні бути підтримані CRM-системою. У ході аналізу було виділено кілька процесів, без яких система не може працювати ефективно:

1. Робота з клієнтською базою. Йдеться не лише про збереження контактів. Значно важливіше бачити історію замовлень, попередні звернення, коментарі менеджера. У таких умовах навіть прості маркери (позначення проблемного клієнта) можуть суттєво впливати на подальшу комунікацію.

2. Обробка замовлення. Замовлення не закінчується в момент оформлення. Далі йде підтвердження, перевірка оплати, уточнення деталей, відправлення, доставка, включаючи можливі повернення товарів. Тому система має фіксувати не лише сам факт покупки, а й увесь життєвий цикл замовлення. Окремої уваги заслуговує збереження даних клієнта безпосередньо в структурі замовлення, щоб історія не змінювалась із часом.

3. Робота зі зверненнями та чатами. У сфері одягу це постійний процес. Питання щодо розміру, кольору, наявності виникають регулярно. Тому доцільно розглядати кожне звернення як окрему сутність із власною історією повідомлень, а не як випадкову переписку.

4. Аналітика та контроль доступу. Важливо не лише отримати цифри, а й обмежити доступ до них відповідно до ролей користувачів, що дозволяє розмежувати функції адміністратора, менеджера та інших учасників процесу.

Процес керування замовленнями є центральним у роботі будь-якого інтернет-магазину. Процес обробки замовлення охоплює повний життєвий цикл - від оформлення до доставки, повернення або повторної взаємодії з клієнтом. Отже, інструмент управління має забезпечувати не лише фіксацію самого факту покупки, а й супровід усього подальшого циклу [40].

Якщо такі процеси не підтримуються централізованою системою, виникає низка типових проблем. Найпоширеніші з них - дублювання даних, втрата частини історії взаємодії з клієнтом, складність контролю статусів замовлень та відсутність цілісної аналітики [11]. Наявність таких проблем свідчить про низький рівень автоматизації процесів та відсутність єдиного інформаційного середовища для роботи з клієнтськими даними. Використання CRM-системи дає змогу усунути зазначені проблеми за рахунок централізації даних і автоматизації ключових процесів.

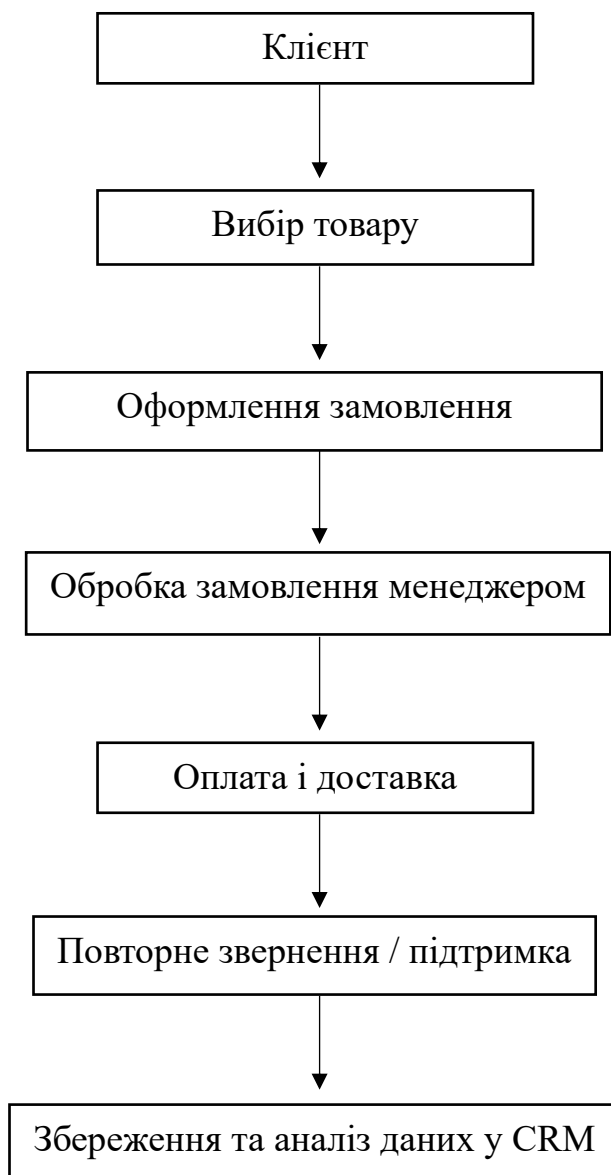


Рис. 1.1. Взаємозв'язок бізнес-процесів інтернет-магазину одягу та функцій CRM-системи

Як показано на рисунку 1.1, функціональні можливості CRM-системи безпосередньо відображають основні бізнес-процеси інтернет-магазину та формують єдиний цикл взаємодії з клієнтом.

Система охоплює повний цикл взаємодії з клієнтом - від першого контакту до накопичення та аналізу даних. Фактично, це центральний інструмент для управління продажами та комунікацією.

Основні вимоги до CRM-системи для інтернет-магазину одягу:

1. Підтримувати єдину базу клієнтів із можливістю зберігання контактних даних, приміток та історії взаємодії.
2. Забезпечувати повний цикл роботи із замовленнями - від створення до завершення, скасування або повернення.
3. Передбачити окремий механізм обробки звернень клієнтів.
4. Необхідним є аналітичний модуль, який дає змогу оцінювати продажі, активність клієнтів і поточні результати роботи.
5. Система повинна бути зручною в щоденному використанні та мати зрозумілу логіку доступу до функціоналу.
6. Відповідати нефункціональним вимогам, зокрема забезпечувати продуктивність, безпеку даних, масштабованість та надійність роботи [19, 32].

На основі цих вимог будується основа для проєктування ефективної платформи, орієнтованої на реальні потреби малого або середнього інтернет-магазину.

Отже, аналіз бізнес-процесів у сфері e-commerce показує, що сучасний інтернет-магазин потребує системи, яка об'єднує клієнтські дані, замовлення, звернення та аналітику в одному середовищі. Для магазину одягу це особливо актуально через велику кількість повторних взаємодій із клієнтом, потребу в оперативному супроводі замовлень і необхідність швидко отримувати актуальну інформацію для прийняття рішень.

1.2. Огляд сучасних технологій веброзробки та стеку MERN

Сучасна веброзробка поступово відходить від підходів, у яких клієнтська та серверна частини існували ізольовано. Натомість дедалі частіше використовуються інтегровані технологічні стеки, де обмін даними, логіка обробки та інтерфейс формують єдине цілісне середовище. Це особливо актуально для прикладних CRM-систем, орієнтованих на постійну інтерактивну взаємодію з користувачем.

Frontend-рівень сучасних вебсистем зазвичай реалізується на основі компонентного підходу, який передбачає декомпозицію інтерфейсу на незалежні частини. Одним із найпоширеніших інструментів у цьому контексті є React, який позиціонується як бібліотека для побудови користувацьких інтерфейсів [35, 36]. Його ключова ідея полягає у повторному використанні компонентів, які можуть комбінуватися між собою, формуючи складні структури сторінок. Додатково важливу роль відіграє використання віртуального DOM, який дозволяє оптимізувати оновлення інтерфейсу та мінімізувати кількість операцій із реальним DOM [35].

Інтерфейс перестає бути єдиною сторінкою, він розпадається на модулі. Таблиці, форми, модальні вікна, списки, фільтри - кожен елемент існує окремо та керується власним станом. Така модель особливо доречна для систем, де інтерфейс насичений інтерактивними елементами, а зміни відбуваються локально, без повного перезавантаження сторінки.

Класичні вебзастосунки зазвичай вимагають повного перезавантаження сторінки після кожної дії користувача. Натомість бібліотека React використовує механізм вибіркового оновлення елементів DOM, модифікуючи лише ті фрагменти інтерфейсу, які зазнали фактичних змін. Це спрощує та мінімізує навантаження на клієнтську сторону. Висока швидкість відгуку інтерфейсу стає визначальною перевагою при маніпулюванні значними масивами інформації.

Серверна частина, своєю чергою, потребує іншого підходу. Тут важлива не лише обробка запитів, а й здатність масштабуватись без значних витрат ресурсів. Node.js розглядається як середовище виконання JavaScript із неблокувальною

моделлю вводу-виводу, забезпечуючи ефективну роботу з великою кількістю одночасних запитів. Це також створює передумови для горизонтального масштабування системи при зростанні кількості користувачів [29, 30].

На відміну від традиційних серверних платформ, де кожен запит може блокувати потік виконання, Node.js використовує подієву модель. У CRM-системах це дає змогу одночасно виконувати авторизацію, запити до бази даних, аналітику та дії користувачів без помітних затримок.

Поверх Node.js зазвичай застосовується Express - мінімалістичний вебфреймворк. Це реалізує базову інфраструктуру для побудови REST API. Його гнучкість полягає в тому, що розробник самостійно формує структуру проєкту: маршрути, middleware, обробники запитів. Express не нав'язує жорсткої архітектури, завдяки чому досягається адаптивність серверної частини до специфічних потреб системи. Використання REST API забезпечує стандартизований обмін даними між клієнтською та серверною частинами системи [13].

Ще один ключовий елемент - база даних. У системах, де структура даних може змінюватися разом із розвитком бізнес-логіки, доцільно використовувати документно-орієнтовані рішення. MongoDB у цьому випадку дозволяє зберігати дані у вигляді JSON-подібних документів, полегшуючи роботу з вкладеними структурами та масивами. Використання такої моделі гарантує високу адаптивність структури даних та значно полегшує процес їх подальшої модифікації. MongoDB належить до класу NoSQL баз даних, які орієнтовані на гнучке зберігання неструктурованих або слабо структурованих даних.

Поєднання зазначених технологій формує стек MERN (MongoDB, Express, React, Node.js). Його ключова особливість полягає у використанні JavaScript на всіх рівнях системи. Використання єдиного технологічного стеку зменшує складність розробки та полегшує підтримку проєкту за рахунок уніфікації підходів [18].

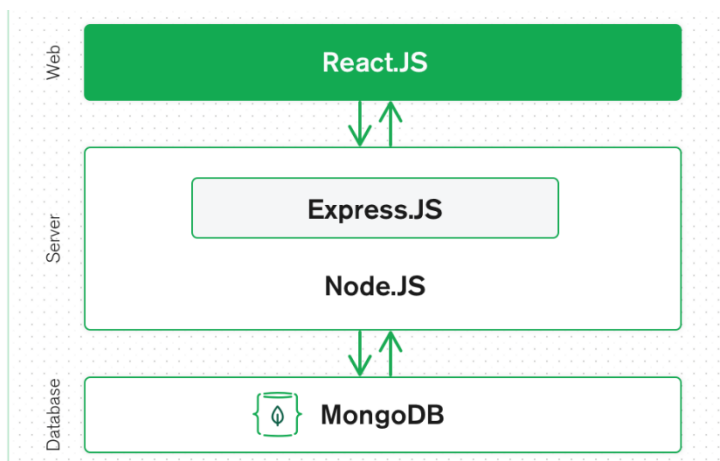


Рис. 1.2. Загальна структура взаємодії технологій стеку MERN

Розподіл функціональності між основними складовими стеку MERN доцільно розглядати як багаторівневу модель взаємодії, у якій кожен рівень виконує чітко визначену роль у загальній архітектурі системи. На рисунку 1.2 відображено узагальнену схему такої взаємодії.

Клієнтський рівень представлений застосунком на основі React, який відповідає за відображення інтерфейсу, обробку дій користувача та формування запитів до серверної частини.

Серверний рівень реалізовано з використанням Node.js та Express, які забезпечують обробку HTTP-запитів, маршрутизацію та виконання бізнес-логіки.

Рівень зберігання даних представлений базою даних MongoDB, яка відповідає за збереження інформації про клієнтів, замовлення та інші сутності системи у вигляді документів.

Кожен рівень архітектури виконує власну функцію: React відповідає за представлення даних, Node.js та Express - за їх обробку, а MongoDB - за зберігання. У такій архітектурі кожен рівень виконує окрему функцію. Це спрощує внесення змін у систему та зменшує взаємозалежність компонентів.

Така багаторівнева організація забезпечує чітке розмежування відповідальності між компонентами системи та підвищує її масштабованість і керованість.

Порівнюючи React з альтернативами, такими як Angular чи Vue, варто зазначити, що Angular передбачає більш жорстку структуру та значно більший обсяг вбудованого функціоналу. Це може бути перевагою для великих корпоративних систем, однак для CRM середнього масштабу така складність не завжди виправдана. Vue, своєю чергою, пропонує більш легкий підхід, проте має меншу екосистему у порівнянні з React.

Таблиця 1.1

Порівняльний аналіз Frontend-фреймворків

Критерій	React	Angular	Vue
Тип інструменту	Бібліотека	Повноцінний фреймворк	Прогресивний фреймворк
Навчання	Середній рівень	Високий (складна крива)	Низький (легкий старт)
Гнучкість	Дуже висока	Обмежена структурою	Висока
Екосистема	Величезна (найбільша)	Велика, корпоративна	Середня

Як видно з таблиці 1.1, для завдань даного проєкту React є найбільш раціональним вибором через його компонентну гнучкість.

Отже, використання стеку MERN дозволяє побудувати веборієнтовану CRM-систему, у якій клієнтський інтерфейс, серверна логіка та модель даних функціонують як єдине середовище. У підсумку формується архітектура, яка легко масштабується, адаптується до нових вимог і не потребує радикальних змін при розширенні функціоналу. Обраний технологічний стек також визначає підходи до організації зберігання даних, що потребує окремого обґрунтування.

1.3. Обґрунтування вибору системи управління базами даних

Одним із базових рішень під час проєктування веборієнтованої CRM-системи є вибір системи управління базами даних. Вона визначає спосіб зберігання інформації та характер взаємодії з нею на рівні прикладної логіки. Від цього залежить гнучкість моделі, швидкість адаптації до змін і складність подальшого супроводу системи.

Вибір СУБД зазвичай здійснюється з урахуванням таких критеріїв, як структура даних, вимоги до продуктивності, можливості масштабування, складність запитів та особливості інтеграції з іншими компонентами системи. Додатково важливими є такі аспекти, як підтримка транзакцій, механізми індексації, рівень консистентності даних, а також здатність системи забезпечувати високу доступність при зростанні навантаження. У розподілених системах також враховується баланс між узгодженістю, доступністю та стійкістю до відмов.

У сучасній веброзробці найчастіше використовують два базові підходи - реляційний і документний. Реляційні системи, такі як MySQL та PostgreSQL, організовують дані у вигляді таблиць із чітко визначеною схемою та зв'язками між ними. Це ефективно у випадках, коли структура даних є стабільною та наперед визначеною, а також забезпечує високу цілісність даних завдяки використанню принципів нормалізації. Однак у прикладних системах, де бізнес-логіка поступово розвивається, часто виникає потреба змінювати структуру сутностей, додавати нові поля та модифікувати зв'язки, ускладнюючи використання жорстко нормалізованої моделі. Важливою перевагою реляційних СУБД є підтримка ACID-транзакцій, які гарантують атомарність, узгодженість, ізолюваність і довговічність операцій. Це особливо критично для фінансових операцій або систем, де необхідна максимальна точність даних [28, 34].

Документно-орієнтований підхід, реалізований у MongoDB, пропонує принципово новий погляд на архітектуру даних. Замість розбиття сутності на кілька пов'язаних таблиць, інформація може зберігатися у вигляді цілісного документа з вкладеними структурами. Це особливо доречно для CRM-системи

інтернет-магазину, де замовлення містить не лише перелік товарів, а й зафіксовані на момент покупки дані клієнта, статуси обробки, службові позначки та часові характеристики. Фізично дані зберігаються у форматі BSON, який є розширенням JSON і оптимізований для ефективної обробки. На відміну від реляційних СУБД, де збереження історичної копії даних клієнта вимагало б створення складних таблиць версіонування або надмірної нормалізації, MongoDB дозволяє зберігати «зліпок» контактних даних (ім'я, телефон, адреса) безпосередньо в документі замовлення в момент його створення. Це гарантує, що історія продажів залишиться незмінною, навіть якщо клієнт оновить свій профіль у майбутньому [26, 27].

У контексті документно-орієнтованої моделі доцільно використовувати підхід денормалізації даних, за якого частина інформації свідомо дублюється з метою збереження історичної цілісності записів. Наприклад, навіть у разі зміни контактних даних клієнта у майбутньому, інформація в уже створених замовленнях залишається незмінною. У реляційній моделі реалізація такого підходу потребувала б складних механізмів версіонування або додаткових таблиць. Денормалізація може призводити до збільшення обсягу збережених даних і потребує додаткового контролю узгодженості інформації при її оновленні.

Переваги документно-орієнтованого підходу особливо проявляються при розробці CRM-системи інтернет-магазину одягу. Це пов'язано з тим, що структура даних у таких системах є складною та ієрархічною. Наприклад, сутність замовлення може містити не лише базові поля, а й масив товарів, snapshot даних клієнта, службові статуси, джерело надходження, часові мітки та допоміжні атрибути. У реляційній моделі це часто вимагало б поділу на кілька таблиць із додатковими зв'язками. У документній моделі такі дані можна зберігати природніше. Те саме стосується звернень, нотаток і платежів, де частина полів може мати різний рівень деталізації залежно від сценарію використання. Це дає змогу зменшити кількість операцій об'єднання даних (join),

які характерні для реляційних СУБД, і відповідно підвищує швидкість читання інформації [42].

Таблиця 1.2

Порівняння СУБД для розробки CRM-системи

Критерій	MySQL / PostgreSQL	MongoDB
Модель даних	Реляційна	Документна
Формат зберігання	Таблиці	JSON-подібні документи
Гнучкість зміни структури	Нижча	Вища
Робота з вкладеними даними	Менш зручна	Більш природна
Відповідність стеку MERN	Часткова	Висока

Порівняння показує, що реляційні СУБД залишаються ефективним рішенням для багатьох прикладних систем. Проте для даної роботи було обрано MongoDB, оскільки вона краще підходить для зберігання JSON-подібних структур і природніше поєднується зі стеком MERN. Це стало визначальним аргументом на користь її вибору. Крім того, MongoDB підтримує горизонтальне масштабування за допомогою механізму шардингу, дозволяючи ефективно працювати з великими обсягами даних [26, 27].

У межах розроблюваної системи використовується стек MERN, де клієнтська та серверна частини працюють із JavaScript і JSON-подібними структурами даних, через це MongoDB добре узгоджується з Node.js, Express і React не лише на рівні популярності, а й на рівні формату даних. Використання спільного формату представлення інформації на рівні клієнта і сервера усуває необхідність складних трансформацій даних, спрощує обробку запитів і робить архітектуру більш узгодженою. Це також спрощує процес розробки, оскільки

зменшується кількість необхідних перетворень даних між різними рівнями системи.

PostgreSQL і MySQL також підтримують роботу з JSON. PostgreSQL має типи `json` і `jsonb`, а також функції й оператори для обробки JSON-структур. MySQL у сучасних версіях теж підтримує JSON-дані поряд із класичною табличною моделлю. Для CRM-системи більш природнє виявилось безпосередньо документне рішення, де сутності одразу проєктуються у форматі, близькому до реального використання в застосунку.

Крім гнучкості структури, MongoDB зручна тим, що дозволяє швидко масштабувати модель даних на етапі розвитку проєкту. Під час реалізації системи функціонал може поступово ускладнюватися: додаються нові поля до клієнта, нові статуси замовлення, нові ознаки аналітики або додаткові блоки профілю. Якщо структура даних надто жорстка, кожна така зміна ускладнює розробку. У MongoDB ці зміни впроваджуються простіше. Вибрана концепція організації сховища органічно вписується в загальну архітектуру стеку MERN, де всі компоненти взаємодіють через JSON-подібні структури.

Отже, вибір СУБД має ґрунтуватися на відповідності моделі даних вимогам прикладної системи. У випадку CRM-платформи для ритейлу документна парадигма надає необхідну гнучкість, спрощує опис сутностей та пришвидшує масштабування системи. З огляду на це, впровадження MongoDB є технічно виправданим рішенням.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЕЙ ДАНИХ СИСТЕМИ

2.1. Загальна архітектура та взаємодія компонентів CRM-платформи

Фундаментом розробки CRM-платформи є її архітектурна побудова. Вона не лише описує структуру, а й визначає поведінку системи під навантаженням та умови її масштабування. Архітектурні рішення формують середовище, де клієнтська частина, серверна логіка та база даних працюють як єдиний механізм, який може бути розширений без зміни базової структури системи.

Архітектура розробленої системи відповідає клієнт-серверній моделі з елементами трирівневої організації, що включає рівень представлення (frontend), рівень прикладної логіки (backend) та рівень зберігання даних (database). Це гарантує чітке розмежування відповідальності між компонентами та підвищує керованість системи [2].

Структурно рішення базується на трьох вузлах: `crm-frontend`, `crm-backend` та `crm-store-demo`. Така декомпозиція забезпечує повну ізоляцію клієнтського інтерфейсу від серверної логіки та чітке розмежування внутрішніх процесів CRM від зовнішніх джерел даних.

Адміністративний інтерфейс `crm-frontend` функціонує як SPA на базі React із використанням інструмента збірки Vite. Це створює умови для швидкого запуску застосунку та миттєвого оновлення інтерфейсу під час розробки, підвищуючи ефективність роботи з клієнтською частиною.

Важливим архітектурним рішенням на рівні frontend є винесення мережевої взаємодії в окремий шар доступу до даних. Запити до сервера інкапсульовані в окремих модулях, які використовують спільний HTTP-адаптер для автоматичного додавання JWT-токенів. Це архітектурне рішення дозволяє чітко розмежувати логіку взаємодії з API та UI-компоненти. У результаті підтримка коду спрощується.

Серверна частина `crm-backend` побудована на базі Node.js та Express і виконує роль центрального координатора системи. Вона відповідає за обробку HTTP-запитів, реалізацію бізнес-логіки, взаємодію з базою даних MongoDB та

формування відповідей для клієнтських застосунків. Архітектура backend організована за принципом модульної маршрутизації, де окремі групи маршрутів відповідають за роботу з різними сутностями системи.

Використання асинхронної моделі обробки запитів сприяє ефективній роботі системи ефективно працювати з великою кількістю одночасних операцій. Зокрема, це дає можливість виконувати обчислення агрегованих показників, формування звітів та інші ресурсоємні операції без блокування роботи інтерфейсу [30].

Взаємодія між клієнтською та серверною частинами побудована за принципом stateless. Для автентифікації використовується механізм JWT, які передаються в заголовок Authorization. Подібний метод виключає потребу в збереженні сесійного стану на сервері, спрощуючи масштабування системи. Контроль доступу базується на рівні middleware, де перевіряються права користувача відповідно до ролей (admin, manager, analyst) [16, 17].

Окреме місце в архітектурі займає компонент crm-store-demo, який імітує зовнішній інтернет-магазин. Його призначення полягає у відтворенні реальних сценаріїв взаємодії клієнта із системою. Через цей модуль до CRM надходять замовлення та звернення, що дає змогу відстежити повний цикл обробки даних - від створення замовлення до його збереження в CRM.

Під час створення замовлення застосовується підхід денормалізації даних у вигляді збереження snapshot копії клієнта. У документ замовлення записуються основні дані (ім'я, телефон, адреса доставки), забезпечуючи збереження історичної цілісності інформації незалежно від подальших змін у профілі клієнта.

Загальна логіка взаємодії компонентів полягає в тому, що crm-frontend і crm-store-demo функціонують як незалежні клієнтські застосунки, які надсилають HTTP-запити до crm-backend. Сервер обробляє ці запити, виконує бізнес-логіку, взаємодіє з базою даних MongoDB та повертає результати у форматі JSON.

У межах системи можна виділити два основні сценарії роботи:

- внутрішній - робота користувача з CRM через crm-frontend
- зовнішній - взаємодія клієнта з demo store

Розмежування цих сценаріїв дозволяє незалежно розвивати внутрішню систему управління та зовнішні канали взаємодії.

Взаємодія між компонентами реалізована через REST API, гарантуючи стандартизований обмін даними. Використання JSON як єдиного формату представлення даних забезпечує узгодженість усіх рівнів системи та спрощує інтеграцію [13].

Для оптимізації роботи системи використано додаткові механізми. Для оновлення даних у модулі звернень застосовується періодичне опитування сервера, даючи змогу підтримувати актуальність інформації. У клієнтській частині для оптимізації пошуку використовується алгоритм затримки запитів, який зменшує навантаження на сервер.

Таблиця 2.1

Основні компоненти архітектури CRM-системи

Компонент	Технології	Призначення
crm-frontend	React, Vite	інтерфейс користувача та взаємодія з системою
crm-backend	Node.js, Express	обробка бізнес-логіки та API
crm-store-demo	React, Vite	імітація зовнішнього магазину
MongoDB	BSON	зберігання даних системи

Як показано в таблиці 2.1, кожен компонент виконує окрему функцію в системі. Такий розподіл забезпечує підвищення гнучкості архітектури та спрощує її подальший розвиток.

Запропонована архітектура враховує також нефункціональні вимоги системи. Розділення компонентів використовується для незалежного розгортання frontend і backend, підвищуючи масштабованість. Централізація бізнес-логіки на сервері підвищує безпеку, оскільки всі критичні перевірки виконуються незалежно від клієнта.

Альтернативним підходом могла б бути монолітна архітектура. Але для CRM-системи з необхідністю інтеграції із зовнішніми джерелами даних більш доцільним є використання розподіленої клієнт-серверної моделі.

Ключову роль у системі відіграє база даних MongoDB, яка виступає єдиною точкою зберігання інформації та забезпечує узгодженість даних між усіма модулями системи [24, 26].

Перевагою обраної архітектури є її розширюваність. У разі необхідності до системи можна додати нові модулі, зокрема інтеграцію з платіжними сервісами, email-сповіщення або механізми оновлення даних у реальному часі, без суттєвих змін у вже реалізованій структурі.

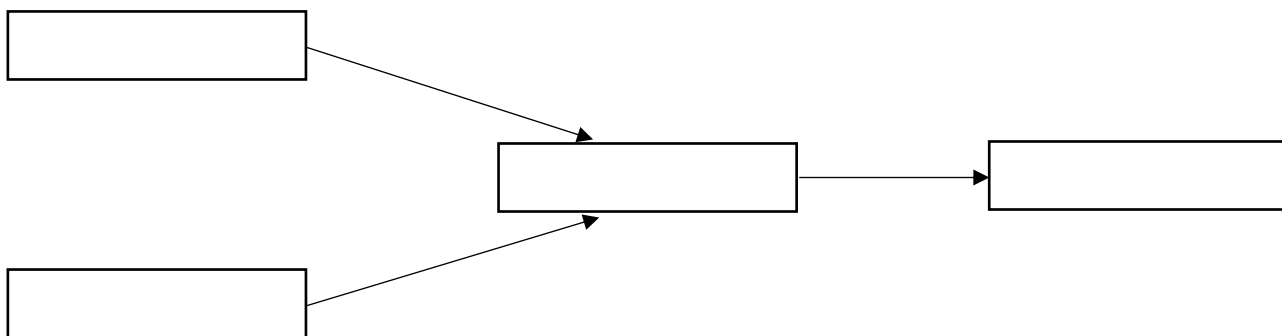


Рис. 2.1. Загальна архітектура взаємодії компонентів CRM-платформи

Як показано на рисунку 2.1, серверний рівень виступає центральним елементом системи, через який здійснюється обробка запитів від внутрішнього інтерфейсу та зовнішнього demo store.

Отже, загальна архітектура CRM-платформи побудована як взаємодія кількох незалежних компонентів і спільної бази даних, сприяючи гнучкості, масштабованості та узгодженості роботи всієї системи.

2.2. Проектування логічної структури бази даних та опис основних моделей

Формування логічної структури бази даних визначає, які саме сутності використовуються в системі, які атрибути вони містять та яким чином організована взаємодія між ними. Для CRM-платформи цей етап є критично важливим, оскільки від нього залежить узгодженість даних, швидкість доступу до інформації та можливість подальшого масштабування системи.

У межах даного проєкту для зберігання даних обрано документо-орієнтовану систему управління базами даних MongoDB. Це обумовлено гнучкістю структури документів і можливістю адаптації схеми до змін бізнес-логіки без необхідності жорсткого перетворення всієї бази. Логічна модель даних реалізована з використанням бібліотеки Mongoose, яка забезпечує опис схем, валідацію полів та інтеграцію з серверною частиною на базі Node.js [20, 21].

Структура бази даних організована у вигляді окремих колекцій, кожна з яких відповідає певній бізнес-сутності. Основу системи становлять колекції: users, clients, orders, requests, requestmessages, notes, payments і counters. Це дозволяє розмежувати операційні дані, комунікації та службову інформацію, не змішуючи їх в межах однієї структури.

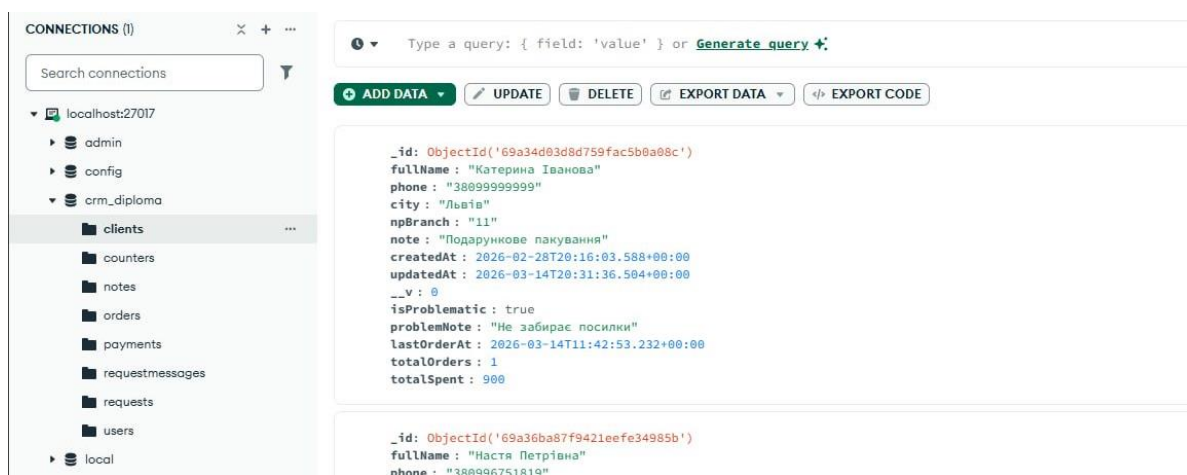


Рис. 2.2. Основні колекції бази даних CRM-системи в MongoDB

Логічна модель даних відображає ключові процеси, що відбуваються у системі:

- дані користувачів системи зберігаються у моделі User.js;

- інформація про клієнтів формується у Client.js;
- кожне замовлення фіксується у моделі Order.js;
- звернення клієнтів зберігаються у Request.js;
- окремі повідомлення діалогу - у RequestMessage.js.

Це забезпечує чітке розмежування відповідальності між сутностями та уникнути дублювання логіки.

Модель користувача (User.js) відповідає за автентифікацію та управління доступом. У ній зберігаються поля email, name та passwordHash. Паролі не зберігаються у відкритому вигляді - використовується хешування із застосуванням алгоритму bcrypt, підвищуючи безпеку системи. Застосовано параметр select: false, який виключає поле пароля з відповідей API [12]. Також для поля email використано індекси (index: true) та обмеження унікальності (unique: true), що прискорює пошук і запобігає дублюванню записів.

```

1 // src/models/User.js
2 const mongoose = require("mongoose");
3
4 const userSchema = new mongoose.Schema(
5   {
6     email: {
7       type: String,
8       required: true,
9       unique: true,
10      trim: true,
11      lowercase: true,
12    },
13
14    name: {
15      type: String,
16      trim: true,
17      default: "",
18    },
19
20    role: {
21      type: String,
22      enum: ["admin", "manager", "analyst"],
23      default: "manager",
24    },
25
26    avatarUrl: {
27      type: String,
28      trim: true,
29      default: "",
30    },
31
32    subscriptionStatus: {
33      type: String,
34      enum: ["active", "inactive", "expired"],
35      default: "inactive",
36    },
37  }
38 );
  
```

Рис. 2.3. Фрагмент моделі користувача User.js

Центральною сутністю системи є клієнт, описаний у моделі Client.js. Вона містить контактні дані (ім'я, телефон, місто, адресу доставки), службові поля та ознаки сегментації клієнта. Важливим рішенням є нормалізація номера телефону, що дозволяє уникнути дублювання записів при різних форматах введення.

У моделі також передбачено денормалізовані агреговані показники:

- totalOrders
- totalSpent
- lastOrderAt

Зберігання цих значень безпосередньо в документі клієнта дає змогу швидко відображати статистику без виконання складних агрегаційних запитів до колекції замовлень.

```

1 // src/models/Client.js
2 const mongoose = require("mongoose");
3
4 function normalizePhone(value) {
5   const digits = String(value || "").replace(/\D/g, "");
6   return digits.length ? digits : undefined;
7 }
8
9 const clientSchema = new mongoose.Schema(
10   {
11     fullName: { type: String, required: true, trim: true },
12     phone: { type: String, trim: true, default: "" },
13     // технічне поле для унікальності телефону (лише цифри)
14     phoneNormalized: { type: String },
15     city: { type: String, trim: true, default: "" },
16     npBranch: { type: String, trim: true, default: "" },
17     // примітка по доставці/поважаннях
18     note: { type: String, trim: true, default: "" },
19     // мітка "проблемний клієнт"
20     isProblematic: { type: Boolean, default: false },
21     problemNote: { type: String, trim: true, default: "" },
22     // агрегати (оновлюються з Orders)
23     totalOrders: { type: Number, default: 0, min: 0 },
24     totalSpent: { type: Number, default: 0, min: 0 },
25     lastOrderAt: { type: Date, default: null },
26   },
27   { timestamps: true }
28 );
29
30 // один телефон = один клієнт (по digits)

```

Рис. 2.4. Фрагмент моделі клієнта Client.js

Модель замовлення (Order.js) є однією з ключових у системі. Вона описує факт здійснення покупки та містить як посилання на клієнта, так і копію його даних на момент оформлення замовлення. Завдяки цьому реалізується патерн Snapshot, який забезпечує збереження історичної коректності інформації. Навіть у разі зміни даних клієнта в майбутньому, інформація у вже створених замовленнях залишається незмінною, гарантуючи стабільність фінансової звітності

У моделі також використовується механізм генерації послідовних номерів замовлень через допоміжну колекцію Counter.js. Вона дозволяє формувати зрозумілі для користувача ідентифікатори, зручні для усної комунікації та операційного обліку.

```

1  const mongoose = require("mongoose");
2
3  const ORDER_STATUSES = ["new", "confirmed", "shipped", "delivered", "canceled"];
4  const PAYMENT_STATUSES = ["unpaid", "paid"];
5
6  const orderItemSchema = new mongoose.Schema(
7    {
8      sku: { type: String, trim: true, default: "" },
9      title: { type: String, required: true, trim: true },
10     qty: { type: Number, required: true, min: 1, default: 1 },
11     price: { type: Number, required: true, min: 0, default: 0 }, // грн
12   },
13   { _id: false }
14 );
15
16 const orderSchema = new mongoose.Schema(
17   {
18     orderNo: { type: String, required: true, trim: true, unique: true },
19
20     clientId: {
21       type: mongoose.Schema.Types.ObjectId,
22       ref: "Client",
23       required: true,
24     },
25
26     // "Швидкі" поля для таблиць (снэпшот на момент створення)
27     clientName: { type: String, required: true, trim: true },
28     clientPhone: { type: String, trim: true, default: "" },
29     clientCity: { type: String, trim: true, default: "" },
30     clientNpBranch: { type: String, trim: true, default: "" },
31
32     // Товари
33     items: { type: [orderItemSchema], default: [] },
34
35     // Сума (якщо items є – буде рахуватись)
36     amount: { type: Number, required: true, min: 0 },
37   }
38 );

```

Рис. 2.5. Фрагмент моделі замовлення Order.js

Підсистема комунікацій реалізована через дві пов'язані моделі - Request.js та RequestMessage.js. Модель Request зберігає загальну інформацію про

звернення (контакт, статус, джерело, службові поля), тоді як RequestMessage відповідає за збереження окремих повідомлень у межах діалогу. Завдяки такому розподілу вдалося сформувати повноцінну структуру переписки, що є значно ефективнішим за використання одного обмеженого текстового поля.

Особливістю є впровадження денормалізованого поля unreadForManager у моделі Request. Це реалізує миттєву візуалізацію кількості нових повідомлень без потреби у ресурсомісткому аналізі всієї колекції записів.

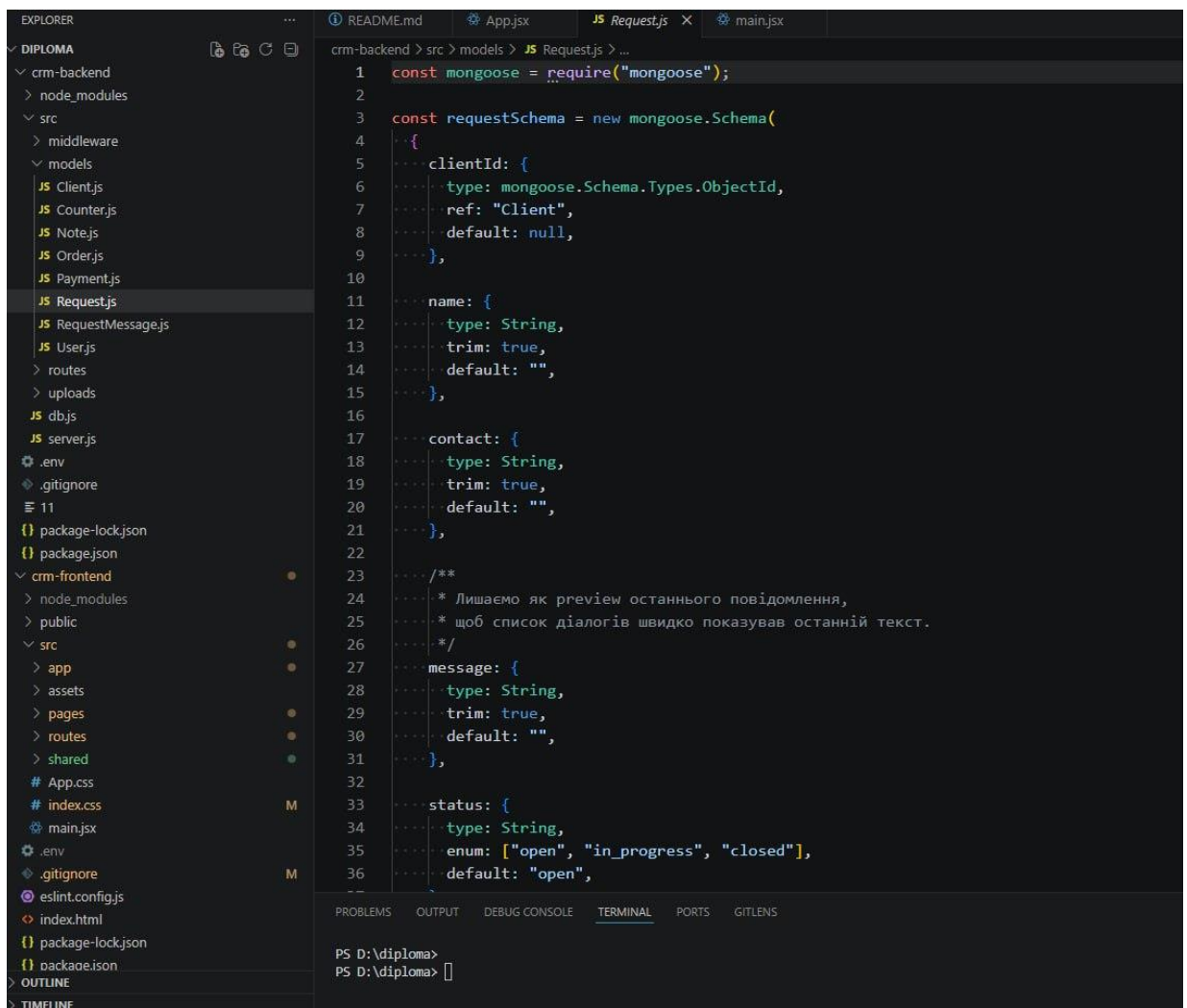


Рис. 2.6. Фрагмент моделі звернення Request.js

Додаткові моделі забезпечують допоміжну функціональність системи. Модель Note.js використовується для збереження внутрішніх нотаток менеджера та може бути прив'язана до клієнта. Модель Payment.js реалізує демо-логіку обліку платежів і містить інформацію про транзакції, статуси та строки доступу. Модель Counter.js виконує технічну роль генерації послідовних значень.

Таблиця 2.2

Основні моделі бази даних CRM-системи

Модель	Призначення	Архітектурна особливість
User.js	Управління доступом	Хешування пароля, RBAC
Client.js	Клієнтська база	Денормалізовані агрегати
Order.js	Облік замовлень	Snapshot даних клієнта
Request.js	Звернення клієнтів	Лічильник непрочитаних
RequestMessage.js	Повідомлення	Структура діалогу
Note.js	Внутрішні записи	Прив'язка до клієнта
Payment.js	Історія платежів	Демо-логіка підписки
Counter.js	Службова модель	Генерація ID

Як видно з таблиці 2.2, кожна модель виконує окрему функцію в системі та має власні архітектурні особливості, які забезпечують ефективну реалізацію бізнес-процесів CRM.

Фрагменти реалізації моделей RequestMessage.js, Note.js та Payment.js наведені у Додатку А.

Отже, логічна структура бази даних побудована як набір взаємопов'язаних, але функціонально ізольованих моделей. Використання денормалізації та патерну Snapshot забезпечує баланс між швидкістю доступу до даних і збереженням їх історичної цілісності. Обрана модель добре узгоджується з архітектурою системи та створює передумови для ефективного масштабування CRM-платформи в майбутньому.

2.3. Розробка алгоритмів бізнес-логіки та механізмів контролю доступу

Після визначення архітектури системи та побудови логічної структури бази даних наступним етапом стала розробка бізнес-логіки, яка регламентує поведінку всіх функціональних модулів системи. На цьому рівні формуються правила обробки даних, сценарії взаємодії між сутностями та обмеження, які накладаються на користувачів системи.

Основна частина бізнес-логіки реалізована на стороні серверного застосунку `crm-backend`. Саме сервер відповідає за перевірку коректності вхідних даних, виконання операцій над базою даних і дотримання встановлених правил, тоді як клієнтська ланка лише відображає результати та взаємодіє з API.

Базовим механізмом безпеки є автентифікація користувачів, яка впроваджена за допомогою Stateless-підходу на основі JSON Web Tokens. Після входу в систему сервер формує токен, що містить ідентифікаційні дані користувача, а клієнтська частина забезпечує його збереження та автоматичну передачу в заголовках HTTP-запитів. Перевірка токена виконується на сервері через `auth.middleware.js` [16, 17].

Контроль доступу побудований на моделі RBAC, яка передбачає розмежування прав залежно від ролі користувача. У системі визначено три ролі: `admin`, `manager` та `analyst`. Серверна перевірка доступу реалізована через `role.middleware.js`, тоді як на клієнті обмеження дублюються у `ProtectedRoute.jsx` та конфігурації `access.js`. Як наслідок, користувач не лише не може виконати заборонену дію, але й не бачить недоступні розділи інтерфейсу [15].

Окремим рівнем обмеження виступає контроль доступу за станом підписки. У випадку неактивного доступу користувач перенаправляється до профілю, де відображається інформація про оплату та термін дії. Така логіка інтегрована у загальну систему авторизації і враховується при обробці запитів.

Алгоритм обробки замовлень є ключовим вузлом системи. При створенні нового замовлення, зокрема через модуль `crm-store-demo`, сервер виконує послідовність дій:

- здійснюється пошук клієнта за нормалізованим номером телефону;

- за відсутності відповідного запису створюється новий клієнт;
- формується документ замовлення;
- оновлюються агреговані показники клієнта.

Такий сценарій реалізує цілісність клієнтської бази та усуває дублювання записів.

У моделі замовлення впроваджено інструментарій `snapshot`, відповідно до якого основні дані клієнта (ім'я, телефон, адреса доставки) копіюються безпосередньо в документ замовлення. Завдяки цьому ретроспективна звітність не змінюється навіть у разі редагування профілю клієнта в майбутньому, що є важливим для коректності фінансової звітності.

Для підтримки актуальних аналітичних даних використовується алгоритм асинхронного оновлення статистики клієнта. Після створення або зміни замовлення сервер оновлює поля `totalOrders`, `totalSpent` і `lastOrderAt`. Обчислення виконується безпосередньо під час операції запису, тому інтерфейс отримує вже готові агреговані значення без додаткових запитів.

Підсистема обробки звернень побудована як модель діалогу. Загальна інформація про звернення зберігається у `Request.js`, тоді як окремі повідомлення фіксуються у `RequestMessage.js`. Для оперативного відображення нових повідомлень у структурі звернення передбачено поле `unreadForManager`, яке оновлюється під час надходження нових повідомлень. Це дає змогу швидко визначати активні діалоги без додаткового аналізу всієї колекції повідомлень.

Через поле `lastSeenOrdersAt` у моделі користувача реалізовано логіку сповіщень про нові події. Система порівнює час останнього перегляду з датами створення замовлень і визначає нові записи. На клієнтському рівні оновлення виконується через HTTP polling із заданим інтервалом, дозволяючи підтримувати актуальність інформації без складних двосторонніх з'єднань.

Додатково в інтерфейсі застосовано оптимізацію пошуку клієнтів за допомогою механізму `debounce`. Запит до сервера виконується із затримкою після введення символів, що зменшує кількість звернень до API та стабілізує навантаження на систему.

Окремий модуль бізнес-логіки пов'язаний із внутрішніми нотатками. У моделі Note.js передбачено можливість створення як загальних записів, так і нотаток, прив'язаних до конкретного клієнта. Наявність статусу та пріоритету перетворює цей інструмент на елемент організації роботи менеджера.

Таблиця 2.3

Основні алгоритми та механізми бізнес-логіки

Механізм	Реалізація	Призначення
JWT-автентифікація	auth.middleware.js, bcrypt	Захист доступу до системи
RBAC	role.middleware.js, ProtectedRoute.jsx	Розмежування прав користувачів
Контроль підписки	Middleware-перевірка subscription.status	Обмеження функціоналу
Обробка замовлень	логіка backend	Синхронізація клієнтів і замовлень
Snapshot	Order.js	Збереження історичних даних
Рекалькуляція статистики	оновлення Client.js	Актуальні агреговані показники
Звернення	Request.js + RequestMessage.js	Реалізація діалогів
Сповіщення	lastSeenOrdersAt, polling	Відображення нових подій
Оптимізація пошуку	debounce	Зменшення навантаження на API

Як видно з таблиці 2.3, механізми відображають ключові аспекти функціонування системи на рівні бізнес-логіки. Вони охоплюють як базові процеси, пов'язані з автентифікацією та розмежуванням доступу, так і прикладні

алгоритми обробки даних, зокрема роботу із замовленнями, зверненнями та аналітичною інформацією.

Отже, у процесі розробки бізнес-логіки та механізмів контролю доступу було інтегровано набір правил і алгоритмів, які визначають поведінку CRM-системи в різних сценаріях використання. Реалізація автентифікації на основі JWT, розмежування прав за моделлю RBAC, а також впровадження логіки snapshot, асинхронного оновлення статистики та оптимізації запитів забезпечують узгоджену та стабільну роботу системи.

Впровадження описаних методів забезпечує підтримання цілісності даних, оперативність їх обробки та безпечний доступ до функціоналу платформи. У результаті бізнес-логіка виступає зв'язуючим елементом між архітектурою системи та її практичним використанням, перетворюючи CRM-платформу на повноцінний інструмент для управління клієнтськими процесами.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ CRM-СИСТЕМИ

3.1. Розробка серверної частини та інтерфейсів програмування API

Практичне функціонування серверного вузла CRM-платформи базується на результатах етапу архітектурного моделювання. У межах розробленої системи серверна ланка виступає центральним компонентом обробки запитів, що забезпечує стабільну обробку транзакцій, валідацію прав доступу та консистентність даних у сховищі.

Серверний рівень реалізований як окремий застосунок `crm-backend` із використанням `Node.js`, фреймворку `Express` та бази даних `MongoDB`. Така комбінація технологій відповідає архітектурі MERN-стеку та спрощує взаємодію між клієнтом і сервером [29, 30].

Структура `backend`-застосунку базується на компонентній декомпозиції і включає директорії `models`, `routes` та `middleware`, а також файли `server.js` і `db.js`.

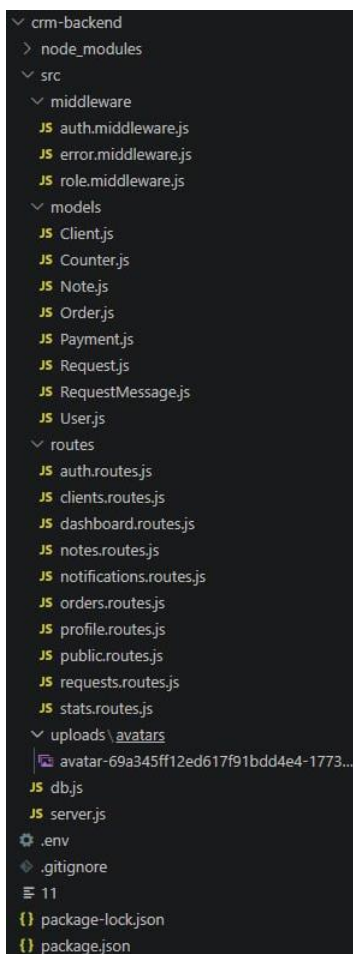


Рис. 3.1. Структура серверної частини `crm-backend`

Файл `server.js` виконує роль точки входу в застосунок. У ньому ініціалізується Express-сервер, налаштовується обробка JSON-запитів, підключається CORS, реалізуючи безпечну крос-доменну взаємодію між `crm-backend` та клієнтськими застосунками, реєструються маршрути та запускається прослуховування порту. Конфігурація сервера також передбачає віддачу статичних ресурсів, а саме - завантажених користувачами файлів [14, 15].

Підключення до бази даних винесено в окремий модуль `db.js`. Це ізолює конфігурацію з'єднання з MongoDB від решти логіки застосунку. Зміна параметрів підключення або середовища виконання не потребує втручання в бізнес-логіку або маршрути.

Функціонування серверної частини значною мірою базується на рівні `middleware`, представленому трьома модулями:

- `auth.middleware.js` - верифікує JWT-токен та ідентифікує поточного користувача.
- `role.middleware.js` - реалізує розмежування прав згідно з обраною моделлю доступу (`admin`, `manager`, `analyst`).
- `error.middleware.js` - консолідує обробку виняткових ситуацій в єдиному центрі.

API серверної частини побудоване за принципами REST, де кожен метод виконує конкретну бізнес-функцію:

- Метод GET реалізовано у маршруті `/api/stats` для динамічного формування аналітичного дашборду менеджера на основі складних агрегацій MongoDB.
- Метод POST у `public.routes.js` забезпечує приймання зовнішніх подій із `demo store` та автоматичне створення клієнтських записів.
- Метод PATCH застосовується для часткового оновлення статусів замовлень, мінімізуючи обсяг даних, які передаються мережею.

Маршрути згруповані відповідно до функціональних модулів системи. Основні з них:

- `auth.routes.js` - реєстрація, вхід, отримання поточного користувача
- `clients.routes.js` - робота з клієнтською базою
- `orders.routes.js` - управління замовленнями
- `requests.routes.js` - обробка звернень і повідомлень
- `notes.routes.js` - внутрішні нотатки
- `stats.routes.js` - аналітика та звіти

Спеціалізовані публічні маршрути (`public.routes.js`) винесені в окремий шар для взаємодії з `demo store`. Вони не потребують авторизації та призначені для приймання зовнішніх замовлень і звернень.

Таблиця 3.1

Поділ маршрутів API за типом доступу

Тип маршруту	Приклади	Призначення
Приватні	<code>/api/clients</code> , <code>/api/orders</code> , <code>/api/notes</code>	внутрішня робота CRM після авторизації
Публічні	<code>/api/public/orders</code> , <code>/api/public/requests</code>	надходження замовлень і звернень із <code>demo store</code>

Розмежування доступу на рівні маршрутів формує чітку межу між внутрішньою логікою CRM і зовнішніми джерелами даних. Публічний шар обробляє лише ті запити, які передбачені сценаріями інтеграції, без доступу до адміністративного функціоналу.

Використання формату JSON гарантує повну сумісність із JavaScript-стеком, прискорюючи маніпуляції з даними на стороні клієнта. Для оптимізації структури відповідей у моделях застосовуються перетворення `toJSON`, які видаляють службові поля та приховують чутливі дані, зокрема `passwordHash`.

Модуль авторизації реалізує Stateless-підхід із використанням JWT. Після перевірки облікових даних сервер формує токен, який передається клієнту та використовується для подальших запитів. Перевірка токена здійснюється на кожному запиті до захищених маршрутів через `auth.middleware.js`, гарантуючи

надійне блокування несанкціонованих запитів та забезпечуючи перевірку доступу до захищених ресурсів [16, 32].

Додатково на рівні серверної логіки застосовано:

- валідацію вхідних даних перед записом у базу
- контроль цілісності зв'язків між сутностями
- обробку помилкових сценаріїв (некоректні дані, відсутність ресурсу,

помилки БД)

Централізована обробка помилок через `error.middleware.js` формує уніфіковану структуру відповіді сервера та спрощує відлагодження системи.

Побудова API орієнтована на прикладні сценарії використання. Наприклад:

- створення замовлення супроводжується перевіркою клієнта та оновленням його статистики

- робота зі зверненнями включає отримання списку діалогів і історії повідомлень

- аналітичні маршрути повертають уже агреговані дані для dashboard

Реалізована серверна частина має модульну структуру, де кожен компонент відповідає за окремий етап обробки запиту. Розподіл на `middleware`, маршрути та моделі формує зрозумілу логіку обробки запитів, а REST-підхід спрощує інтеграцію з клієнтською частиною та зовнішніми джерелами.

Отже, backend CRM-системи реалізовано як окремий модульний застосунок, що дозволяє виконання бізнес-логіки, перевірку доступу та обробку даних. Архітектурні рішення, використані на цьому рівні, формують основу для стабільної роботи платформи та її подальшого розширення.

3.2. Реалізація клієнтської частини та логіки управління станом

Клієнтську частину CRM-системи реалізовано як окремий застосунок `crm-frontend` на базі бібліотеки `React` у форматі `SPA`. Це забезпечує безперервну взаємодію користувача з системою без перезавантаження сторінок і дозволяє швидко реагувати на зміни даних [36].

Під час розробки основна увага приділялася зручності роботи з ключовими модулями системи: авторизацією, клієнтами, замовленнями, зверненнями, нотатками та аналітикою. Інтерфейс побудований на компонентній основі, спрощуючи масштабування та повторне використання окремих елементів.

Frontend-проект створено з використанням `Vite`, який використовується для швидкого запуску середовища розробки та підтримки сучасних можливостей `JavaScript` через `ES-модулі`. Структура застосунку організована як набір окремих модулів - сторінок, маршрутів і службових компонентів, що спрощує підтримку коду та його подальший розвиток [44].

Маршрутизація реалізована через компонент `AppRouter.jsx` із використанням `react-router-dom`. У ньому визначено переходи між основними сторінками: `вхід`, `dashboard`, `клієнти`, `замовлення`, `звернення`, `нотатки`, `звіти` та `профіль`. Для обмеження доступу до приватних розділів використовується компонент `ProtectedRoute.jsx`, який перевіряє наявність `JWT-токена` та права користувача перед відображенням сторінки [6, 38].

Керування станом авторизації зосереджене в компоненті `AuthProvider.jsx`. Він відповідає за:

- збереження `JWT-токена`
- відновлення сесії після перезавантаження сторінки
- отримання даних поточного користувача
- надання доступу до цих даних іншим компонентам

Такий підхід прибирає дублювання логіки і спрощує підтримку коду.

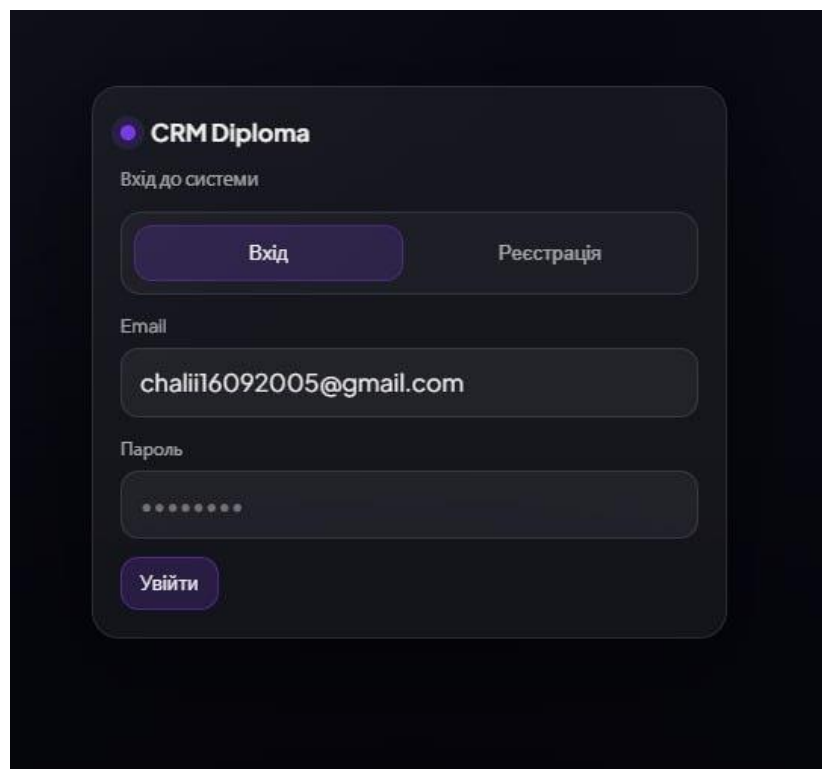


Рис. 3.2. Інтерфейс сторінки входу до CRM-системи

Після входу користувач переходить до головної панелі керування. DashboardPage.jsx виконує функцію узагальнення стану системи. На цій сторінці відображаються ключові показники: кількість клієнтів, замовлень, сумарний дохід, нові звернення та інші агреговані метрики. Дані отримуються через API та оновлюються відповідно до обраного періоду.

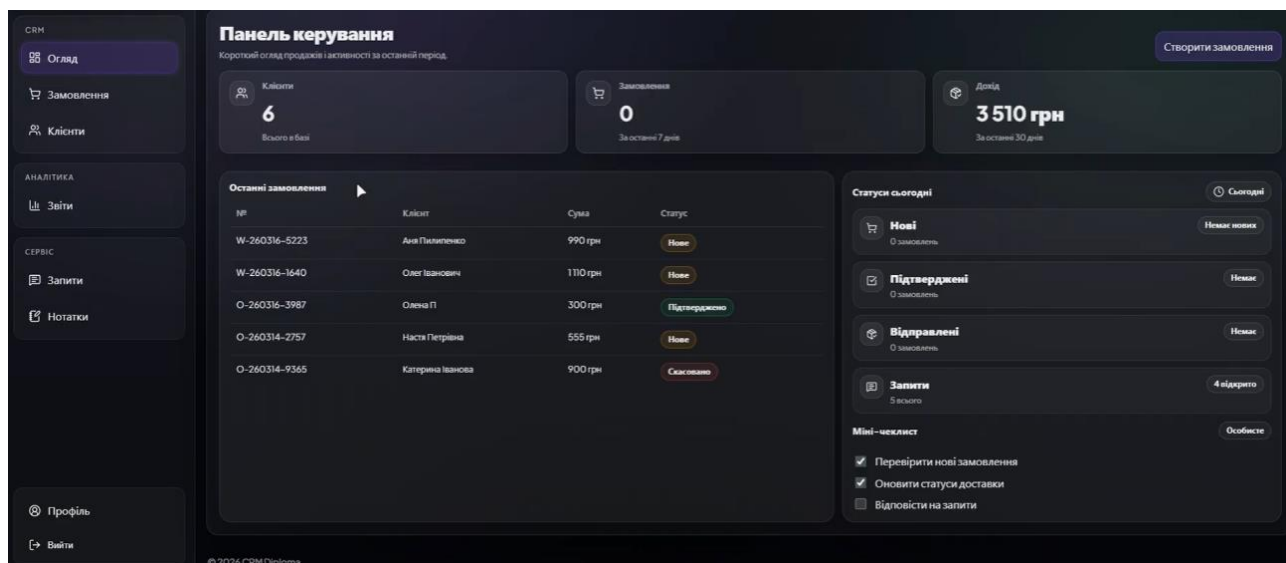


Рис. 3.3. Головна панель керування CRM-системи

Кожен функціональний модуль реалізовано у вигляді окремої сторінки:

- ClientsPage.jsx - робота з клієнтською базою
- OrdersPage.jsx - управління замовленнями
- RequestsPage.jsx - обробка звернень
- NotesPage.jsx - внутрішні нотатки
- ReportsPage.jsx - аналітика
- ProfilePage.jsx - налаштування користувача

Модуль клієнтів є одним із ключових у системі. Він включає відображення списку, пошук за параметрами, редагування записів і роботу з додатковими атрибутами (примітки, статуси, позначка проблемного клієнта). Взаємодія з даними відбувається асинхронно через API, із оновленням інтерфейсу без перезавантаження сторінки.

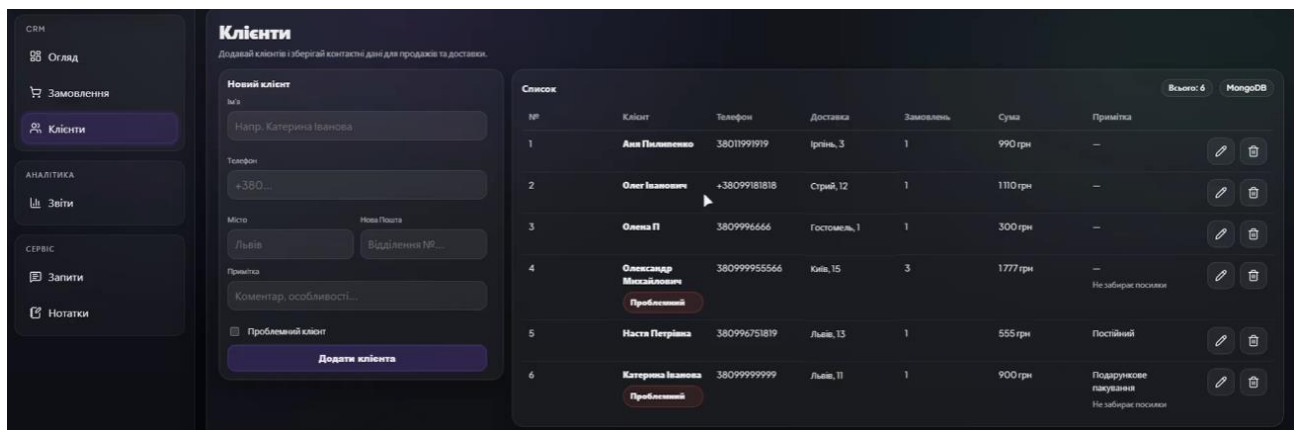


Рис. 3.4. Інтерфейс модуля роботи з клієнтами

Використання проміжного шару доступу до даних (repo layer) є важливою частиною клієнтської архітектури. У директорії shared/data зосереджено незалежні модулі для взаємодії з API:

- clientsRepo
- ordersRepo
- requestsRepo
- statsRepo
- notesRepo
- profileRepo

Ці модулі беруть на себе HTTP-запити та повертають готові дані для компонентів. Завдяки цьому UI-рівень не містить безпосередніх викликів `fetch` або `axios`, полегшуючи читання коду та зменшує залежність від конкретної реалізації API.

У клієнтській частині активно використовуються повторно застосовувані компоненти. До них належать:

- модальні вікна (див. Додаток Б)
- форми введення даних
- таблиці та списки
- елементи навігації
- блок сповіщень

Повторне використання компонентів гарантує єдність інтерфейсу та зменшує обсяг дубльованого коду.

Ключовим аспектом реалізації стала динамічна адаптація інтерфейсу. Відображення сторінок і елементів залежить від ролі користувача та стану підписки. Частина перевірок виконується на рівні маршрутизації, інша - безпосередньо в компонентах. Таким чином, інтерфейс відображає лише ті можливості, які доступні в поточному контексті.

Таблиця 3.2

Основні сторінки клієнтської частини CRM-системи

Сторінка	Призначення
LoginPage.jsx	авторизація користувача
DashboardPage.jsx	огляд основних показників системи
ClientsPage.jsx	робота з клієнтською базою
OrdersPage.jsx	робота із замовленнями
RequestsPage.jsx	робота зі зверненнями
NotesPage.jsx	робота з нотатками
ReportsPage.jsx	аналітика та звіти
ProfilePage.jsx	профіль, роль, безпека, billing

Як видно з таблиці 3.2, структура клієнтської частини відображає модульну організацію інтерфейсу, де кожна сторінка відповідає окремому бізнес-процесу. Взаємодія з даними реалізована через API із використанням проміжного шару, що відокремлює UI від деталей мережових запитів. Керування станом авторизації централізоване, а маршрутизація враховує обмеження доступу.

Отже, клієнтська частина CRM-системи реалізована як самостійний React-застосунок із чіткою структурою, керуванням станом та інтеграцією з серверним API. Реалізовані рішення забезпечують узгоджену роботу інтерфейсу, підтримку основних сценаріїв взаємодії та коректне відображення даних у всіх модулях системи.

3.3. Тестування інтеграції із зовнішніми джерелами та аналіз функціоналу

На завершальному етапі було проведено тестування працездатності CRM-системи в умовах, максимально наближених до реального використання. Основна увага була зосереджена на наскрізному тестуванні (End-to-End), яке дозволяє оцінити повний цикл обробки даних: від формування запиту на стороні користувача до його відображення у внутрішніх модулях CRM.

Для реалізації такого підходу використано `crm-store-demo`, що імітує роботу інтернет-магазину та взаємодіє з `crm-backend` через публічні API-маршрути. У цьому сценарії зовнішній застосунок формує HTTP-запити (створення замовлень і звернень), які приймаються сервером, проходять валідацію, обробляються відповідною бізнес-логікою та зберігаються в базі MongoDB. Після цього дані стають доступними у відповідних розділах інтерфейсу без додаткового втручання користувача.

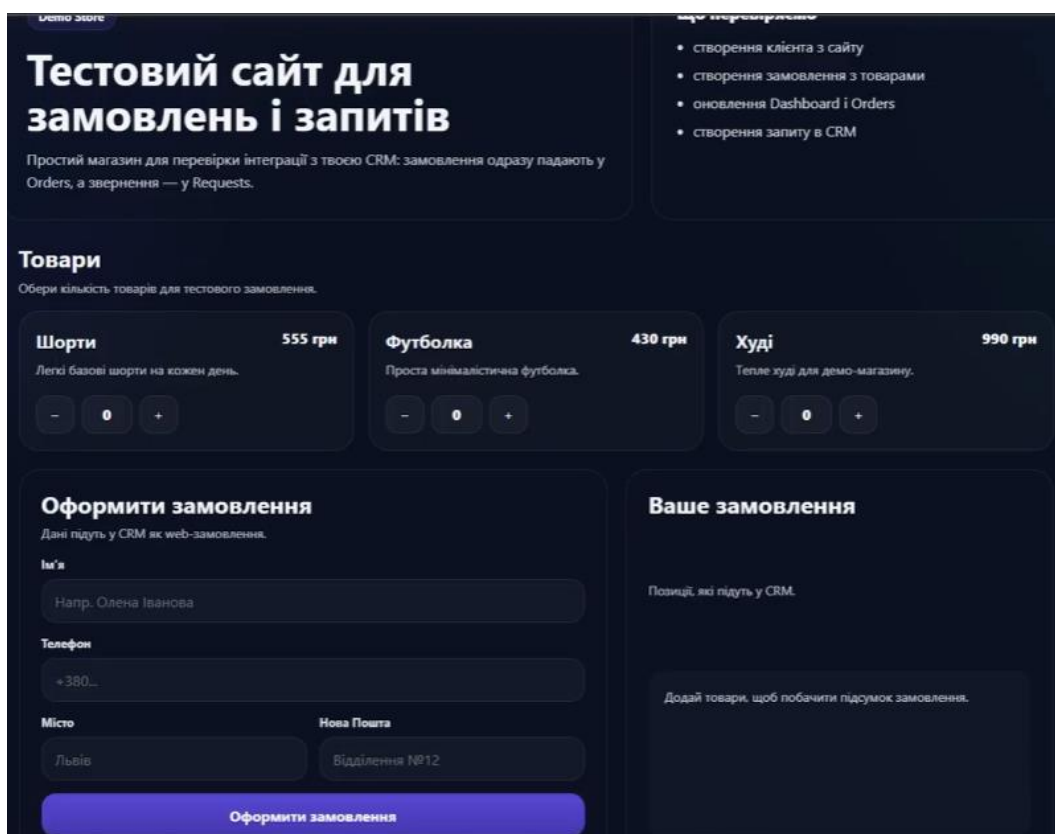


Рис. 3.5. Інтерфейс зовнішнього demo store

Під час тестування модуля замовлень було відтворено сценарій оформлення покупки через demo store. Після надсилання форми сервер виконує послідовність дій: нормалізацію номера телефону, пошук клієнта в колекції

clients, створення нового запису у разі відсутності збігу, а також формування документа замовлення в колекції orders. При цьому використовується механізм snapshot - основні дані клієнта (ім'я, телефон, адреса) копіюються безпосередньо в замовлення, забезпечуючи їх незмінність у майбутньому.

Аналіз результатів засвідчив стабільність алгоритму оновлення агрегованих показників клієнта. Після створення замовлення сервер автоматично виконує перерахунок полів totalOrders, totalSpent і lastOrderAt. Зміни одразу відображаються як у списку клієнтів, так і в аналітичних віджетах dashboard, що підтверджує узгодженість між операційними та аналітичними даними.

Товари
Обери кількість товарів для тестового замовлення.

Товар	Ціна	Кількість
Шорти Легкі базові шорти на кожен день.	555 грн	0
Футболка Проста мінімалістична футболка.	430 грн	2
Худі Тепле худі для демо-магазину.	990 грн	0

Оформити замовлення
Дані підуть у CRM як web-замовлення.

Ім'я: Анастасія Іванівна

Телефон: 38099995115

Місто: Львів

Нова Пошта: 1

Оформити замовлення

Ваше замовлення
Позиції, які підуть у CRM.

Футболка 2 × 430 грн	860 грн
Разом	860 грн

Рис. 3.6. Форма оформлення замовлення на demo store

Протестовано сценарій повторного оформлення замовлення тим самим клієнтом. У цьому випадку система використовує вже існуючий запис без створення дубліката. Це досягається за рахунок попередньої нормалізації контактних даних. Така поведінка дозволяє зберігати цілісність клієнтської бази при активному потоці замовлень.

Підсистема звернень перевірялася через форму зворотного зв'язку demo store. Після надсилання повідомлення створюється новий запис у колекції requests і відповідний запис у requestmessages. У процесі тестування підтверджено коректну роботу логіки діалогу: фіксацію історії повідомлень,

оновлення статусу звернення та зміну поля unreadForManager. Завдяки цьому менеджер отримує актуальну інформацію про нові повідомлення без необхідності аналізу всієї історії листування.

Також перевірено механізм сповіщень про нові замовлення. Поле lastSeenOrdersAt у моделі користувача дозволяє визначити, які записи були створені після останнього перегляду. На клієнтському рівні це використовується в компоненті OrdersBell.jsx, який через polling (із заданим інтервалом) отримує нові дані та відображає їх у вигляді спливаючих сповіщень (результати інтеграції із demo store та вигляд сповіщень наведено у Додатку Б).

Таблиця 3.3

Основні сценарії тестування CRM-системи

Модуль	Що перевірялося
Авторизація	JWT-вхід, відновлення сесії, доступ до приватних маршрутів
Клієнти	створення, пошук, нормалізація контактів
Замовлення	створення, snapshot, оновлення статистики
Звернення	діалогова модель, статуси, непрочитані повідомлення
Нотатки	створення, редагування, прив'язка до клієнта
Звіти	відображення агрегованих показників
Інтеграція	повний цикл обробки даних із demo store

Як видно з таблиці 3.3, тестування охоплює як внутрішні модулі CRM-системи, так і процеси взаємодії із зовнішнім джерелом даних. Для кожного функціонального блоку перевірялися ключові сценарії використання, зокрема коректність обробки запитів, узгодженість між моделями та відсутність дублювання даних. Окрему увагу приділено перевірці наскрізних процесів, у межах яких одна дія користувача впливає одразу на кілька підсистем.

Перевірка підтвердила узгоджену роботу основних вузлів системи. Операції, виконані через зовнішній інтерфейс, коректно відображаються у внутрішніх компонентах CRM. Наприклад, створене замовлення впливає не лише на список замовлень, а й на статистику клієнта, dashboard та механізм сповіщень. Аналогічно звернення синхронізуються між модулем діалогів і загальним навантаженням менеджера.

Було визначено поточні обмеження реалізації: оновлення звернень і сповіщень здійснюється через HTTP polling. Це зумовлює періодичне зростання навантаження на серверні ресурси, проте є виправданим для архітектури малого масштабу через простоту реалізації порівняно з WebSockets. Платіжний модуль має демонстраційний характер і не інтегрований із зовнішніми платіжними системами. Крім того, crm-store-demo використовується як тестове середовище і не охоплює повний функціонал реального e-commerce рішення.

Отже, проведене тестування підтвердило працездатність CRM-системи як у внутрішніх сценаріях, так і при взаємодії із зовнішнім джерелом даних. Обробка інформації відбувається коректно, без дублювання записів і втрати зв'язків між сутностями. Реалізований функціонал може розглядатися як стабільна основа для подальшого розвитку системи, зокрема в напрямках оптимізації обміну даними та розширення інтеграцій.

ВИСНОВКИ

У роботі реалізовано розробку веборієнтованої CRM-системи для автоматизації бізнес-процесів інтернет-магазину одягу. На основі проведених досліджень, проектування та програмної реалізації сформульовано такі узагальнені підсумки:

1. Аналіз предметної області та бізнес-процесів підтвердив, що сучасний онлайн-ритейл у соціальних мережах потребує консолідації даних. Встановлено, що ключовими проблемами малого бізнесу є втрата історії взаємодії та помилки при ручній обробці замовлень у месенджерах. Визначено функціональні вимоги до системи: необхідність підтримки єдиної клієнтської бази, управління життєвим циклом замовлень та інтеграція модулів комунікації.

2. Обґрунтування технологічного стеку показало переваги використання архітектури MERN (MongoDB, Express, React, Node.js). Вибір NoSQL-бази даних MongoDB є технічно виправданим завдяки гнучкості JSON-подібних структур, відповідаючи специфіці ієрархічної побудови інформаційних масивів інтернет-магазину. Використання JavaScript на всіх рівнях розробки забезпечує узгодженість формату даних та спрощує інтеграцію компонентів [18, 43].

3. Проектування архітектури та моделей даних дозволило створити надійний каркас системи. Платформу декомпозовано на три незалежні ланки: серверний вузол (crm-backend), адміністративний інтерфейс (crm-frontend) та тестову вітрину (crm-store-demo). Наукова новизна та авторський внесок полягають у впровадженні патерну Snapshot у моделі Order.js. Це архітектурне рішення гарантує незмінність історичних даних клієнта в чеку навіть при подальшій корекції його профілю, визначаючи високу достовірність майбутньої фінансової звітності.

4. Програмна реалізація та контроль доступу базуються на сучасних протоколах безпеки. Впроваджено Stateless-авторизацію на основі JWT-токенів та рольову модель доступу (RBAC), розмежовуючи повноваження адміністраторів, менеджерів та аналітиків. Клієнтську частину реалізовано як

SPA на базі React із використанням Vite, мінімізуючи час відгуку інтерфейсу на дії користувача.

5. Експериментальна перевірка функціоналу через наскрізне тестування з crm-store-demo підтвердила цілісність руху даних: від оформлення заявки на сайті до її автоматичної обробки в CRM. Тестування виявило, що асинхронне оновлення статистики клієнта (поля `totalOrders`, `totalSpent`) дозволяє миттєво отримувати аналітику без навантаження на базу даних.

Практична значущість одержаних результатів полягає у створенні готового програмного інструменту для автоматизації рутинних операцій. Застосування розробленої системи в реальному магазині одягу дозволяє скоротити час обробки замовлень та зменшити кількість ручних операцій.

Рекомендації та перспективи розвитку:

- Для підвищення оперативності комунікації пропонується перехід від HTTP-polling до протоколу WebSockets, що забезпечить оновлення статусу звернень у реальному часі без зайвих запитів до сервера [41].
- Доцільно інтегрувати систему з API державних логістичних сервісів (наприклад, «Нова Пошта») для автоматичної генерації накладних безпосередньо з інтерфейсу CRM.
- Перспективним напрямком є розширення аналітичного модуля алгоритмами прогнозування попиту на основі накопичених даних про сезонні продажі.

Отже, мета роботи досягнута, а всі поставлені завдання виконані у повному обсязі. Розроблена веборієнтована система є стабільною та масштабованою платформою для цифровізації малого та середнього бізнесу в сфері e-commerce.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аспекти використання Node.js та MongoDB для створення вебзастосунків. Інформаційні технології в соціокультурній сфері. 2023. Вип. 6. URL: <http://infotech-soccult.knukim.edu.ua/article/view/283958> (дата звернення: 10.11.2025).
2. Готович В. А., Ралік І. Р. Програмне забезпечення на основі клієнт-серверної архітектури для обліку реалізації товарів в торгівлі : матеріали Всеукр. наук.-практ. конф. молодих учених та студентів, м. Тернопіль, 2021. URL: <https://elartu.tntu.edu.ua/handle/lib/49460> (дата звернення: 20.09.2025).
3. Гузеляк О., Шевчук Ю., Береженко Б. М., Боднарчук І. О. Програмна архітектура в розподілених командах гнучких проєктів. Сучасні комп'ютерні інформаційні технології : матеріали X Міжнар. наук.-практ. конф., м. Тернопіль, 19-21 трав. 2022. Тернопіль : ТНТУ, 2022. С. 110–112.
4. Задача проєктування програмної архітектури в процесах забезпечення якості : матеріали конференції. URL: <https://elartu.tntu.edu.ua/handle/lib/40339> (дата звернення: 25.09.2025).
5. Липова О. М., Люта М. В. Особливості розробки інтерфейсу веб-додатків : навч. посібник. Київ : КНУТД, 2021. URL: <https://er.knutd.edu.ua/handle/123456789/19938> (дата звернення: 05.10.2025).
6. Основи React Router : навч. посібник. Дніпро : ДНУ, 2019. URL: <http://fpm.dnu.dp.ua/2019/12/04/react-router/> (дата звернення: 05.01.2026).
7. Про електронну комерцію : Закон України від 03.09.2015 № 675-VIII. URL: <https://zakon.rada.gov.ua/go/675-19> (дата звернення: 15.09.2025).
8. Про електронні документи та електронний документообіг : Закон України від 22.05.2003 № 851-IV. URL: <https://zakon.rada.gov.ua/go/851-15> (дата звернення: 15.09.2025).
9. Про електронну ідентифікацію та електронні довірчі послуги : Закон України від 05.10.2017 № 2155-VIII. URL: <https://zakon.rada.gov.ua/go/2155-19> (дата звернення: 18.09.2025).

10. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. URL: <https://zakon.rada.gov.ua/go/2297-17> (дата звернення: 18.09.2025).
11. Юрчук Н. П. CRM-системи: особливості функціонування та аналіз українського ринку. Вісник Ужгородського національного університету. Серія: Міжнародні економічні відносини та світове господарство. 2019. Вип. 23. Ч. 2. С. 201–207.
12. Bcrypt : official documentation. URL: <https://www.npmjs.com/package/bcrypt> (дата звернення: 10.12.2025).
13. Express 4.x API Reference : official site. URL: <https://expressjs.com/en/4x/api.html> (дата звернення: 20.11.2025).
14. Express - Routing : guide. URL: <https://expressjs.com/en/guide/routing.html> (дата звернення: 22.11.2025).
15. Express - Using middleware : guide. URL: <https://expressjs.com/en/guide/using-middleware.html> (дата звернення: 25.11.2025).
16. JSON Web Token (JWT) : RFC 7519 standard. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 01.12.2025).
17. JSON Web Token Introduction : site. URL: <https://jwt.io/introduction> (дата звернення: 02.12.2025).
18. MERN Stack Explained : site. URL: <https://www.mongodb.com/resources/languages/mern-stack> (дата звернення: 28.10.2025).
19. Microsoft. What is a CRM system? : official site. URL: <https://www.microsoft.com/en-us/dynamics-365/resources/what-is-a-crm-system> (дата звернення: 15.09.2025).
20. Mongoose - Getting Started : official documentation. URL: <https://mongoosejs.com/docs/> (дата звернення: 05.11.2025).
21. Mongoose Guides : technical docs. URL: <https://mongoosejs.com/docs/guides.html> (дата звернення: 07.11.2025).
22. Mongoose Models : documentation. URL: <https://mongoosejs.com/docs/models.html> (дата звернення: 08.11.2025).

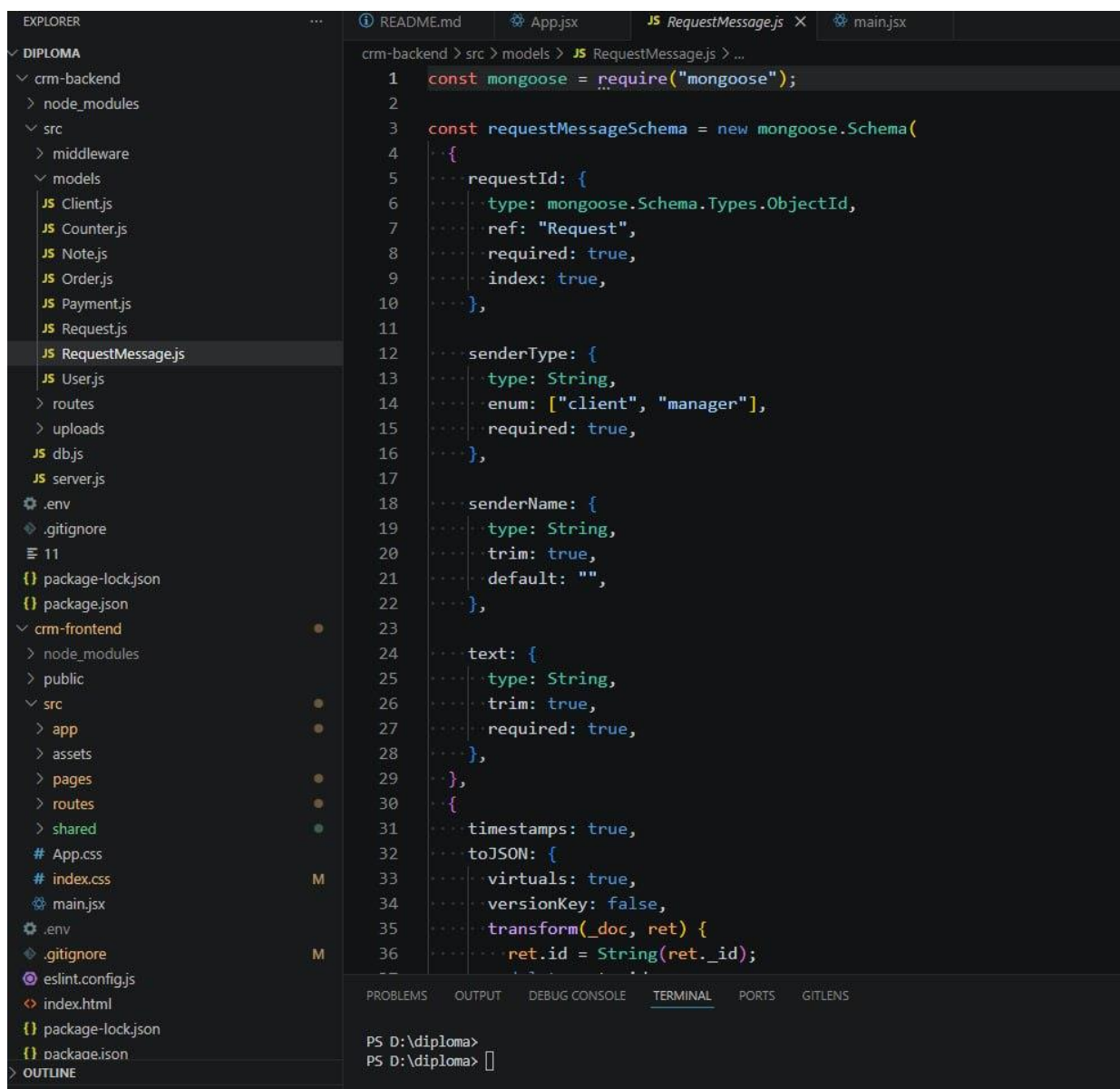
23. Mongoose Schemas : developer guide. URL: <https://mongoosejs.com/docs/guide.html> (дата звернення: 09.11.2025).
24. MongoDB Docs - Databases and Collections : technical docs. URL: <https://www.mongodb.com/docs/manual/core/databases-and-collections/> (дата звернення: 25.10.2025).
25. MongoDB Docs - Documents : technical manual. URL: <https://www.mongodb.com/docs/manual/core/document/> (дата звернення: 26.10.2025).
26. MongoDB Manual : official site. URL: <https://www.mongodb.com/docs/manual/> (дата звернення: 27.10.2025).
27. MongoDB. What is MongoDB? : introduction. URL: <https://www.mongodb.com/docs/manual/introduction/> (дата звернення: 24.10.2025).
28. MySQL Reference Manual : official site. URL: <https://dev.mysql.com/doc/refman/8.4/en/> (дата звернення: 12.10.2025).
29. Node.js - About Node.js : official site. URL: <https://nodejs.org/en/about> (дата звернення: 15.11.2025).
30. Node.js - Overview of Blocking vs Non-Blocking : guide. URL: <https://nodejs.org/en/learn/asynchronous-work/overview-of-blocking-vs-non-blocking> (дата звернення: 18.11.2025).
31. Node.js Modules : documentation. URL: <https://nodejs.org/api/modules.html> (дата звернення: 19.11.2025).
32. OWASP Top Ten : official site. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 12.12.2025).
33. PostgreSQL Documentation - JSON Functions and Operators : official site. URL: <https://www.postgresql.org/docs/current/functions-json.html> (дата звернення: 14.10.2025).
34. PostgreSQL Documentation - JSON Types : official site. URL: <https://www.postgresql.org/docs/current/datatype-json.html> (дата звернення: 15.10.2025).

35. Quick Start - React : official documentation. URL: <https://react.dev/learn> (дата звернення: 10.01.2026).
36. React - Describing the UI : developer docs. URL: <https://react.dev/learn/describing-the-ui> (дата звернення: 12.01.2026).
37. React - Managing State : developer docs. URL: <https://react.dev/learn/managing-state> (дата звернення: 14.01.2026).
38. React Router Documentation : official site. URL: <https://reactrouter.com/home> (дата звернення: 08.01.2026).
39. Salesforce. What is CRM? : official site. URL: <https://www.salesforce.com/crm/what-is-crm/> (дата звернення: 18.09.2025).
40. Shopify. What is an Order Management System? : official site. URL: <https://www.shopify.com/enterprise/blog/order-management-system-oms> (дата звернення: 22.09.2025).
41. Socket.IO Documentation : official site. URL: <https://socket.io/docs/v4/> (дата звернення: 20.01.2026).
42. Stonebraker M. Stonebraker on NoSQL. Communications of the ACM. 2011. Vol. 54, No. 8. P. 10–11. DOI: <https://doi.org/10.1145/2076256.2076257> (дата звернення: 16.10.2025).
43. Tilkov S. Node.js for Scalable Web Applications. InfoQ. 2010. URL: <https://www.infoq.com/articles/node-js-scalability/> (дата звернення: 17.11.2025).
44. Vite Guide : official site. URL: <https://vite.dev/guide/> (дата звернення: 18.01.2026).
45. Vite - Building for Production : developer docs. URL: <https://vite.dev/guide/build> (дата звернення: 22.01.2026).

ДОДАТКИ

Додаток А

Фрагмент моделі повідомлення RequestMessage.js



```
EXPLODER  ...  README.md  App.jsx  JS RequestMessage.js  main.jsx

✓ DIPLOMA
  ✓ crm-backend
    > node_modules
    ✓ src
      > middleware
      ✓ models
        JS Client.js
        JS Counter.js
        JS Note.js
        JS Order.js
        JS Payment.js
        JS Request.js
        JS RequestMessage.js
        JS User.js
      > routes
      > uploads
      JS db.js
      JS server.js
    .env
    .gitignore
    11
    {} package-lock.json
    {} package.json
  ✓ crm-frontend
    > node_modules
    > public
    ✓ src
      > app
      > assets
      > pages
      > routes
      > shared
    # App.css
    # index.css
    main.jsx
    .env
    .gitignore
    eslint.config.js
    index.html
    {} package-lock.json
    {} package.json
  > OUTLINE

crm-backend > src > models > JS RequestMessage.js > ...

1  const mongoose = require("mongoose");
2
3  const requestMessageSchema = new mongoose.Schema(
4    {
5      requestId: {
6        type: mongoose.Schema.Types.ObjectId,
7        ref: "Request",
8        required: true,
9        index: true,
10     },
11
12     senderType: {
13       type: String,
14       enum: ["client", "manager"],
15       required: true,
16     },
17
18     senderName: {
19       type: String,
20       trim: true,
21       default: "",
22     },
23
24     text: {
25       type: String,
26       trim: true,
27       required: true,
28     },
29   },
30   {
31     timestamps: true,
32     toJSON: {
33       virtuals: true,
34       versionKey: false,
35       transform(_doc, ret) {
36         ret.id = String(ret._id);
37       }
38     }
39   }
40 );
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS D:\diploma>
PS D:\diploma>

Фрагмент моделі нотатки Note.js

The image shows a Visual Studio Code editor with a project named "DIPLOMA". The Explorer sidebar on the left displays the project structure, including folders like "src", "models", and "routes". The main editor window shows the code for "Note.js" in the "models" directory. The code defines a Mongoose schema for a note, including fields for "clientId", "text", "status", "priority", "timestamps", and "toJSON". The terminal at the bottom shows the command prompt in the project directory.

```

1  const mongoose = require("mongoose");
2
3  const noteSchema = new mongoose.Schema(
4    {
5      clientId: {
6        type: mongoose.Schema.Types.ObjectId,
7        ref: "Client",
8        default: null,
9      },
10
11     text: {
12       type: String,
13       trim: true,
14       required: true,
15     },
16
17     status: {
18       type: String,
19       enum: ["active", "done"],
20       default: "active",
21     },
22
23     priority: {
24       type: String,
25       enum: ["normal", "important"],
26       default: "normal",
27     },
28   },
29   {
30     timestamps: true,
31     toJSON: {
32       virtuals: true,
33       versionKey: false,
34       transform(_doc, ret) {
35         ret.id = String(ret._id);
36         delete ret._id;
37       },
38     },
39   },
40 );

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS D:\diploma>
PS D:\diploma>

Фрагмент моделі платежу Payment.js

The screenshot shows a code editor with the following structure:

- EXPLORER (Left Panel):**
 - DIPLOMA
 - crm-backend
 - node_modules
 - src
 - middleware
 - models
 - Client.js
 - Counter.js
 - Note.js
 - Order.js
 - Payment.js** (selected)
 - Request.js
 - RequestMessage.js
 - User.js
 - routes
 - uploads
 - db.js
 - server.js
 - .env
 - .gitignore
 - 11
 - package-lock.json
 - package.json
 - crm-frontend
 - node_modules
 - public
 - src
 - app
 - assets
 - pages
 - routes
 - shared
 - App.css
 - index.css
 - main.jsx
 - .env
 - .gitignore
 - eslint.config.js
 - index.html
 - package-lock.json
 - package.json
 - OUTLINE
 - TIMELINE

- Code Editor (Main Panel):**
- Tab: JS Payment.js
- Path: crm-backend > src > models > JS Payment.js > ...
- Code:


```

1  const mongoose = require("mongoose");
2
3  const paymentSchema = new mongoose.Schema(
4    {
5      ...userId: {
6        ...type: mongoose.Schema.Types.ObjectId,
7        ...ref: "User",
8        ...required: true,
9        ...index: true,
10     ...},
11
12     ...amount: {
13       ...type: Number,
14       ...required: true,
15       ...min: 0,
16     ...},
17
18     ...currency: {
19       ...type: String,
20       ...trim: true,
21       ...default: "UAH",
22     ...},
23
24     ...periodDays: {
25       ...type: Number,
26       ...enum: [30, 90, 180],
27       ...required: true,
28     ...},
29
30     ...status: {
31       ...type: String,
32       ...enum: ["paid"],
33       ...default: "paid",
34     ...},
35
36     ...provider: {

```
- TERMINAL (Bottom Panel):**
- PS D:\diploma>
- PS D:\diploma> █

Модалне вікно редагування даних клієнта

Редагування

Онови дані клієнта та збережи зміни.

Ім'я

Олександр Михайлович

Телефон

380999955566

Місто

Київ

Нова Пошта

15

Примітка

☒ Проблемний клієнт

Причина

Не забирає посилки

Скасувати

Зберегти

Інтерфейс модуля роботи із замовленнями

Замовлення

Список замовлень з доставкою, статусом та оплатою. Дані беруться з MongoDB.

Створити замовлення

Список

Всі

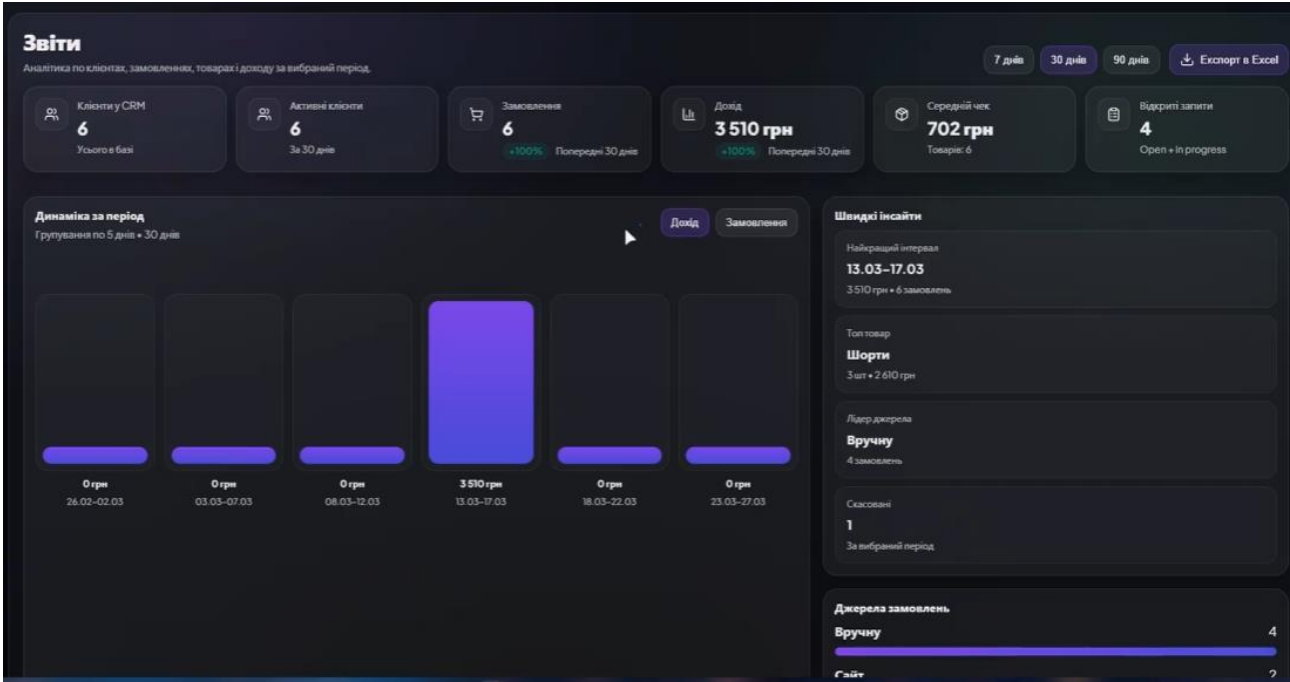
Пошук: № / клієнт / телефон

Замовлення	Клієнт	Доставка	Товари	Сума	Статус	Оплата	
W-260316-5223 16.03.26, 20:22	Аня Поліщук 38011999999	Ірпінь 3	Худяк1	990 грн	Нове	Не оплачено	 
W-260316-1640 16.03.26, 20:18	Олег Іванович +380999181818	Стрий 12	Шортик2	1110 грн	Нове	Не оплачено	 
O-260316-3987 16.03.26, 18:20	Олена П 38099966666	Гостомель 1	Носки2	300 грн	Підтверджено	Оплачено	 
O-260314-2757 14.03.26, 22:36	Настя Петрівна 380999751819	Львів 13	Шортик1	555 грн	Нове	Не оплачено	 
O-260314-9365 14.03.26, 13:42	Катерина Іванова 38099999999	Львів 11	—	900 грн	Скасовано	Не оплачено	 
O-260314-6617 14.03.26, 10:54	Олександр Михайлович 380999955566	Київ 15	—	555 грн	Підтверджено	Оплачено	 

Показано 1-6 з 6

Назад1Далі

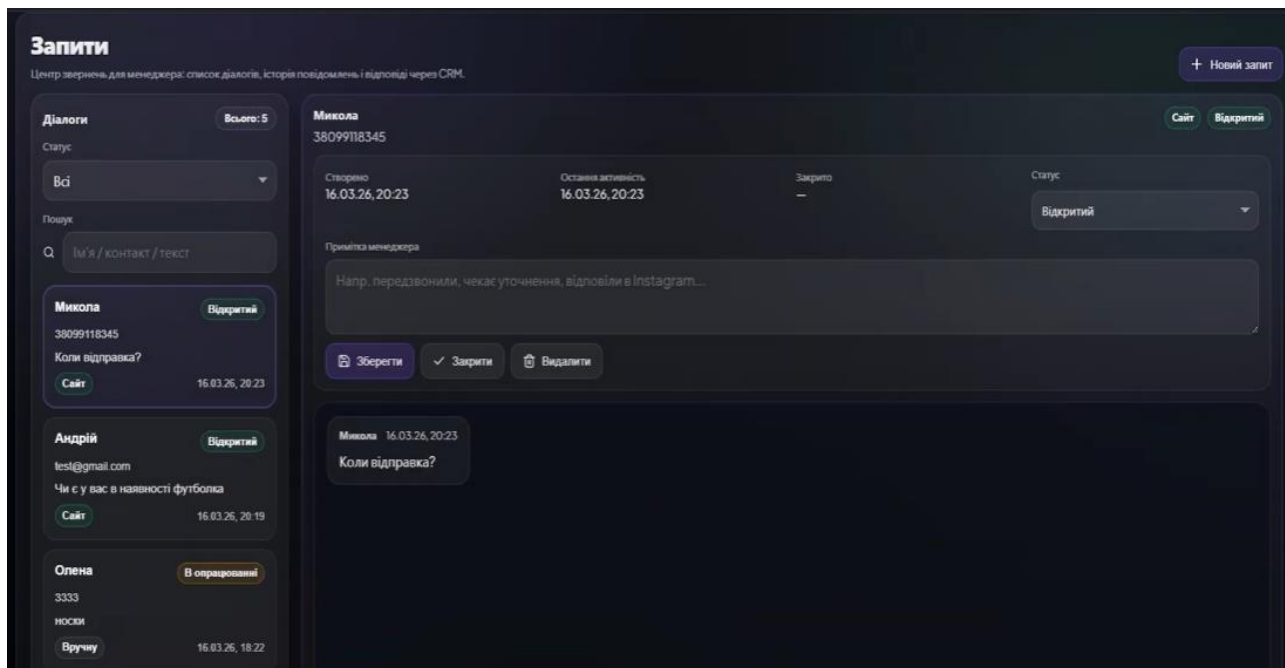
Інтерфейс модуля звітності



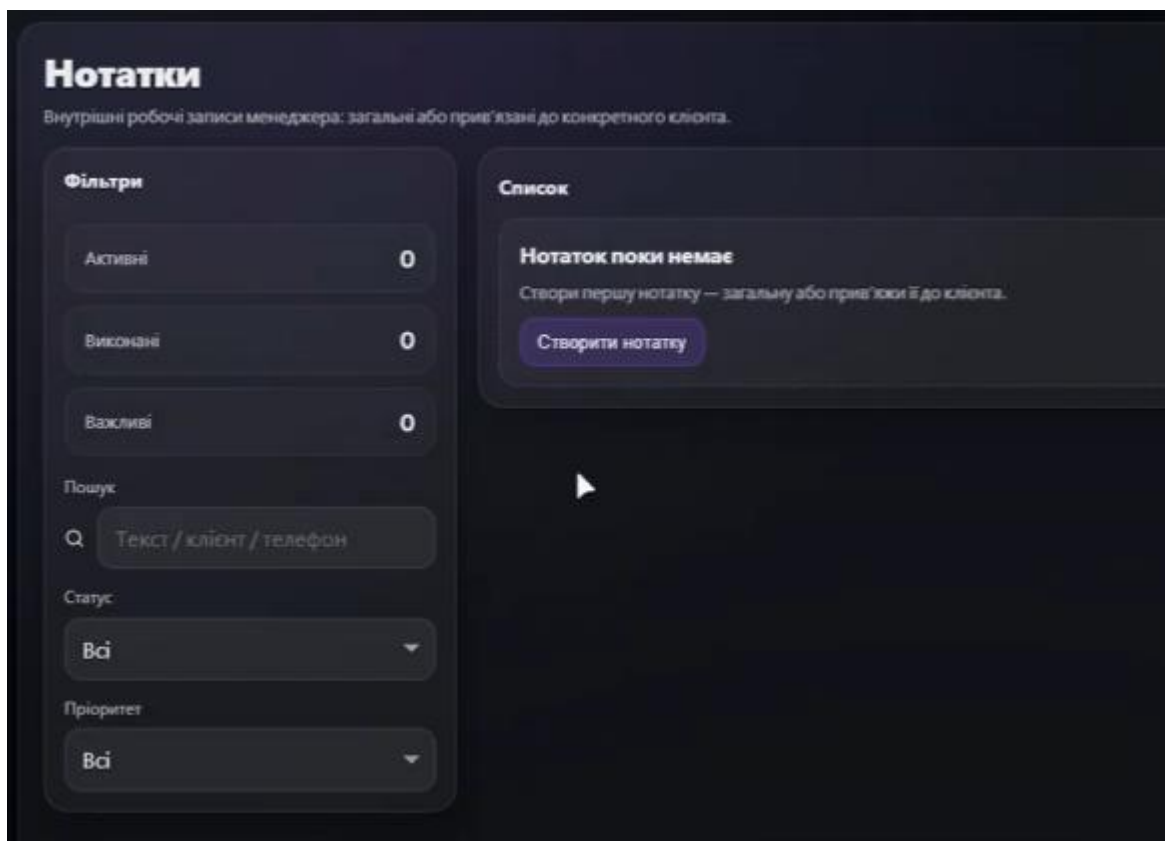
crm-report-30d (1) - Excel

Показник		
А	В	С
1	Показник	Значення
2	Період	30 днів
3	Початок	26.02.26, 00:00
4	Кінець	27.03.26, 00:11
5	Клієнтів у CRM	6
6	Активних клієнтів	6
7	Відкритих запитів	4
8	Замовлень	6
9	Дохід	3510
10	Середній чек	702
11	Одиниць товарів	6
12	Скасованих замовлень	1
13	Замовлень із сайту	2
14	Замовлень вручну	4
15	Дохід vs попередній період	100%
16	Замовлення vs попередній період	100%
17	Найкращий інтервал	13.03-17.03 • 3510
18	Топ товар	Шорти • 3 шт
19		

Інтерфейс модуля запитів клієнтів



Інтерфейс модуля внутрішніх нотаток



Нова нотатка

✕

Текст нотатки

Напр. Передзвонити клієнту завтра після 15:00...

Статус

Активна

▼

Пріоритет

Звичайна

▼

Прив'язка до клієнта

🔍

Пошук: ім'я / телефон / місто

Зберегти

Створити

Результати інтеграції із demo store

Замовлення

Список замовлень з доставкою, статусом та оплатою. Дані беруться з MongoDB.

Створити замовлення

Список

Всі

Пошук: № / клієнт / телефон

Замовлення	Клієнт	Доставка	Товари	Сума	Статус	Оплата
W-260327-5431 27.03.26, 00:13	Анастасія Іванівна 38099995115	Львів 1	Футболкаx2	860 грн	Нове	Не оплачено