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

КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Веб-платформа управління замовленнями та клієнтами малого бізнесу з використанням JavaScript»

на здобуття освітнього ступеня бакалавр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Богдан САНЖАРОВСЬКИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр. ІСД-41
Богдан САНЖАРОВСЬКИЙ
(ім'я, ПРІЗВИЩЕ)

Керівник: Оксана ТКАЛЕНКО
к.т.н.,доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

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

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Санжаровському Богдану Євгенійовичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Веб-платформа управління замовленнями та клієнтами малого бізнесу з використанням JavaScript
керівник кваліфікаційної роботи: Оксана ТКАЛЕНКО к.т.н., доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “20” лютого 2026 р. № 51

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

3. Вихідні дані кваліфікаційної роботи:
1. Науково-технічна література з розробки веб-застосунків на Node.js, Express та MongoDB.
2. Вимоги до функціональності веб-платформ автоматизації бізнес-процесів малого підприємства.
3. Стандарти та методологія проєктування REST API та реляційних баз даних документного типу.

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

1. Теоретичні засади побудови веб-платформ для управління бізнес-процесами малого бізнесу.

2. Аналіз вимог та проєктування системи: архітектура, база даних, REST API.

3. Розробка та реалізація веб-платформи: модулі автентифікації, клієнтів, замовлень, статистики; тестування системи.

5.Перелік ілюстраційного матеріалу: *презентація*

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	**Примітка**
1.	Підбір літератури, аналіз технологій Node.js, Express, MongoDB та архітектурних підходів	24.02-03.03.26	
2.	Аналіз тенденцій цифровізації малого бізнесу та огляд існуючих CRM-рішень	04.03-17.03.26	
3.	Проєктування архітектури системи, моделювання бази даних, специфікація REST API	18.03-31.03.26	
4.	Розробка серверної частини: автентифікація, CRUD-модулі клієнтів, замовлень, товарів; модуль статистики	01.04-24.04.26	
5.	Функціональне тестування системи, висновки по роботі	25.04-16.05.26	
6.	Розробка демонстраційних матеріалів, доповідь.	19.05-20.05.26	
7.	Оформлення бакалаврської роботи	21.05-30.05.26	

Здобувач вищої освіти _____ Богдан САНЖАРОВСЬКИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 65 стор., 16 рис., 11 табл., 20 джерел.

Мета роботи – розробити та науково обґрунтувати веб-платформу для управління замовленнями та клієнтами малого бізнесу з використанням JavaScript-технологій (Node.js, Express, MongoDB).

Об'єкт дослідження – процес управління замовленнями та клієнтами в малому бізнесі у сфері технічних послуг.

Предмет дослідження – методи та засоби розробки веб-платформ на основі JavaScript-технологій для автоматизації операційних процесів управління замовленнями та клієнтами малого бізнесу.

Короткий зміст роботи. У бакалаврській роботі проведено аналіз тенденцій цифровізації малого бізнесу та виявлено обмеження існуючих комерційних CRM-систем для мікропідприємств сфери технічних послуг. Обґрунтовано доцільність розробки власної мінімалістичної веб-платформи. Спроєктовано трирівневу клієнт-серверну архітектуру з REST API, чотири Mongoose-моделі бази даних (User, Client, Order, Product) та специфікацію 27 ендпоінтів. Реалізовано серверну частину на Node.js/Express з JWT-автентифікацією та рольовим доступом (адміністратор/менеджер), CRUD-модулі для управління клієнтами, замовленнями й каталогом товарів і послуг, а також модуль статистики з формуванням 25+ аналітичних показників. Клієнтська частина побудована на Vanilla JS без фреймворків. Проведено функціональне тестування за 15 сценаріями, всі сценарії пройдено успішно.

КЛЮЧОВІ СЛОВА: ВЕБ-ПЛАТФОРМА, CRM-СИСТЕМА, NODE.JS, EXPRESS, MONGODB, REST API, JWT-АВТЕНТИФІКАЦІЯ, УПРАВЛІННЯ ЗАМОВЛЕННЯМИ, МАЛИЙ БІЗНЕС, MONGOOSE.

ЗМІСТ

ВСТУП

Актуальність теми. Управління взаємовідносинами з клієнтами та відстеження замовлень — одне з ключових завдань будь-якого малого бізнесу, що надає послуги чи продає товари. Проте більшість невеликих компаній досі покладаються на таблиці Microsoft Excel, записники або окремі месенджери для ведення обліку. Такий підхід має суттєві обмеження: дані розпорошені між різними файлами, відсутня єдина картка клієнта з історією замовлень, неможливо швидко відстежити статус виконання або заборгованість за оплатою [1].

Існуючі комерційні CRM-системи — HubSpot CRM, Pipedrive, Zoho CRM, Salesforce — здебільшого орієнтовані на середній і великий бізнес або мають складний інтерфейс і коштують від кількох десятків до кількох сотень доларів на місяць. Для мікробізнесу з командою 3–10 осіб ці витрати є надмірними, а сам функціонал таких систем виявляється занадто громіздким для повсякденних потреб [2].

Актуальним рішенням у цьому контексті є розробка власної спеціалізованої веб-платформи, яка враховує конкретні бізнес-процеси компанії, не містить зайвого функціоналу та не потребує щомісячної абонентської плати. Сучасний JavaScript-стек — Node.js, Express та MongoDB — дозволяє побудувати таку систему з відносно невеликими зусиллями, забезпечуючи при цьому достатню продуктивність і надійність для потреб малого бізнесу [3].

Для демонстрації та тестування платформи в роботі використовується умовне модельне підприємство «Безпека Плюс» — змодельована компанія, що спеціалізується на продажу, монтажі й налаштуванні систем відеоспостереження та охоронної сигналізації. Такий тип бізнесу обрано як типовий представник сфери технічних послуг: він поєднує продаж обладнання (товари) і виконання робіт (послуги) в межах одного замовлення, має потребу у відстеженні стану оплати та веденні клієнтської бази. Для цього класу підприємств ручний облік у таблицях особливо болючий: при 20–40 активних замовленнях одночасно неможливо оперативно перевірити статус виконання, знайти клієнта чи дізнатися

суму заборгованості. Розроблена платформа вирішує ці проблеми, і саме на прикладі «Безпека Плюс» наповнено тестову базу даних та перевірено коректність роботи всіх модулів.

Таким чином, тема кваліфікаційної роботи є актуальною як з теоретичної точки зору — у частині вивчення підходів до побудови веб-застосунків для управління бізнес-процесами, — так і з практичної, оскільки результатом є готова функціонуюча система, придатна до використання підприємствами відповідного профілю.

Мета роботи — розробити та науково обґрунтувати веб-платформу для управління замовленнями та клієнтами малого бізнесу, реалізовану з використанням JavaScript-технологій (Node.js, Express, MongoDB), яка забезпечує автоматизацію ключових операційних процесів компанії.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

1) проаналізовано сучасний стан і тенденції цифровізації малого бізнесу та визначено вимоги до спеціалізованих веб-рішень для управління замовленнями;

2) досліджено архітектурні підходи до побудови веб-застосунків та спроєктовано архітектуру платформи, структуру бази даних і REST API;

3) розроблено та реалізовано серверну частину платформи на Node.js/Express із системою JWT-автентифікації, рольового доступу, CRUD-модулів та модуля статистики;

4) проведено функціональне тестування системи та підтверджено відповідність реалізованого функціоналу сформульованим вимогам.

Об'єкт дослідження — процес управління замовленнями та клієнтами в малому бізнесі у сфері надання технічних послуг (продаж, монтаж і обслуговування обладнання).

Предмет дослідження — методи та засоби розробки веб-платформ на основі JavaScript-технологій для автоматизації операційних процесів управління замовленнями та клієнтами малого бізнесу.

Методи дослідження. У процесі виконання кваліфікаційної роботи використовувалися: метод аналізу та синтезу, структурно-функціональний аналіз,

об'єктно-орієнтоване проєктування, UML-моделювання, прототипування та функціональне тестування.

Практична значущість отриманих результатів. Розроблена веб-платформа є повністю функціонуючою системою, протестованою на прикладі модельного підприємства «Безпека Плюс», що забезпечує ведення клієнтської бази, створення та відстеження замовлень з обліком товарів і послуг, моніторинг статусів і стану оплати, а також перегляд зведеної аналітики по клієнтах, замовленнях та ефективності менеджерів. Підходи до організації серверної частини, рольова модель контролю доступу, структура REST API та схема бази даних можуть бути використані як основа при створенні аналогічних систем для підприємств малого бізнесу у сферах надання послуг або дрібнооптової торгівлі.

Апробація результатів. Окремі результати дослідження щодо застосування стеку Node.js, Express та MongoDB для побудови веб-систем автоматизації процесів збору клієнтських даних та управління замовленнями було представлено у тезах доповіді: Руденко О. О., Санжаровський Б. Є. «Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот» на VII Міжнародній науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». — Київ : ДУІКТ, 2026. — С. 37-38.

1 ТЕОРЕТИЧНІ ЗАСАДИ ПОБУДОВИ ВЕБ-ПЛАТФОРМ ДЛЯ УПРАВЛІННЯ БІЗНЕС-ПРОЦЕСАМИ

1.1 Цифровізація малого бізнесу: стан та тенденції

Малий бізнес — підприємства зі штатом до 50 осіб і річним оборотом у межах кількох мільйонів гривень — становить основу економіки більшості країн. За даними Державної служби статистики, частка малих підприємств у загальній кількості суб'єктів господарювання в Україні перевищує 95 %, проте рівень їхньої цифровізації залишається суттєво нижчим, ніж у великого бізнесу [1].

Під цифровізацією бізнес-процесів розуміється переведення ручних або паперових операцій у цифровий формат з метою підвищення швидкості, точності та контрольованості роботи. Для малого бізнесу це насамперед означає автоматизацію управління клієнтами, обліку замовлень, контролю стану оплати та формування звітності [2].

Дослідження Крауса та ін. (2021) показує, що малі підприємства, які впровадили спеціалізовані цифрові інструменти управління, демонструють вищу операційну ефективність і менше втрат через людський фактор — помилки при ручному введенні даних, дублювання записів, загублені заявки. При цьому ключовою перешкодою для цифровізації є не брак технологій, а невідповідність готових рішень специфіці конкретного бізнесу [2].

Типовий сценарій у малому бізнесі сфери послуг виглядає так: нова заявка фіксується в телефонному дзвінку або повідомленні месенджера, вноситься менеджером до Excel-таблиці, а контроль виконання та стан оплати відстежуються вручну або в окремих нотатках. При зростанні кількості замовлень понад 20–30 на місяць така система стає нежиттєздатною: замовлення губляться, клієнти не отримують зворотного зв'язку вчасно, заборгованість виявляється із запізненням.

Для демонстрації та тестування розробленої платформи у роботі використовується умовне модельне підприємство «Безпека Плюс» —

змодельована компанія, що спеціалізується на продажу, монтажі й налаштуванні систем відеоспостереження та охоронної сигналізації. Цей профіль діяльності обрано як типовий представник сфери технічних послуг: він поєднує продаж обладнання (товари) і виконання монтажних робіт (послуги) в межах одного замовлення, передбачає ведення клієнтської бази та відстеження стану оплати. Саме на прикладі цього умовного підприємства наповнено тестову базу даних та перевірено коректність роботи всіх модулів системи.

Серед можливих підходів до розв'язання типових операційних проблем малого бізнесу сфери послуг можна виділити чотири основних: ручні таблиці (Excel, Google Sheets), хмарні SaaS CRM-системи (HubSpot, Pipedrive, Zoho CRM), ERP-системи корпоративного рівня (NetSuite, SAP Business One) та розробка власного програмного рішення. Кожен підхід має свої переваги та обмеження.

Таблиця 1.1

Порівняльний аналіз підходів до автоматизації управління замовленнями

Критерій	**Excel / таблиці**	**Хмарні CRM (HubSpot, Pipedrive, Zoho)**	**Власна платформа (проєкт)**	**ERP-системи (NetSuite, SAP B1)**
Вартість	Безкоштовно	$15–90/міс. на користувача	Одноразова розробка	$200+/міс.
Адаптація під специфіку бізнесу	Низька	Середня	Висока	Середня
Складність освоєння	Низька	Висока	Низька	Дуже висока
Контроль над даними	Повний (локально)	Обмежений (хмара)	Повний	Обмежений (SaaS)
Спеціалізована бізнес-логіка	Відсутня	Часткова	Повна	Часткова

Продовження таблиці 1.1

Вбудована аналітика	Ручна	Є, але загальна	Спеціалізована	Є
Рольовий доступ	Відсутній	Є	Є (admin / manager)	Є

Як видно з таблиці 1.1, хмарні CRM-системи пропонують більше функцій порівняно з таблицями, однак їх вартість і надлишковий функціонал роблять їх невигідними для мікробізнесу. Власна платформа є оптимальним варіантом за критеріями адаптованості, вартості та повноти контролю над даними. Важливо зазначити, що мета розробки не конкурувати з корпоративними системами, а закрити конкретний набір задач без надлишку функціоналу.

Сучасний стан JavaScript-екосистеми дозволяє розробити повноцінний серверний застосунок для такого завдання силами одного розробника за розумний термін. Node.js, Express і MongoDB утворюють технологічний стек, що добре підходить для прикладних задач управління даними в масштабі малого бізнесу. Детальний огляд цього стеку наведено у підрозділі 1.3.

1.2 Класифікація веб-застосунків та архітектурні підходи

Веб-застосунок — це програмне забезпечення, що виконується на веб-сервері та взаємодіє з користувачем через браузер за допомогою протоколу HTTP/HTTPS. На відміну від десктопного програмного забезпечення, веб-застосунок не потребує встановлення і доступний з будь-якого пристрою з підключенням до мережі [3].

За архітектурою представлення виділяють два основних типи веб-застосунків: багатосторінкові (MPA — Multi-Page Application) та односторінкові (SPA — Single-Page Application). У MPA кожна дія користувача ініціює запит до сервера, який повертає повну HTML-сторінку. У SPA початкове завантаження відбувається один раз, а подальші взаємодії здійснюються через AJAX-запити з

частковим оновленням DOM без перезавантаження сторінки. SPA забезпечують плавнішу взаємодію, але потребують JavaScript-фреймворку (React, Vue, Angular) та складнішої організації клієнтського коду [3].

Платформа, розроблена в рамках даної роботи, використовує гібридний підхід. Кожен функціональний розділ системи — клієнти, замовлення, статистика тощо — є окремою HTML-сторінкою (MPA-структура навігації). Проте всередині кожної сторінки дані завантажуються та оновлюються асинхронно через fetch()-запити до REST API без перезавантаження сторінки (SPA-поведінка в межах одного розділу). Такий підхід свідомо обрано замість повноцінного SPA-фреймворку: він спрощує структуру проєкту, усуває залежність від фреймворку та знижує поріг підтримки системи, зберігаючи при цьому зручність роботи з інтерфейсом.

Ключовим архітектурним рішенням є розподіл застосунку на клієнтську і серверну частини. Клієнтська частина — статичні HTML-файли з підключеним JavaScript — відповідає за відображення даних та взаємодію з користувачем. Серверна частина — Node.js/Express — обробляє HTTP-запити, реалізує бізнес-логіку та взаємодіє з базою даних через Mongoose. Такий поділ забезпечує незалежність шарів: клієнтська частина не знає деталей реалізації сервера і навпаки [4].

Комунікація між клієнтом і сервером реалізована через REST API (Representational State Transfer). Цей архітектурний стиль, описаний Роєм Філдінгом у дисертації 2000 року, визначає набір обмежень для побудови масштабованих веб-систем. Ключові принципи REST, що реалізовані у проєкті: одноманітний інтерфейс — кожен ресурс (clients, orders, products) має власний базовий URL і стандартний набір методів (GET, POST, PUT, DELETE, PATCH); відсутність стану на сервері між запитами (stateless) — кожен запит містить усі необхідні дані для його обробки; клієнт-серверне розмежування — фронтенд і бекенд розвиваються незалежно [5].

На рівні серверної частини застосовується адаптований патерн MVC (Model-View-Controller). Моделі (models/) описують структуру даних і

взаємодіють з MongoDB через Mongoose. Контролери (controllers/) містять бізнес-логіку — обробку запитів, валідацію, формування відповідей. Маршрути (routes/) зв'язують HTTP-ендпоінти з відповідними функціями контролерів. Роль View у даному проєкті виконує клієнтська частина — статичні HTML/JS-файли, що рендеряться в браузері [4].

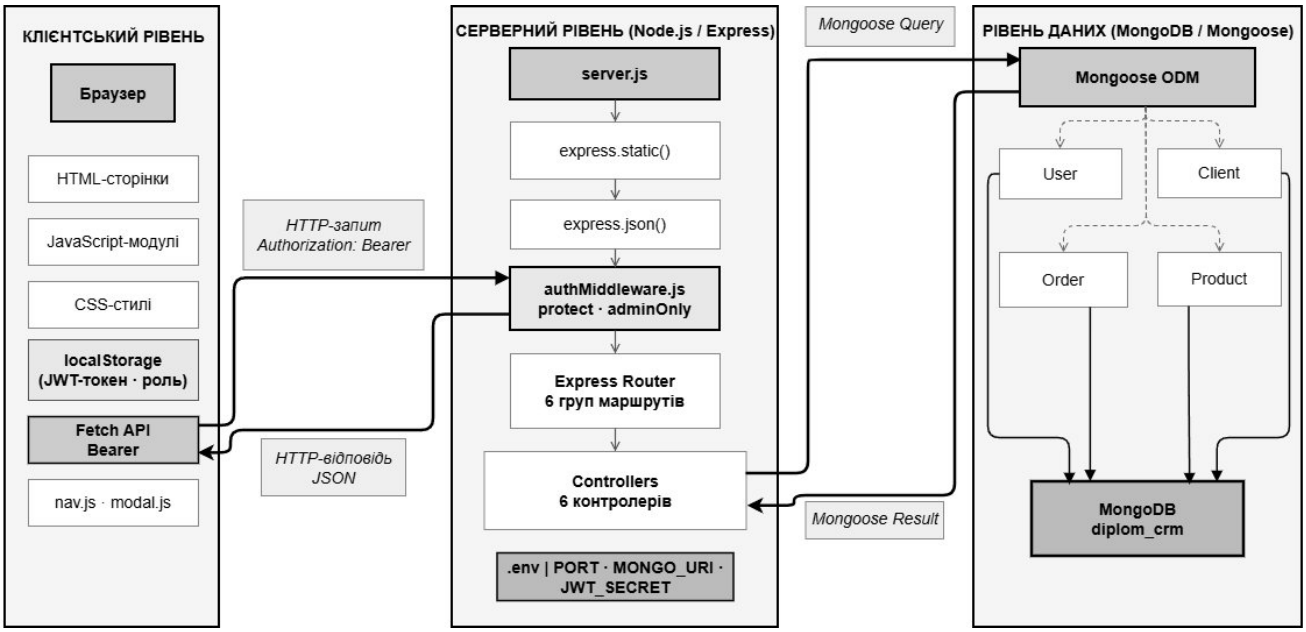


Рис. 1.1 Схема клієнт-серверної архітектури веб-платформи

Для захисту API від несанкціонованого доступу використовується механізм токенової автентифікації на базі JSON Web Tokens (JWT). При успішному вході користувач отримує підписаний токен, який зберігається у localStorage браузера та передається у заголовку Authorization кожного подальшого запиту. Сервер верифікує токен через middleware функцію protect, не звертаючись до бази даних — це відповідає принципу stateless архітектури REST [6]. Рольовий контроль доступу забезпечується додатковим middleware adminOnly, що перевіряє поле role у декодованому токені.

1.3 Огляд технологічного стеку: Node.js, Express, MongoDB, Mongoose

Node.js. Node.js — серверне середовище виконання JavaScript, засноване на рушії V8 від Google. Його ключова особливість — асинхронна неблокуюча

модель введення/виведення на основі Event Loop. Замість того щоб очікувати завершення операцій вводу/виводу (запит до бази даних, читання файлу), Node.js реєструє callback або Promise і одразу переходить до обробки наступного запиту. Коли операція завершується, обробник викликається через чергу подій. Це дозволяє ефективно обслуговувати велику кількість одночасних з'єднань при відносно невисокому споживанні пам'яті [7].

Суттєвою перевагою Node.js для даного проєкту є використання JavaScript як на клієнті, так і на сервері. Це дозволяє застосовувати єдиний формат обміну даними (JSON) на обох рівнях, зменшує контекстне перемикання між мовами та спрощує розуміння системи в цілому. У проєкті Node.js використовується з модульною системою CommonJS (require / module.exports), що забезпечує сумісність з усіма задіяними npm-пакетами.

Express.js. Express — мінімалістичний та гнучкий веб-фреймворк для Node.js. Він надає необхідний мінімум для побудови HTTP-сервера: визначення маршрутів, ланцюжки middleware, зручний доступ до параметрів запиту та формування відповідей. На відміну від більш «самодостатніх» фреймворків (NestJS, Fastify з вбудованим DI-контейнером), Express дає розробнику максимальний контроль над архітектурою без нав'язування додаткових абстракцій [3].

У проєкті Express виконує такі функції: реєстрація маршрутів для шести ресурсів API (/api/auth, /api/clients, /api/orders, /api/products, /api/users, /api/statistics); підключення middleware — express.json() для парсингу тіла запиту у форматі JSON та express.static() для роздачі статичних HTML/CSS/JS-файлів клієнтської частини; передача управління між middleware-функціями через механізм next(). Проєкт використовує Express версії 5 (^5.2.1), яка підтримує async/await у обробниках маршрутів без додаткових обгорток для перехоплення помилок.

MongoDB. MongoDB — документ-орієнтована СУБД, що зберігає дані у форматі JSON-подібних документів (BSON — Binary JSON). Ключова відмінність

від реляційних баз даних: відсутність фіксованої схеми таблиць, нативна підтримка вкладених об'єктів і масивів у документі, гнучка мова запитів MQL [8].

Для даного проєкту MongoDB є оптимальним вибором з кількох конкретних причин. По-перше, модель Order містить вкладений масив products (список товарів із кількістю) та масив statusHistory (історія змін статусу з датою та ідентифікатором автора) — у реляційній базі це потребувало б двох додаткових таблиць і JOIN-запитів при кожному зверненні до замовлення. По-друге, відсутність жорсткої схеми дозволяла додавати нові поля до документів у процесі розробки без складних міграцій. По-третє, нативна підтримка JSON-формату природно вписується в JavaScript-стек.

Таблиця 1.2

Порівняння реляційних і документ-орієнтованих СУБД стосовно задач проєкту

Критерій	**Реляційні СУБД (PostgreSQL, MySQL)**	**MongoDB (документ-орієнтована)**
Схема даних	Фіксована, потребує міграцій	Гнучка, без жорсткої схеми таблиць
Одиниця зберігання	Рядки таблиць з фіксованими стовпцями	JSON-документи з довільними вкладеними полями
Вкладені масиви у записі	Реалізуються через окремі таблиці + JOIN	Нативна підтримка: products[], statusHistory[]
Горизонтальне масштабування	Складне, потребує додаткових рішень	Вбудована підтримка (sharding)
Мова запитів	SQL	MQL (MongoDB Query Language)
Інтеграція з Node.js	Через ORM (Sequelize, Prisma)	Mongoose ODM або нативний MongoDB driver
Відповідність структурі проєкту	Можлива, але потребує більше JOIN-запитів	Оптимальна: вкладені товари і історія статусів в одному документі

Таблиця 1.2 наочно показує, що структура даних проєкту — зокрема вкладені масиви products[] та statusHistory[] в моделі Order — природно відображається в документах MongoDB без потреби у складних JOIN-запитах, характерних для реляційних СУБД.

Mongoose. Mongoose — ODM (Object Document Mapper) для MongoDB у Node.js. Він додає до гнучкості MongoDB структурованість на рівні коду: кожна колекція описується схемою (Schema), що визначає типи полів, значення за замовчуванням, обов'язковість, enum-обмеження та middleware-хуки (наприклад, виклик функцій до збереження документа) [10]. Хоча MongoDB не потребує схем на рівні бази даних, Mongoose забезпечує передбачувану структуру документів і спрощує валідацію вхідних даних безпосередньо в коді моделей.

У проєкті Mongoose використовується для опису чотирьох колекцій. Модель User визначає поля name, email, password та role (enum: admin | manager) з обов'язковістю та унікальністю email. Модель Client зберігає контактні дані клієнта та посилання createdBy на користувача, що додав запис. Модель Order є найскладнішою: містить вкладений масив products (ObjectId-посилання на Product + кількість), масив statusHistory (статус, дата, автор зміни), поля для обліку оплати (totalAmount, paidAmount, paymentStatus) та дедлайн. Модель Product описує єдиний довідник для товарів і послуг із полем type (enum: product | service) і прапорцем isActive. Функція populate() Mongoose використовується для автоматичної підстановки пов'язаних документів при вибірці — наприклад, заповнення даних клієнта та переліку товарів у відповіді на запит про замовлення.

Додаткові залежності проєкту. Повний перелік npm-пакетів, використаних у проєкті, наведено у таблиці 1.3.

Таблиця 1.3

Залежності проєкту з поясненням призначення (package.json)

Пакет	**Версія**	**Призначення у проєкті**
express	^5.2.1	Веб-фреймворк, маршрутизація HTTP-запитів

Продовження таблиці 1.3

mongoose	^9.6.1	ODM для MongoDB: схеми, валідація, populate
jsonwebtoken	^9.0.3	Генерація та верифікація JWT-токенів
bcryptjs	^3.0.3	Хешування паролів (алгоритм bcrypt, salt rounds = 10)
dotenv	^17.4.2	Завантаження конфігурації зі змінних середовища (.env)
cors	^2.8.6	Middleware для управління CORS-заголовками
nodemon (dev)	^3.1.14	Автоматичний перезапуск сервера при змінах коду (тільки розробка)

З таблиці 1.3 видно, що проєкт має мінімальний набір залежностей — шість production-пакетів і один dev-пакет. Така ощадливість свідомо закладена в архітектуру: кожна залежність виконує чітко визначену функцію, і жодна не є зайвою. Відсутність UI-фреймворків і важких ORM підтверджує підхід до мінімалістичної розробки, орієнтованої на конкретні задачі.

1.4 Аналіз існуючих рішень для управління замовленнями та клієнтами

Перед формуванням вимог до власної платформи було проведено огляд провідних готових рішень, доступних на ринку CRM та управління замовленнями. Мета аналізу — визначити, чи здатне будь-яке з них задовольнити специфічні потреби мікробізнесу сфери технічних послуг без надлишкового функціоналу і при прийнятній вартості.

HubSpot CRM — один із найпопулярніших безкоштовних CRM у світі (США). Базовий план є безкоштовним, але містить обмежений функціонал; розширені можливості починаються від $45/міс. HubSpot орієнтований на

управління воронкою продажів і маркетингову аналітику. Для задач проєкту він має принципові обмеження: відсутній облік конкретних товарів і послуг у складі замовлення, немає історії зміни статусів виконання, інтерфейс перевантажений маркетинговими інструментами, що не є релевантними для монтажного підприємства.

Pipedrive — CRM-система (Естонія), орієнтована на управління угодами у вигляді Kanban-дошки. Добре підходить для відстеження воронки продажів, але принципово не призначена для обліку монтажних замовлень: відсутній список товарів і послуг у складі угоди, не підтримується історія статусів виконання (відмінних від статусів угоди), немає розрахунку заборгованості по оплаті.

Zoho CRM — багатофункціональна хмарна CRM (Індія) з широкими можливостями кастомізації. Теоретично підходить для задач проєкту, але має суттєві практичні обмеження: налаштування специфічної логіки (наприклад, автоматичний розрахунок paymentStatus або масив товарів у замовленні) потребує значних зусиль з адміністрування системи, вартість починається від $14/міс. на користувача, а безкоштовний план обмежений трьома обліковими записами.

Google Sheets / Microsoft Excel — найбільш поширений інструмент управління даними в мікробізнесі. Повністю безкоштовний і знайомий більшості користувачів. Проте принципово не може замінити спеціалізовану систему: відсутні автоматизація бізнес-логіки, рольовий доступ для різних менеджерів, автоматичний розрахунок статусу оплати, прив'язка клієнтів до замовлень як пов'язаних сутностей, а також вбудована статистика та аналітика.

NetSuite (Oracle) та SAP Business One — ERP-системи корпоративного рівня з повним охопленням бізнес-процесів підприємства. Вартість впровадження та місячна підписка (від $200/міс.) роблять їх економічно недоцільними для мікробізнесу. Крім того, подібні системи потребують тривалого налаштування та навчання персоналу.

Проведений аналіз показав, що жодне з розглянутих рішень не задовольняє одночасно трьом ключовим критеріям задачі: прийнятна вартість без абонентської плати, простота освоєння без тривалого навчання, наявність специфічних функцій

для обліку монтажних замовлень (поєднання товарів і послуг в одному записі, історія статусів, автоматичний розрахунок оплати, рольовий доступ). Це і обґрунтовує рішення розробити власну мінімалістичну веб-платформу.

Таблиця 1.4

Порівняльний аналіз існуючих систем управління замовленнями і клієнтами

Система	**Вартість**	**Орієнтація**	**Обмеження для задач проєкту**
HubSpot CRM	Безкоштовно / $45+/міс.	Sales & marketing CRM	Орієнтований на воронку продажів, немає обліку монтажних замовлень та товарів, обмежена кастомізація в безкоштовному плані
Pipedrive	$14–99/міс.	Sales CRM	Відсутній облік товарів/послуг у замовленні, немає історії статусів монтажу, обмежена аналітика
Zoho CRM	$14–52/міс.	Mid-market CRM	Широкий функціонал, але складне налаштування під специфічну логіку монтажних послуг, англомовний інтерфейс
Google Sheets / Excel	Безкоштовно	Таблиці	Відсутні автоматизація, рольовий доступ, прив'язка клієнт–замовлення, автоматичний розрахунок оплати та статистика
NetSuite / SAP B1	$200+/міс.	ERP (великий бізнес)	Надто дорогі й складні для мікробізнесу, потребують тривалого впровадження та навчання

Продовження таблиці 1.4

Власна платформа (даний проєкт)	Одноразова розробка	Сфера послуг / мікробізнес	Повна відповідність вимогам: облік товарів і послуг, історія статусів, автоматична нумерація, рольовий доступ, спеціалізована аналітика

Таблиця 1.4 підтверджує: власна розробка є єдиним варіантом, що одночасно задовольняє всі специфічні вимоги без надлишкового функціоналу та обов'язкової місячної підписки. Це не означає, що розроблена платформа перевершує комерційні системи в цілому — вона навмисно вузькоспеціалізована і вирішує конкретний набір задач максимально прямолінійно.

У першому розділі проведено теоретичний аналіз, що охоплює предметну область, технологічний стек і конкурентне середовище розробленої платформи. На підставі розглянутого матеріалу сформульовано такі висновки:

– Більшість малих підприємств не мають спеціалізованих цифрових інструментів для управління замовленнями, використовуючи натомість таблиці та ручний облік. Зі зростанням обсягу роботи до 20–40 активних замовлень одночасно такий підхід призводить до регулярних операційних помилок і втрат.

– Готові CRM-системи (HubSpot, Pipedrive, Zoho CRM) та ERP (NetSuite, SAP Business One) не задовольняють одночасно критеріям вартості, простоти освоєння та наявності специфічних функцій обліку монтажних замовлень. Це обґрунтовує доцільність розробки власної мінімалістичної платформи.

– Гібридна архітектура MPA-навігація + SPA-поведінка всередині сторінки через fetch() та REST API дозволяє відмовитись від важких JavaScript-фреймворків, зберігши при цьому зручність роботи з інтерфейсом. Це спрощує структуру проєкту та знижує поріг підтримки.

– JavaScript-стек (Node.js + Express + MongoDB + Mongoose) є обґрунтованим вибором для даного проєкту: єдина мова на клієнті та сервері, нативний JSON, гнучка схема MongoDB для вкладених структур (products[],

statusHistory[]), мінімальний набір залежностей та зріла npm-екосистема для автентифікації та безпеки.

– Механізм JWT-автентифікації з двома рівнями middleware (protect та adminOnly) забезпечує stateless-захист API відповідно до принципів REST і дозволяє реалізувати рольову модель доступу без збереження сесій на сервері.

Результати теоретичного аналізу склали основу для проєктування архітектури платформи, моделей бази даних та специфікації REST API, що розглядається в наступному розділі.

2 АНАЛІЗ ВИМОГ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Характеристика предметної області та бізнес-процесів

Для визначення вимог до веб-платформи необхідно чітко окреслити предметну область та типові бізнес-процеси підприємства, для якого вона розробляється. У даній роботі як модельний об'єкт використовується умовне підприємство «Безпека Плюс» — змодельована компанія сфери технічних послуг, що надає комплексні послуги з відеоспостереження та охорони: продаж обладнання, виїзний монтаж і пусконалагоджувальні роботи. Такий профіль діяльності обрано як типовий для мікробізнесу технічних послуг, де в одному замовленні поєднуються товари (камери, реєстратори, кабелі) і послуги (монтаж, налаштування). Саме на прикладі цього підприємства сформовано тестову базу даних і перевірено роботу всіх модулів системи.

Основна операційна одиниця — замовлення. Воно виникає після звернення клієнта (дзвінок, повідомлення), проходить кілька стадій виконання і завершується оплатою та здачею об'єкта. Типовий бізнес-процес обробки замовлення складається з таких кроків:

– Клієнт звертається до компанії; менеджер фіксує контактні дані та формує замовлення з переліком товарів та/або послуги.

– Замовленню присвоюється статус «нове» (new) і унікальний номер.

– Після початку виконання статус змінюється на «в роботі» (in_progress); зміна фіксується в історії.

– По завершенні робіт статус переводиться на «виконано» (completed). Якщо замовлення не відбулося — «скасовано» (cancelled).

– Протягом або після виконання клієнт здійснює оплату; менеджер фіксує сплачену суму; система автоматично визначає статус оплати.

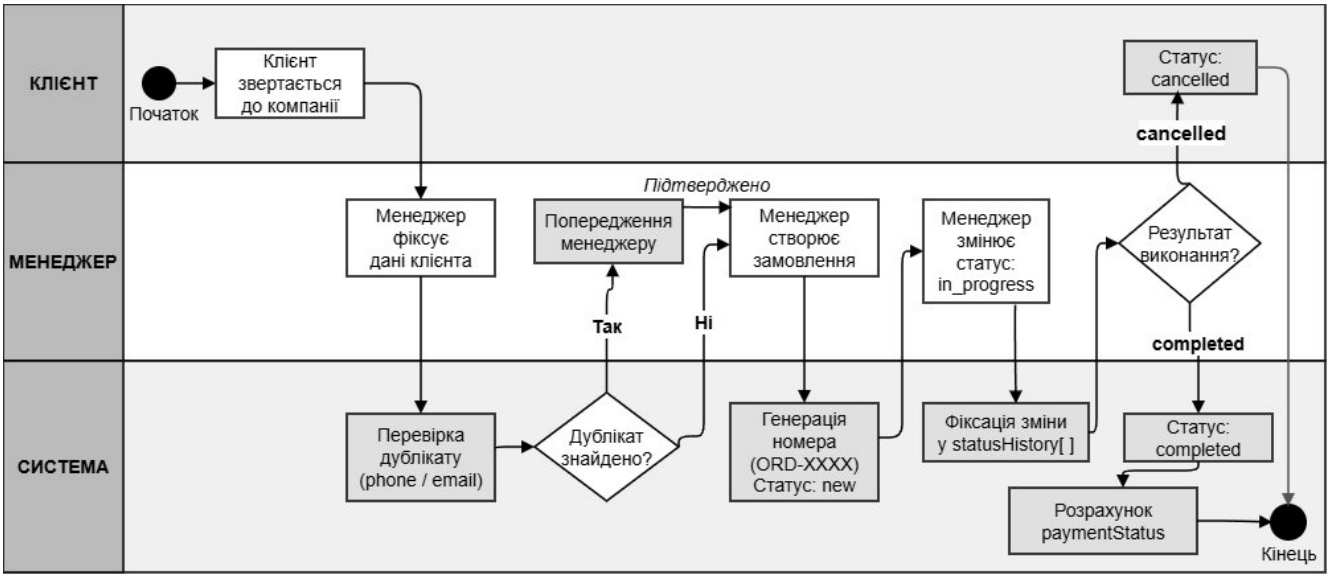


Рис. 2.1 Діаграма бізнес-процесу обробки замовлення

Крім замовлень, система повинна вести базу клієнтів — фізичних та юридичних осіб, що замовляють послуги. Кожен клієнт має контактні дані (ім'я, телефон, email, адреса, компанія) і пов'язаний з усіма своїми замовленнями. Реквізити клієнта не дублюються у кожному замовленні — натомість зберігається ObjectId-посилання, яке Mongoose розгортає (populate) при потребі.

Окремий довідник об'єднує товари та послуги, що реалізує компанія. Єдина колекція Product з полем type (product | service) дозволяє уникнути дублювання логіки: і товари, і послуги мають назву, категорію, ціну та прапорець активності. При цьому в замовленні можна одночасно вказати кілька товарів (масив products[]) та одну послугу (serviceId).

Система передбачає два типи користувачів з різними повноваженнями. Адміністратор має повний доступ до всіх модулів і додатково керує обліковими записами менеджерів. Менеджер — рядовий оператор, що щоденно працює з клієнтами, замовленнями та каталогом, але не має доступу до управління персоналом.

Таблиця 2.1

Ролі користувачів системи та їх повноваження

Роль	**Доступні модулі та операції**	**Обмеження**
admin (адміністратор)	Повний доступ до всіх модулів: клієнти, замовлення, товари/послуги, статистика. Управління менеджерами (CRUD): перегляд, додавання, редагування, видалення. Перегляд статистики окремого менеджера.	Не може видалити власний акаунт (захист від самовидалення).
manager (менеджер)	Клієнти: перегляд, додавання, редагування, видалення. Замовлення: перегляд, створення, редагування, видалення. Товари/послуги: перегляд, додавання, редагування, деактивація. Статистика: перегляд зведених показників.	Немає доступу до модуля управління менеджерами (/api/users — захищено adminOnly). Не бачить особистої статистики інших менеджерів.

Таблиця 2.1 ілюструє принцип найменших привілеїв: кожен користувач отримує рівно той доступ, що необхідний для виконання його функцій. Технічно це реалізується через два middleware — protect (перевірка JWT) та adminOnly (перевірка role === 'admin'), що утворюють дворівневий захист.

2.2 Функціональні та нефункціональні вимоги до системи

Вимоги до системи сформульовані на основі аналізу бізнес-процесів, описаних у підрозділі 2.1. Вони поділяються на функціональні (що система повинна робити) та нефункціональні (яким чином і з якими характеристиками).

Функціональні вимоги описують конкретні дії, що система повинна виконувати. Їх згруповано за модулями системи та наведено у таблиці 2.2.

Таблиця 2.2

Функціональні вимоги до веб-платформи

№	Модуль	Функціональна вимога	Рівень доступу
Ф-01	Автентифікація	Система повинна надавати доступ користувачу після введення коректного email та пароля і повертати JWT-токен терміном дії 8 годин.	Усі
Ф-02	Автентифікація	Система повинна відмовляти у доступі при невірному email або паролі з відповідним повідомленням.	Усі
Ф-03	Автентифікація	Кожен захищений запит до API повинен перевіряти наявність та коректність JWT-токена у заголовку Authorization.	Система
Ф-04	Клієнти	Система повинна надавати можливість перегляду списку клієнтів з текстовим пошуком за ПІБ, телефоном та email.	admin, manager
Ф-05	Клієнти	Система повинна перевіряти наявність дублікату клієнта за телефоном або email перед збереженням нового запису.	admin, manager
Ф-06	Клієнти	Система повинна надавати перегляд повної картки клієнта з усіма його замовленнями.	admin, manager
Ф-07	Клієнти	Система повинна забезпечувати створення, редагування та видалення записів клієнтів.	admin, manager

Продовження таблиці 2.2

Ф-08	Замовлення	Система повинна автоматично генерувати унікальний номер замовлення у форматі ORD-XXXX при створенні.	Система
Ф-09	Замовлення	Замовлення повинне підтримувати прив'язку до клієнта, переліку товарів (з кількістю) та послуги одночасно.	admin, manager
Ф-10	Замовлення	Система повинна підтримувати чотири статуси замовлення: нове, в роботі, виконано, скасовано — та фіксувати кожну зміну статусу в історії.	admin, manager
Ф-11	Замовлення	Система повинна автоматично розраховувати статус оплати (не оплачено / частково / оплачено) на основі totalAmount та paidAmount.	Система
Ф-12	Замовлення	Система повинна підтримувати фільтрацію замовлень за статусом та клієнтом, а також пошук за назвою, номером і ПІБ клієнта.	admin, manager
Ф-13	Товари / Послуги	Система повинна підтримувати єдиний довідник для товарів (type: product) і послуг (type: service) з категоріями та ціною.	admin, manager
Ф-14	Товари / Послуги	Система повинна дозволяти активацію та деактивацію позицій (isActive) без їх видалення.	admin, manager
Ф-15	Менеджери	Адміністратор повинен мати можливість переглядати, створювати, редагувати та видаляти облікові записи менеджерів.	admin

Продовження таблиці 2.2

Ф-16	Менеджери	Адміністратор повинен переглядати індивідуальну статистику кожного менеджера (кількість замовлень, виручка, клієнти).	admin
Ф-17	Статистика	Система повинна формувати зведену аналітику: кількість клієнтів і замовлень, суми, статуси, динаміка за 30 днів, топ-клієнти, топ-товари.	admin, manager

Наведений перелік охоплює 17 функціональних вимог, що повністю відображають реалізований функціонал платформи. Кожна вимога прив'язана до конкретного модуля та рівня доступу, що унеможливлює неоднозначне трактування при реалізації та тестуванні.

Нефункціональні вимоги визначають якісні характеристики системи — безпеку, надійність, зручність використання та підтримуваність. Їх перелік наведено у таблиці 2.3.

Таблиця 2.3

Нефункціональні вимоги до веб-платформи

№	**Категорія**	**Вимога**
НФ-01	Безпека	Паролі користувачів зберігаються виключно у вигляді bcrypt-хешу (salt rounds = 10); відкритий текст паролю ніколи не зберігається і не передається у відповідях API.
НФ-02	Безпека	Усі захищені ендпоінти перевіряють JWT-токен; ендпоінти з adminOnly повертають 403 для ролі manager.
НФ-03	Безпека	Поле password виключається з відповідей API при вибірці користувачів (select('-password')).

Продовження таблиці 2.3

НФ-04	Надійність	Система перевіряє унікальність email користувача та наявність обов'язкових полів (fullName, clientId) до збереження в базу даних.
НФ-05	Надійність	При помилці з'єднання з MongoDB сервер завершує роботу з кодом 1 (process.exit(1)); нормальна операція не можлива без бази даних.
НФ-06	Зручність	Інтерфейс повинен коректно відображатись у сучасних браузерах (Chrome, Firefox, Edge) без додаткового встановлення програм.
НФ-07	Зручність	Система повинна відображати актуальний статус оплати замовлення автоматично, без ручного вибору менеджером.
НФ-08	Підтримуваність	Структура проєкту повинна відповідати патерну MVC: controllers/, models/, routes/ — кожна директорія з чіткою відповідальністю.
НФ-09	Підтримуваність	Конфіденційна конфігурація (URI бази даних, JWT-секрет) зберігається у файлі .env і не включається до репозиторію (.gitignore).
НФ-10	Продуктивність	API повинен відповідати на типові запити (список замовлень, статистика) в межах прийнятного часу при роботі з тестовою базою розміром до 1000 записів.

Нефункціональні вимоги НФ-01–НФ-03 стосуються безпеки: паролі ніколи не зберігаються у відкритому вигляді, JWT перевіряється на кожному захищеному запиті, а поле password виключається із відповідей API. Вимоги НФ-08 та НФ-09 забезпечують підтримуваність коду: чіткий MVC-поділ і зовнішня конфігурація через .env спрощують розвиток системи в майбутньому.

2.3 Проєктування архітектури платформи

Архітектура платформи базується на класичній клієнт-серверній моделі з чітким розмежуванням між трьома рівнями: клієнтський рівень (браузер), серверний рівень (Node.js/Express) та рівень даних (MongoDB). Такий поділ забезпечує незалежність кожного шару та спрощує підтримку системи [9].

Платформа реалізована за трирівневою клієнт-серверною архітектурою, що забезпечує чіткий розподіл відповідальності між компонентами системи. Клієнтський рівень формують статичні HTML-сторінки та JavaScript-модулі, розміщені у директорії /public. Взаємодія з сервером реалізована виключно через асинхронні fetch()-запити із передачею JWT-токена у заголовку Authorization: Bearer. Таким чином клієнтська частина не має прямого доступу до бази даних і не знає деталей серверної реалізації — вона взаємодіє лише з REST API.

Серверний рівень реалізовано на Node.js з використанням Express. Кожен HTTP-запит проходить через послідовний ланцюжок обробки: маршрутизатор (routes/) визначає відповідний контролер, middleware-функції (authMiddleware.js) перевіряють JWT-токен та рівень доступу, після чого управління передається до контролера (controllers/), який формує запит до бази даних. Рівень даних представлений СУБД MongoDB з ODM Mongoose: схеми моделей (User, Client, Order, Product) описують структуру колекцій, а функція populate() автоматично підставляє пов’язані документи при вибірці. Така трирівнева організація забезпечує незалежність кожного шару і спрощує подальший супровід системи.

Клієнтський рівень представлений статичними HTML-сторінками та JavaScript-файлами, що розміщені у директорії /public. Кожна сторінка системи — окремий HTML-файл у /public/pages/ з власним JS-модулем у /public/js/. Список сторінок: login.html, dashboard.html, clients.html, client-detail.html, orders.html, order-detail.html, products.html, managers.html, manager-detail.html, statistics.html. Клієнтський JavaScript взаємодіє з серверним API виключно через асинхронні fetch()-запити з передачею JWT-токена у заголовку Authorization: Bearer <token>. Токен зберігається в localStorage браузера після успішного входу.

Серверний рівень реалізований на Node.js з фреймворком Express. Точка входу — файл server.js, що виконує такі функції: ініціалізація застосунку Express, підключення middleware (express.json для парсингу тіла запиту, express.static для роздачі файлів клієнтської частини), реєстрація шести груп маршрутів, підключення до MongoDB та запуск HTTP-сервера на порту з .env (за замовчуванням 3000). Структура директорій серверної частини відповідає адаптованому патерну MVC.

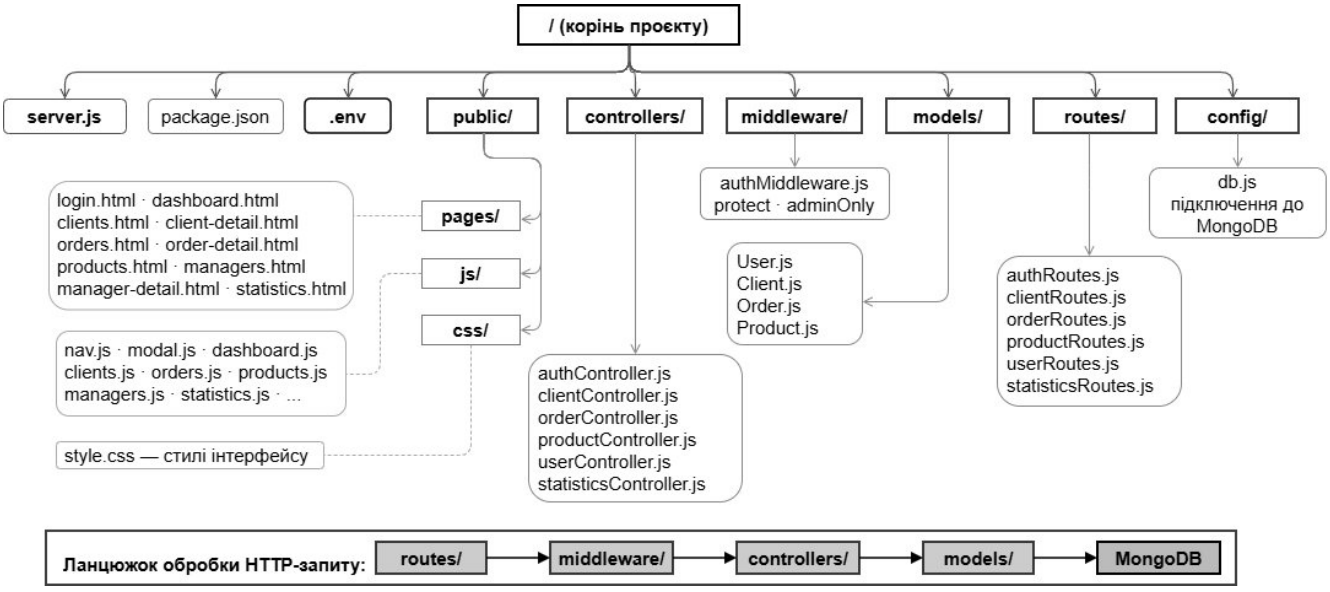


Рис. 2.2 Структура директорій проєкту

Обробка кожного HTTP-запиту відбувається через ланцюжок: маршрут (routes/) → middleware автентифікації (middleware/) → контролер (controllers/) → модель (models/) → MongoDB. Контролери повністю відповідають за бізнес-логіку: формують запит до бази даних через Mongoose, обробляють результат, повертають HTTP-відповідь з відповідним кодом статусу (200, 201, 400, 401, 403, 404, 500).

Рівень даних представлений СУБД MongoDB з ODM Mongoose. База даних розгортається локально (mongodb://127.0.0.1:27017/diplom_crm згідно з .env). Mongoose формує чотири колекції відповідно до визначених схем: users, clients, orders, products. Зв'язки між колекціями реалізовані через ObjectId-посилання та функцію populate(), що підставляє пов'язані документи при вибірці. Наприклад,

при отриманні замовлення автоматично підставляються дані клієнта, кожного товару зі списку та автора запису.

2.4 Моделювання бази даних

База даних складається з чотирьох колекцій: users, clients, orders, products. Зв'язки між ними організовані через ObjectId-посилання в документах і розгортаються за потребою через Mongoose populate(). Жодних таблиць-зв'язок (junction tables) не використовується — завдяки документній моделі MongoDB вкладені дані (масиви товарів, історія статусів) зберігаються безпосередньо в документі замовлення [8].

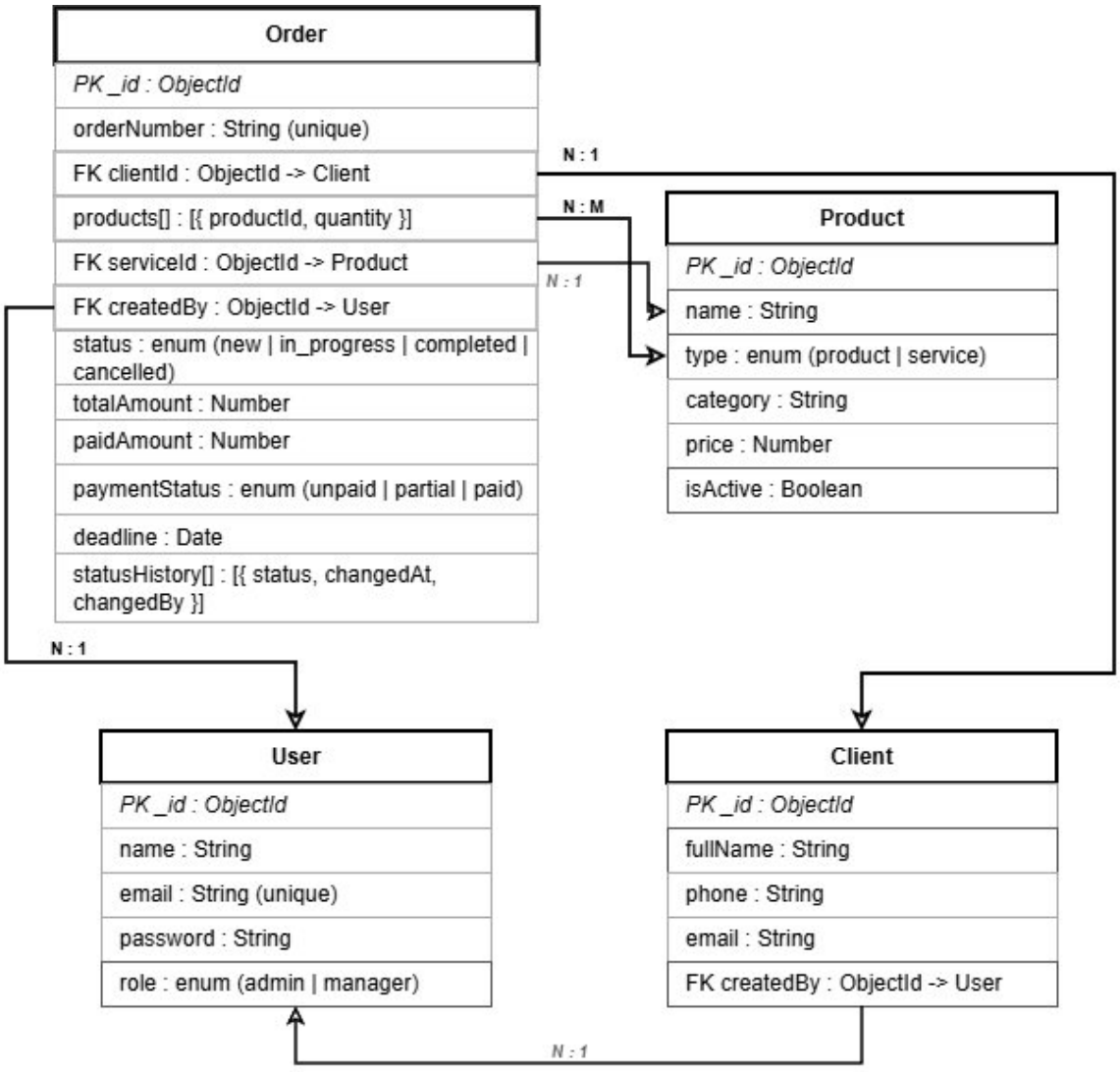


Рис. 2.3 ER-діаграма бази даних веб-платформи

Центральною є колекція orders — більшість зовнішніх посилань спрямовані до неї або з неї. Кожне замовлення посилається на одного клієнта (clientId), одного або кількох товарів через вкладений масив (products[].productId), на послугу (serviceId) та на автора замовлення (createdBy). Окрема колекція для зв'язку замовлень з товарами не потрібна — масив products[] зберігається всередині документа Order разом з кількістю кожного товару.

Модель Order. Найскладніша модель системи. Її ключові поля та призначення описані у таблиці 2.4.

Таблиця 2.4

Опис полів моделі Order

Поле	**Тип**	**За замовч.**	**Опис**
orderNumber	String (unique)	—	Унікальний номер замовлення формату ORD-XXXX, генерується автоматично
clientId	ObjectId → Client	required	Посилання на клієнта, обов'язкове поле
title	String	''	Назва замовлення; якщо порожня і є тільки товари — встановлюється 'Замовлення товарів'
description	String	''	Довільний опис, нотатки по замовленню
products[]	Array of { productId, quantity }	[]	Список товарів із кількістю; productId → ObjectId посилання на Product
serviceId	ObjectId → Product	null	Посилання на послугу з каталогу (тип service)
serviceName	String	''	Назва послуги (зберігається поряд з serviceId для історії)

Продовження таблиці 2.4

createdBy	ObjectId → User	null	Менеджер, що створив замовлення
status	String (enum)	'new'	Поточний статус: new | in_progress | completed | cancelled
totalAmount	Number	0	Загальна сума замовлення (грн)
paidAmount	Number	0	Фактично сплачена сума (грн)
paymentStatus	String (enum)	'unpaid'	Розрахунковий статус оплати: unpaid | partial | paid
deadline	Date	null	Дедлайн виконання; при простроченні замовлення позначається як overdue
statusHistory[]	Array of { status, changedAt, changedBy }	[]	Лог змін статусу: фіксується при кожній зміні поля status
createdAt, updatedAt	Date	auto	Автоматично додаються Mongoose через опцію timestamps: true

Логіка розрахунку paymentStatus реалізована у допоміжній функції calcPaymentStatus(paid, total): якщо paidAmount = 0 — статус 'unpaid'; якщо paidAmount >= totalAmount — 'paid'; інакше — 'partial'. Функція викликається при створенні та оновленні замовлення і завжди повертає актуальний статус без необхідності ручного вибору менеджером.

Поле statusHistory[] є масивом вбудованих документів. При кожній зміні поля status (порівнюється новий і старий status в updateOrder) до масиву додається новий запис з поточним статусом, датою та ідентифікатором менеджера, що вніс

зміну. При першому створенні замовлення до історії автоматично додається початковий запис.

Модель Client. Зберігає контактні дані клієнта та посилання на автора запису. Єдине обов'язкове поле — fullName. Телефон та email є необов'язковими, але використовуються для перевірки дублікатів через окремий ендпоінт /api/clients/check-duplicate. Поля таблиці 2.5.

Таблиця 2.5

Опис полів моделі Client

Поле	**Тип**	**За замовч.**	**Опис**
fullName	String	required	Повне ім'я клієнта, єдине обов'язкове поле
phone	String	''	Номер телефону; використовується для пошуку та перевірки дублікатів
email	String	''	Електронна пошта; використовується для пошуку та перевірки дублікатів
companyName	String	''	Назва компанії (для юридичних осіб)
address	String	''	Адреса об'єкта / доставки
notes	String	''	Довільні нотатки менеджера про клієнта
createdBy	ObjectId → User	null	Менеджер, що додав клієнта до бази
createdAt, updatedAt	Date	auto	Автоматично додаються Mongoose (timestamps: true)

Модель Product описує єдиний довідник для товарів і послуг. Поле type з enum-обмеженням (product | service) дозволяє зберігати обидва типи в одній колекції та фільтрувати за потребою (?type=product або ?type=service). Поле isActive дозволяє деактивувати позицію без видалення — вона не пропонується

при створенні замовлення, але зберігається в системі для історії. Обов'язкові поля: name, type, category.

Модель User зберігає облікові записи всіх користувачів системи. Поле role (enum: admin | manager) визначає рівень доступу і включається до JWT-токена при вході. Поле password зберігається виключно у вигляді bcrypt-хешу і виключається з API-відповідей через .select('-password').

2.5 Проєктування REST API

REST API платформи охоплює шість груп ресурсів: auth, clients, orders, products, users, statistics. Усі ендпоінти (крім /api/auth/login) захищені JWT-автентифікацією через middleware protect. Ендпоінти ресурсу users додатково захищені middleware adminOnly, що перевіряє поле role у декодованому токені [3].

Механізм захисту API працює таким чином: клієнт надсилає HTTP-запит із заголовком Authorization: Bearer <jwt_token>. Middleware protect витягує токен, перевіряє підпис через jwt.verify() з JWT_SECRET, і при успіху записує декодований payload (id, name, role) у req.user. Якщо токен відсутній або недійсний — повертається HTTP 401. Якщо потрібен adminOnly і роль не admin — HTTP 403.

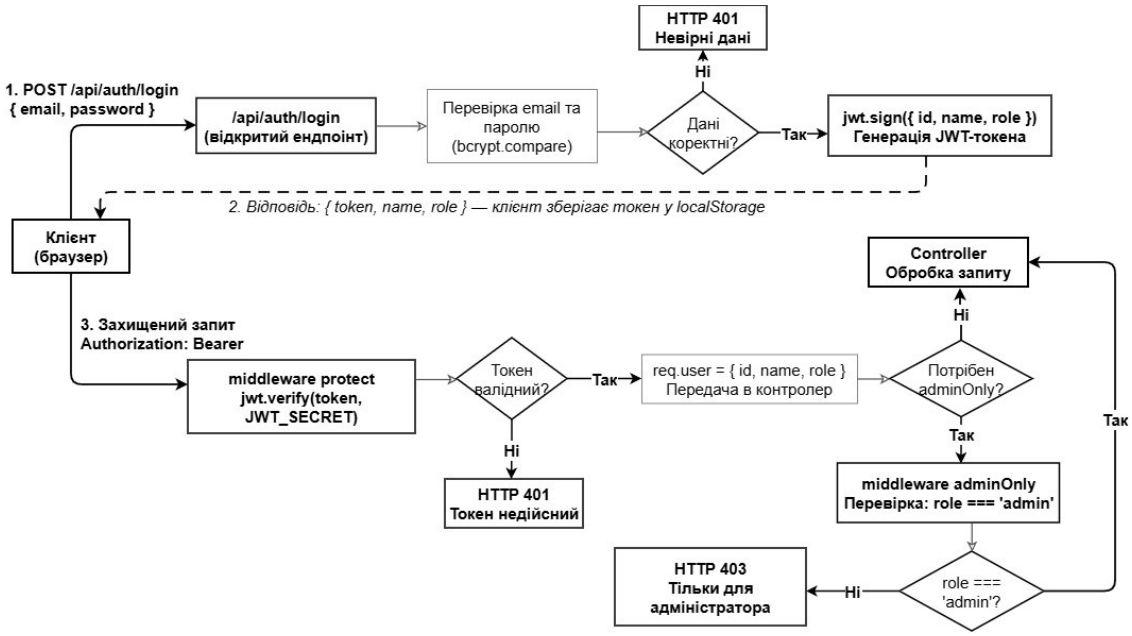


Рис. 2.4 Схема JWT-автентифікації та захисту ендпоінтів

Повна специфікація ендпоінтів API наведена у таблиці 2.6. Рядки згруповані за групами ресурсів: auth, clients, orders, products, users, statistics.

Таблиця 2.6

Специфікація REST API веб-платформи

Метод	**Ендпоінт**	**Захист**	**Контролер**	**Призначення**
POST	/api/auth/login	Відкритий	login	Вхід в систему, повертає JWT
GET	/api/clients	protect	getClients	Список клієнтів (пошук ?search=)
GET	/api/clients/check-duplicate	protect	checkDuplicate	Перевірка дубл. за phone/email
GET	/api/clients/:id	protect	getClientById	Картка клієнта за ID
GET	/api/clients/:id/orders	protect	getClientOrders	Замовлення конкретного клієнта
POST	/api/clients	protect	createClient	Створення нового клієнта
PUT	/api/clients/:id	protect	updateClient	Оновлення даних клієнта
DELETE	/api/clients/:id	protect	deleteClient	Видалення клієнта
GET	/api/orders	protect	getOrders	Список замовлень (фільтри, пошук)
GET	/api/orders/:id	protect	getOrderById	Деталі замовлення за ID

Продовження таблиці 2.6

POST	/api/orders	protect	createOrder	Створення замовлення
PUT	/api/orders/:id	protect	updateOrder	Оновлення замовлення (з фіксацією зміни статусу)
DELETE	/api/orders/:id	protect	deleteOrder	Видалення замовлення
GET	/api/products	protect	getProducts	Каталог (фільтри type, category, active)
GET	/api/products/:id	protect	getProductById	Позиція каталогу за ID
POST	/api/products	protect	createProduct	Додавання товару/послуги
PUT	/api/products/:id	protect	updateProduct	Оновлення позиції каталогу
DELETE	/api/products/:id	protect	deleteProduct	Видалення позиції
PATCH	/api/products/:id/toggle	protect	toggleActive	Перемикання isActive (активна / деактивована)
GET	/api/users	protect + adminOnly	getUsers	Список менеджерів
GET	/api/users/:id	protect + adminOnly	getUserById	Профіль менеджера за ID

Продовження таблиці 2.6

GET	/api/users/:id/statistics	protect + adminOnly	getUserStatistics	Статистика конкретного менеджера
POST	/api/users	protect + adminOnly	createUser	Додавання менеджера / адміна
PUT	/api/users/:id	protect + adminOnly	updateUser	Редагування облікового запису
DELETE	/api/users/:id	protect + adminOnly	deleteUser	Видалення менеджера (не себе)
GET	/api/statistics	protect	getStatistics	Зведена аналітика підприємства

Важливою особливістю маршрутизації є порядок реєстрації специфічних маршрутів перед параметричними у межах одного ресурсу. Наприклад, у clientRoutes.js маршрут GET /api/clients/check-duplicate реєструється раніше, ніж GET /api/clients/:id, — інакше Express інтерпретував би рядок 'check-duplicate' як значення параметра :id і викликав неправильний контролер. Аналогічно GET /api/users/:id/statistics реєструється до GET /api/users/:id.

Усі API-відповіді повертаються у форматі JSON. При успішному створенні ресурсу повертається HTTP 201 Created з тілом нового документа. При помилках клієнта (відсутнє обов'язкове поле, некоректний ID, дублікат) — HTTP 400 або 404 з описовим повідомленням. При серверних помилках — HTTP 500.

При отриманні замовлень (getOrders, getOrderById) виконується ланцюжок populate(): підставляються повні дані клієнта (fullName, phone), кожного товару зі списку products[] (name, price, type), послуги (serviceId), та автора (createdBy з

name та email). Це дозволяє клієнтській частині відображати повні дані без додаткових запитів.

У другому розділі виконано аналіз предметної області та повний цикл проєктування системи. Основні результати:

– Описано типовий бізнес-процес обробки замовлення в підприємстві технічних послуг: від звернення клієнта до завершення оплати. Визначено п'ять основних кроків процесу та зафіксовано їх у вигляді діаграми з переходами між статусами.

– Сформульовано 17 функціональних вимог у розрізі шести модулів системи та 10 нефункціональних вимог, що охоплюють безпеку, надійність, зручність і підтримуваність. Усі вимоги прив'язані до конкретних технічних рішень, реалізованих у коді.

– Спроєктовано трирівневу архітектуру: клієнтський рівень (статичні HTML + Vanilla JS з fetch), серверний рівень (Node.js/Express, MVC-структура), рівень даних (MongoDB/Mongoose). Кожен рівень незалежний і взаємодіє з іншими через чітко визначені інтерфейси.

– Спроєктовано чотири Mongoose-моделі: User, Client, Order, Product. Модель Order є центральною і зберігає вкладені масиви products[] і statusHistory[] без необхідності окремих колекцій-зв'язок. Визначено зв'язки між колекціями через ObjectId-посилання та populate().

– Специфіковано повний REST API з 27 ендпоінтами у шести ресурсних групах. Описано дворівневий механізм захисту (protect + adminOnly), порядок реєстрації маршрутів та поведінку populate() при вибірці складених документів.

Результати проєктування, зафіксовані у вигляді таблиць вимог, ER-діаграми та специфікації API, є вичерпною основою для реалізації всіх модулів системи, що описується в наступному розділі.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ

3.1 Структура проєкту та організація серверної частини

Реалізація платформи розпочинається з організації файлової структури проєкту, яка безпосередньо відображає архітектурний патерн MVC, описаний у розділі 2. Файли серверної частини розміщені у кореневій директорії, клієнтська частина зосереджена у /public.

Файлова структура проєкту безпосередньо відображає обраний архітектурний патерн MVC. Кореневий рівень містить три елементи: файл server.js як точку входу застосунку, файл package.json з описом залежностей та файл .env зі змінними середовища (PORT, MONGO_URI, JWT_SECRET), виключений з репозиторію через .gitignore. Серверна логіка розподілена між чотирма директоріями: config/ (підключення до MongoDB через db.js), controllers/ (шість контролерів бізнес-логіки), middleware/ (authMiddleware.js з функціями protect та adminOnly) та models/ (чотири Mongoose-схеми: User, Client, Order, Product). Маршрутизація HTTP-запитів зосереджена у директорії routes/, що містить шість файлів відповідно до груп ресурсів API.

Клієнтська частина повністю розміщена у директорії public/, яка поділена на три піддиректорії: pages/ (десять HTML-сторінок інтерфейсу), js/ (дванадцять JavaScript-модулів, по одному на кожну сторінку, та спільні модулі nav.js і modal.js) і css/ (єдиний файл стилів style.css). Директорія scripts/ містить допоміжний скрипт seedDemoData.js для генерації тестових даних, що запускається командою npm run seed:demo і створює набір користувачів, клієнтів, товарів та замовлень для перевірки коректності роботи системи.

Точкою входу застосунку є файл server.js. Він виконує п'ять послідовних дій: завантаження конфігурації з .env через dotenv.config(), ініціалізація підключення до MongoDB через connectDB(), створення екземпляра Express, реєстрація middleware та маршрутів, запуск HTTP-сервера на зазначеному порту.

Фрагмент коду 3.1 — server.js: точка входу застосунку

```
        const express   = require('express');
        const dotenv    = require('dotenv');
        const path      = require('path');
        const connectDB = require('./config/db');

        dotenv.config();
        connectDB();

        const app = express();
        app.use(express.json());
        app.use(express.static(path.join(__dirname, 'public')));

        app.use('/api/auth',       require('./routes/authRoutes'));
        app.use('/api/clients',    require('./routes/clientRoutes'));
        app.use('/api/orders',     require('./routes/orderRoutes'));
        app.use('/api/statistics', require('./routes/statisticsRoutes'));
        app.use('/api/products',   require('./routes/productRoutes'));
        app.use('/api/users',      require('./routes/userRoutes'));

        app.get('/', (req, res) => res.redirect('/pages/login.html'));
        const PORT = process.env.PORT || 3000;
        app.listen(PORT, () => console.log(`Сервер запущено на
http://localhost:${PORT}`));
```

Middleware express.json() забезпечує автоматичний парсинг JSON-тіла запитів у req.body. Middleware express.static() роздає HTML/CSS/JS клієнтської частини з директорії /public, тому клієнт отримує інтерфейс від того самого сервера, що обслуговує API.

Конфігурація підключення до MongoDB винесена у config/db.js. Функція connectDB() встановлює з'єднання через mongoose.connect() з URI зі змінної

MONGO_URI. При помилці підключення сервер завершує роботу з кодом 1 — без бази даних жодна операція з даними неможлива.

Фрагмент коду 3.2 — config/db.js: підключення до MongoDB

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('MongoDB connected successfully');
  } catch (error) {
    console.error('MongoDB connection error:', error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Файл .env (виключений з репозиторію через .gitignore) містить три змінні: PORT (порт сервера, за замовчуванням 3000), MONGO_URI (URI підключення до MongoDB; у поточній конфігурації — mongodb://127.0.0.1:27017/diplom_crm) та JWT_SECRET (секретний ключ для підпису токенів).

3.2 Реалізація модуля автентифікації та управління ролями

Модуль автентифікації складається з двох компонентів: контролера authController.js з логікою входу та middleware-функцій у authMiddleware.js для захисту маршрутів.

Логіка входу (login). Функція login виконує три кроки: знаходить користувача за email (User.findOne({ email })), перевіряє пароль через bcrypt.compare() і при успіху генерує JWT-токен через jwt.sign() з payload { id,

name, role } та терміном дії 8 годин. Роль включається у токен і повертається клієнту для управління видимістю елементів інтерфейсу [12].

Фрагмент коду 3.3 — authController.js: функція login

```
const login = async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(401).json({ message: 'Невірний email або
пароль' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(401).json({ message: 'Невірний email або
пароль' });

    const token = jwt.sign(
      { id: user._id, name: user.name, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '8h' }
    );
    res.json({ token, name: user.name, role: user.role });
  } catch (error) {
    res.status(500).json({ message: 'Помилка сервера' });
  }
};
```

Middleware protect та adminOnly. Функція protect витягує JWT з заголовку Authorization (формат «Bearer <token>»), верифікує підпис через jwt.verify() і записує декодований payload у req.user. При невалідному токені — HTTP 401. Функція adminOnly перевіряє req.user.role: якщо не 'admin' — HTTP 403 [6].

Фрагмент коду 3.4 — authMiddleware.js: protect та adminOnly

```
const protect = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Немає токена, доступ заборонено' });
  }
  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded; // { id, name, role }
    next();
  } catch (error) {
    return res.status(401).json({ message: 'Токен недійсний' });
  }
};

const adminOnly = (req, res, next) => {
  if (req.user && req.user.role === 'admin') return next();
  return res.status(403).json({ message: 'Доступ заборонено. Тільки для адміністратора.' });
};
```

На клієнтській стороні після успішного входу токен і роль зберігаються у localStorage. Модуль nav.js, що завантажується першим на кожній сторінці, зчитує роль і виконує дві операції: відображає бейдж ролі у навігаційному рядку та приховує елементи з класом .admin-nav-link для менеджерів. Аватар формується з ініціалів імені через розбиття рядка за пробілами. При виході токен видаляється і відбувається редирект на login.html.

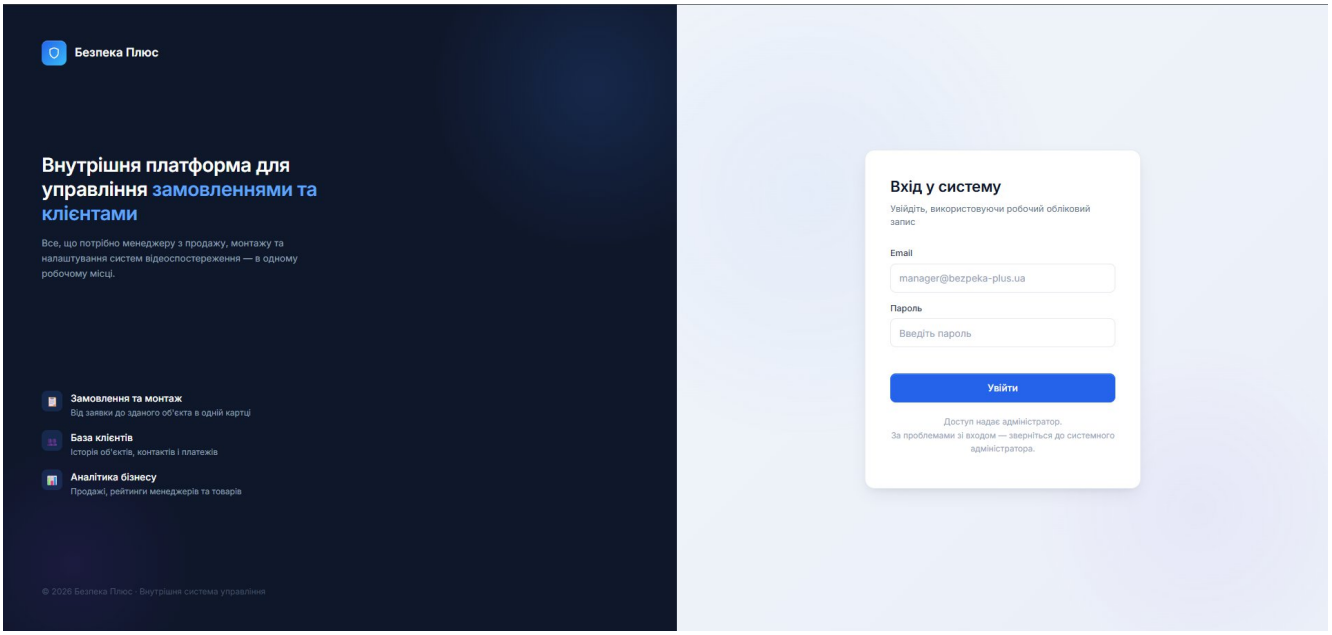


Рис. 3.1 Сторінка авторизації платформи

3.3 Реалізація CRUD-модулів: клієнти, замовлення, товари та послуги

Модуль клієнтів реалізований у clientController.js і охоплює сім маршрутів.

Функція getClients підтримує пошук через параметр ?search=: MongoDB-запит з оператором $or перевіряє поля fullName, phone та email через регулярний вираз. Populate підставляє дані менеджера, що додав клієнта. Результати сортуються за датою створення у спадному порядку.

Функція checkDuplicate (GET /api/clients/check-duplicate) приймає параметри phone та/або email і шукає існуючий збіг через Client.findOne({ $or: conditions }). При знайденому дублікаті клієнтський JS відображає модальне вікно підтвердження (confirmDuplicate з modal.js) перед збереженням. Цей ендпоінт реєструється першим у маршрутах, щоб Express не прийняв рядок 'check-duplicate' за значення параметра :id.

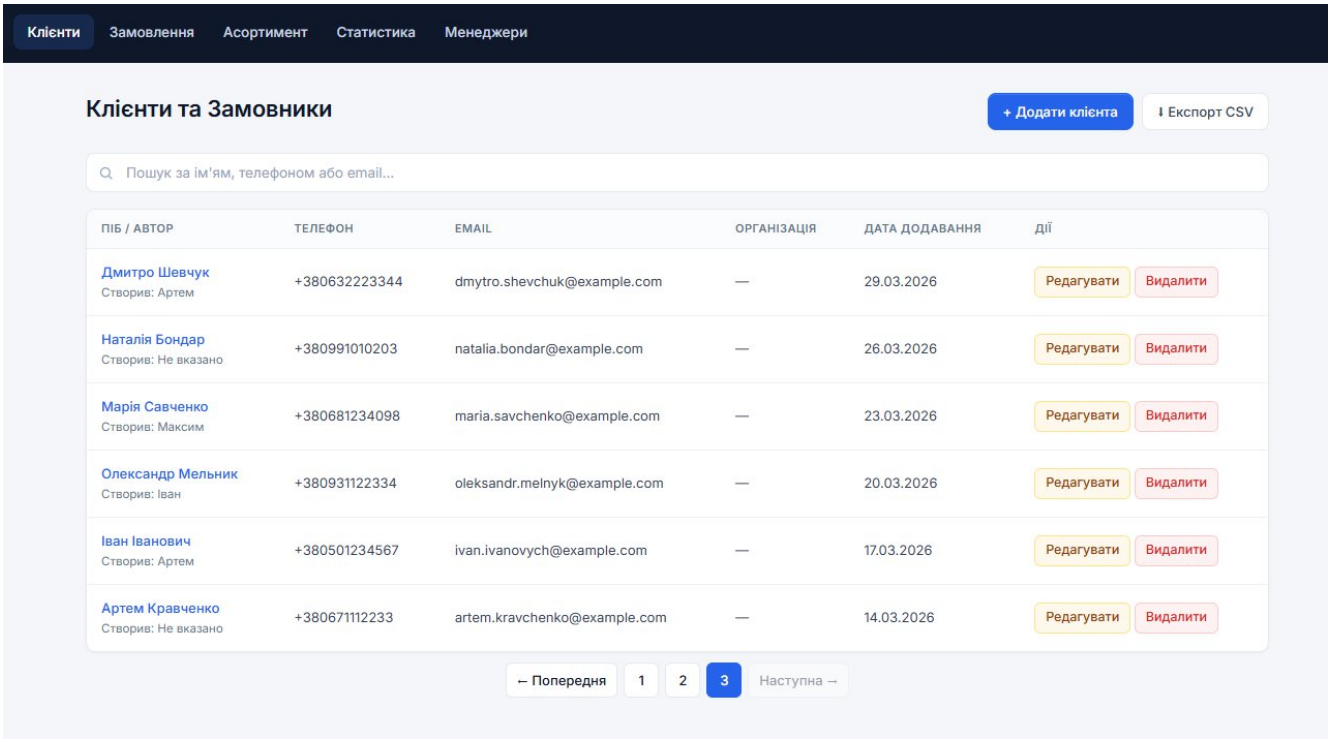


Рис. 3.2 Сторінка управління клієнтами

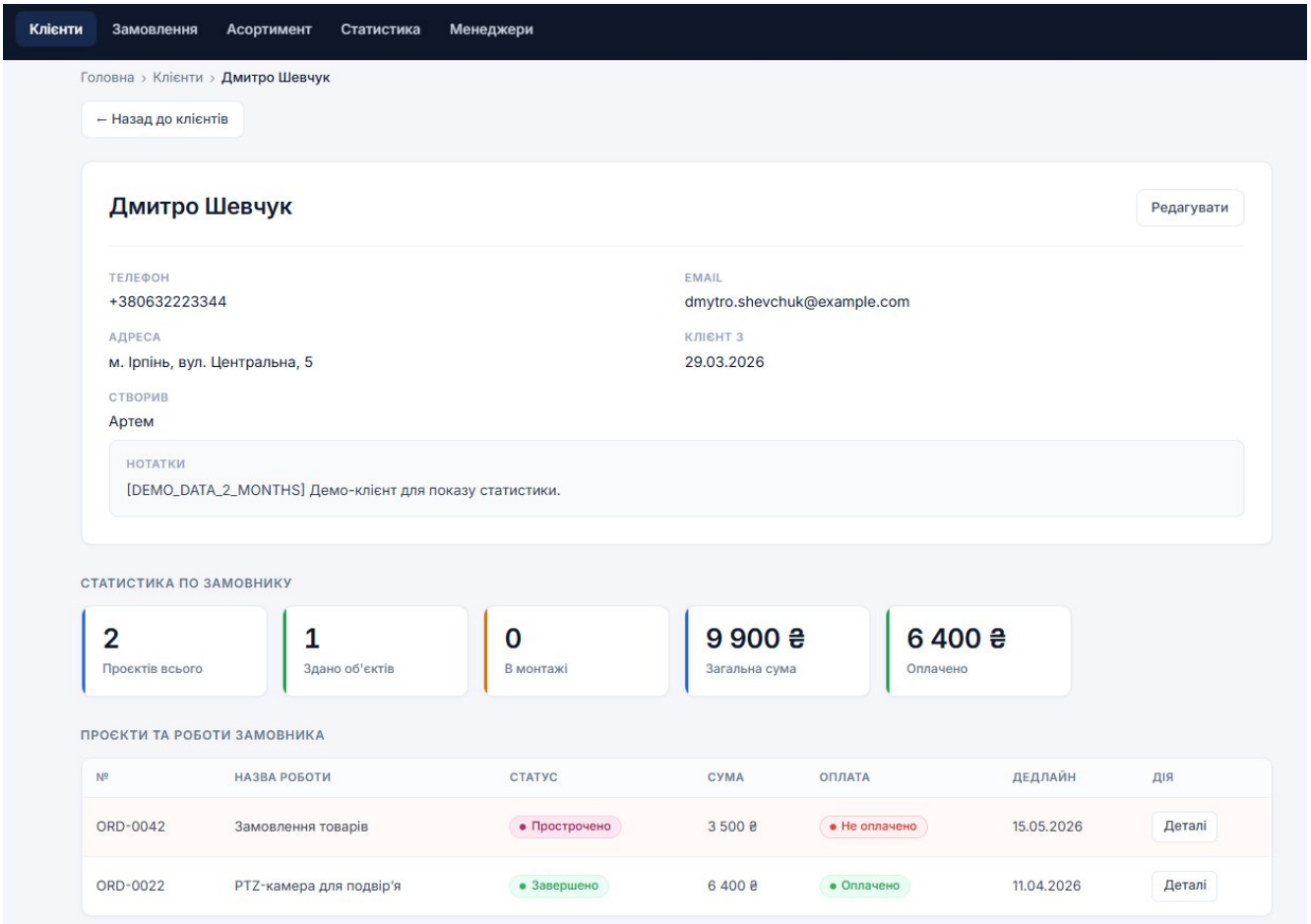


Рис. 3.3 Картка клієнта з історією замовлень

Модуль замовлень є найбільш функціонально насиченим і реалізований у orderController.js. Три ключових аспекти: автоматична нумерація, підтримка товарів і послуг, історія статусів.

Автоматична нумерація реалізована у функції generateOrderNumber(). Вона вибирає всі замовлення з числовим форматом ORD-XXXX, знаходить максимальний номер і повертає наступний із нульовим вирівнюванням до 4 цифр. Це забезпечує передбачувану послідовну нумерацію.

Фрагмент коду 3.5 — orderController.js: функція generateOrderNumber

```
async function generateOrderNumber() {
  const orders = await Order.find(
    { orderNumber: { $regex: /^ORD-\d+$/ } }
  ).select('orderNumber');

  const maxNumber = orders.reduce((max, order) => {
    const num = parseInt(order.orderNumber.replace('ORD-', ''), 10);
    return Number.isNaN(num) ? max : Math.max(max, num);
  }, 0);

  return 'ORD-' + String(maxNumber + 1).padStart(4, '0');
}
```

Підтримка товарів і послуг в одному замовленні реалізована через функцію prepareProducts(body). Вона нормалізує вхідні дані у масив { productId, quantity }[]. Паралельно обробляється поле serviceId для послуги. Таким чином, одне замовлення може містити кілька товарів різної кількості та одну послугу.

Історія зміни статусів реалізована в updateOrder. Функція отримує старий order, порівнює старий і новий status і лише при їх відмінності додає новий запис до statusHistory[] через оператор MongoDB $push зі статусом, датою та ID менеджера. Якщо статус не змінився — історія не оновлюється.

Розрахунок paymentStatus виконується функцією calcPaymentStatus(paid, total) при кожному створенні/оновленні: paid = 0 → 'unpaid'; paid >= total → 'paid'; інакше → 'partial'. Менеджер вводить лише суми — статус визначається автоматично.

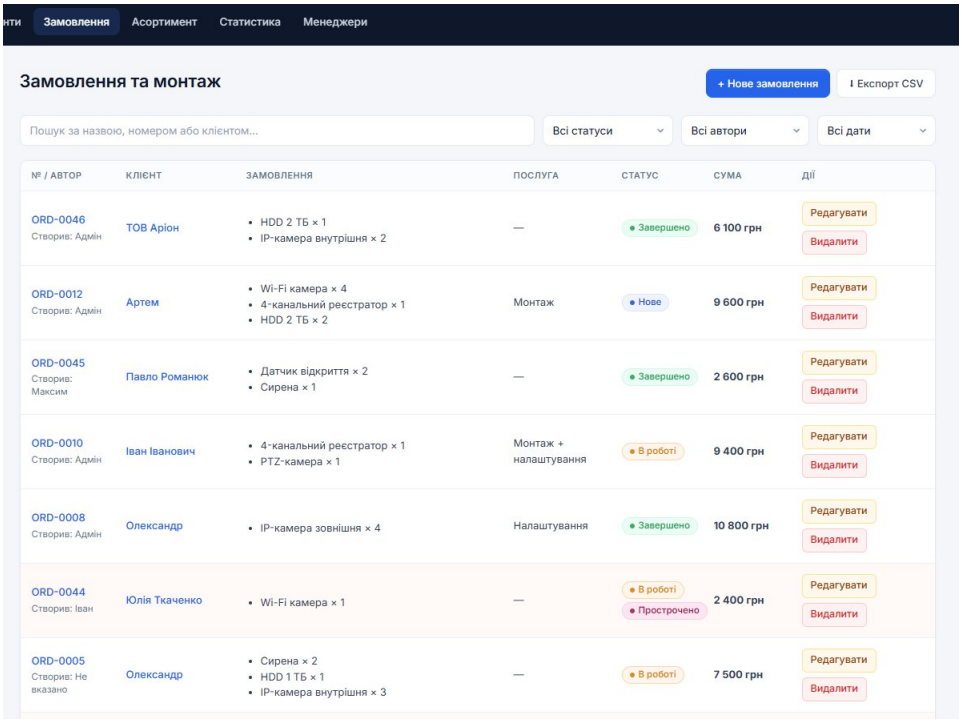


Рис. 3.4 Сторінка управління замовленнями

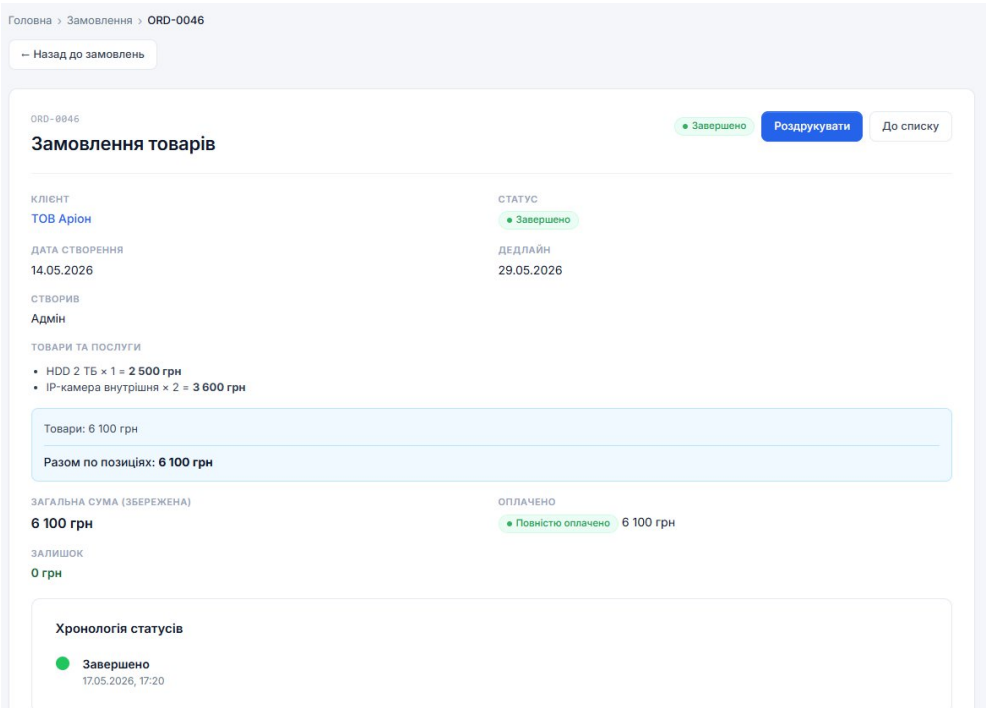


Рис. 3.5 Картка замовлення з історією статусів

Модуль товарів та послуг реалізований у productController.js з єдиною колекцією Product для обох типів сутностей. Фільтрація через query-параметри: ?type=product, ?type=service, ?category=..., ?active=true. Функція toggleActive (PATCH /api/products/:id/toggle) перемикає isActive без видалення запису, що дозволяє зберегти історичні дані в замовленнях.

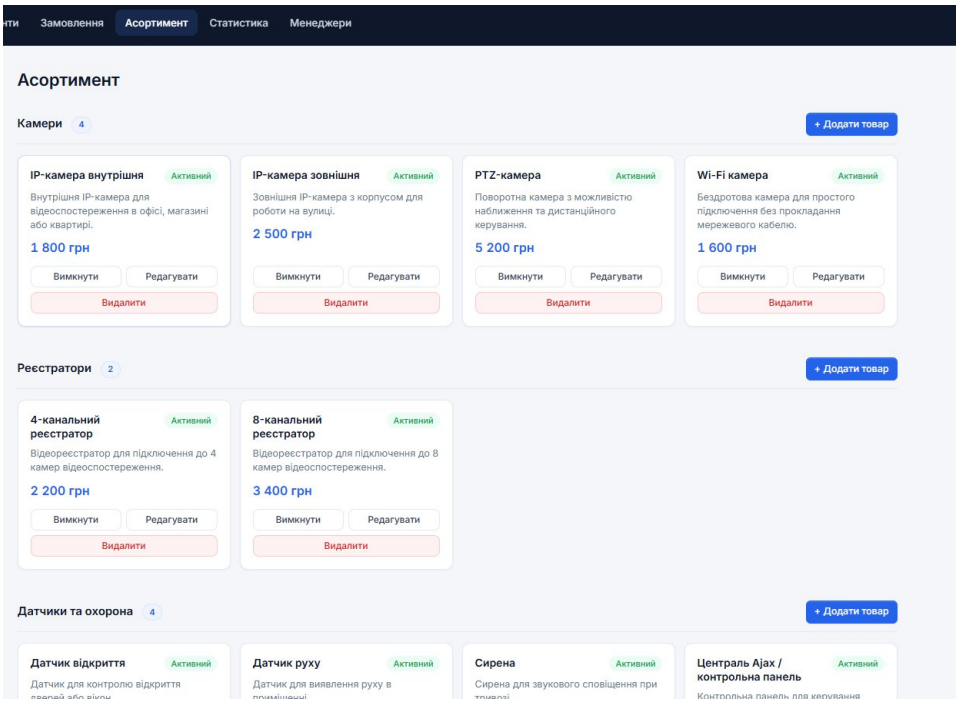


Рис. 3.6 Сторінка управління каталогом товарів і послуг

Модуль менеджерів (userController.js, доступний лише адміну) реалізує повний CRUD для облікових записів. createUser хешує пароль через bcrypt.hash(password, 10) та валідує email. updateUser оновлює пароль лише при передачі непорожнього значення. deleteUser порівнює req.params.id з req.user.id і повертає HTTP 400 при збігу — захист від самовидалення. Функція getUserStatistics повертає агреговані показники менеджера: кількість замовлень, суми, кількість доданих клієнтів та останні 5 замовлень.

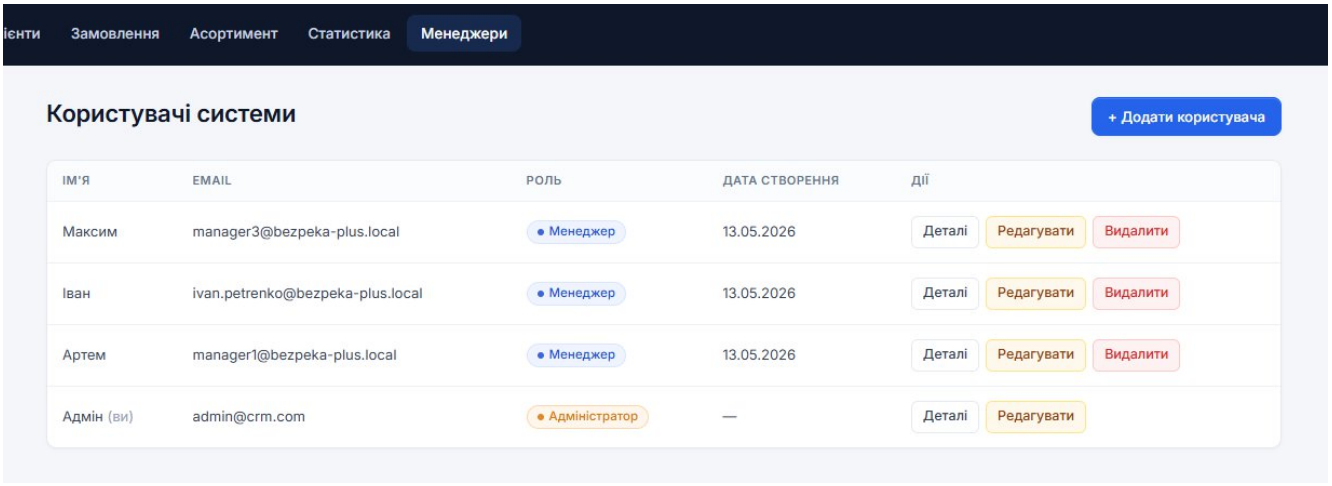


Рис. 3.7 Сторінка управління обліковими записами менеджерів

3.4 Модуль статистики та аналітики

Модуль реалізований у statisticsController.js і є функціонально найскладнішим компонентом системи. Весь розрахунок виконується одним запитом GET /api/statistics: завантажуються усі замовлення з повним populate() (клієнти, товари, послуги, автори), після чого дані обробляються в пам'яті через JavaScript Map-структури [11].

Контролер обчислює такі показники:

– зведені лічильники: кількість клієнтів, замовлень, активних товарів і послуг, деактивованих позицій;

– розподіл за статусами замовлень: new, in_progress, completed, cancelled;

– фінансові показники: totalAmount, paidAmount, debtAmount (різниця між сумою і оплатою по незасокрачованих замовленнях), середній чек;

– проблемні замовлення: прострочені (deadline < now і статус не completed/cancelled), неоплачені, частково оплачені;

– динаміка за 30 днів: масиви ordersByDay[] (кількість) та salesByDay[] (суми) для кожного дня;

– показники поточного дня: кількість і суми замовлень, розбивка по менеджерах;

– рейтинги: топ-5 клієнтів за сумою, топ-5 товарів за кількістю, топ-5 послуг за частотою, топ-3 менеджери за сумою та кількістю;

– дані для діаграм: statusChart[] (розподіл за статусами), paymentChart[] (оплачено / борг).

Фрагмент коду 3.6 — statisticsController.js: обробка замовлень і накопичення показників

```
orders.forEach(order => {
  const total = Number(order.totalAmount) || 0;
  const paid  = Number(order.paidAmount)  || 0;

  totalAmount += total;
  paidAmount  += paid;
  statusCounts[order.status]++;

  if (order.paymentStatus === 'unpaid')  unpaidOrders++;
  if (order.paymentStatus === 'partial') partialOrders++;
  if (order.status !== 'cancelled') debtAmount += Math.max(total - paid, 0);

  // Динаміка за 30 днів
  if (new Date(order.createdAt) >= last30Date) {
    const dayKey = getDateKey(order.createdAt);
    dailyMap.set(dayKey, (dailyMap.get(dayKey) || 0) + 1);
  }

  // Топ-клієнти
  if (order.clientId?.fullName) {
    const c = clientMap.get(order.clientId.fullName) ||
              { name: order.clientId.fullName, totalAmount: 0, ordersCount: 0 };
    c.totalAmount += total; c.ordersCount++;
    clientMap.set(order.clientId.fullName, c);
```

```
    }
});
```

Клієнтська сторінка statistics.js отримує дані одним fetch-запитом і будує нативний SVG-графік динаміки за 30 днів без сторонніх бібліотек. Графік відображає дві polyline-лінії: кількість замовлень (ліва вісь) та суму (права вісь). Сторінка dashboard.html також отримує дані статистики та відображає зведений огляд і таблицю «проблемних» замовлень.

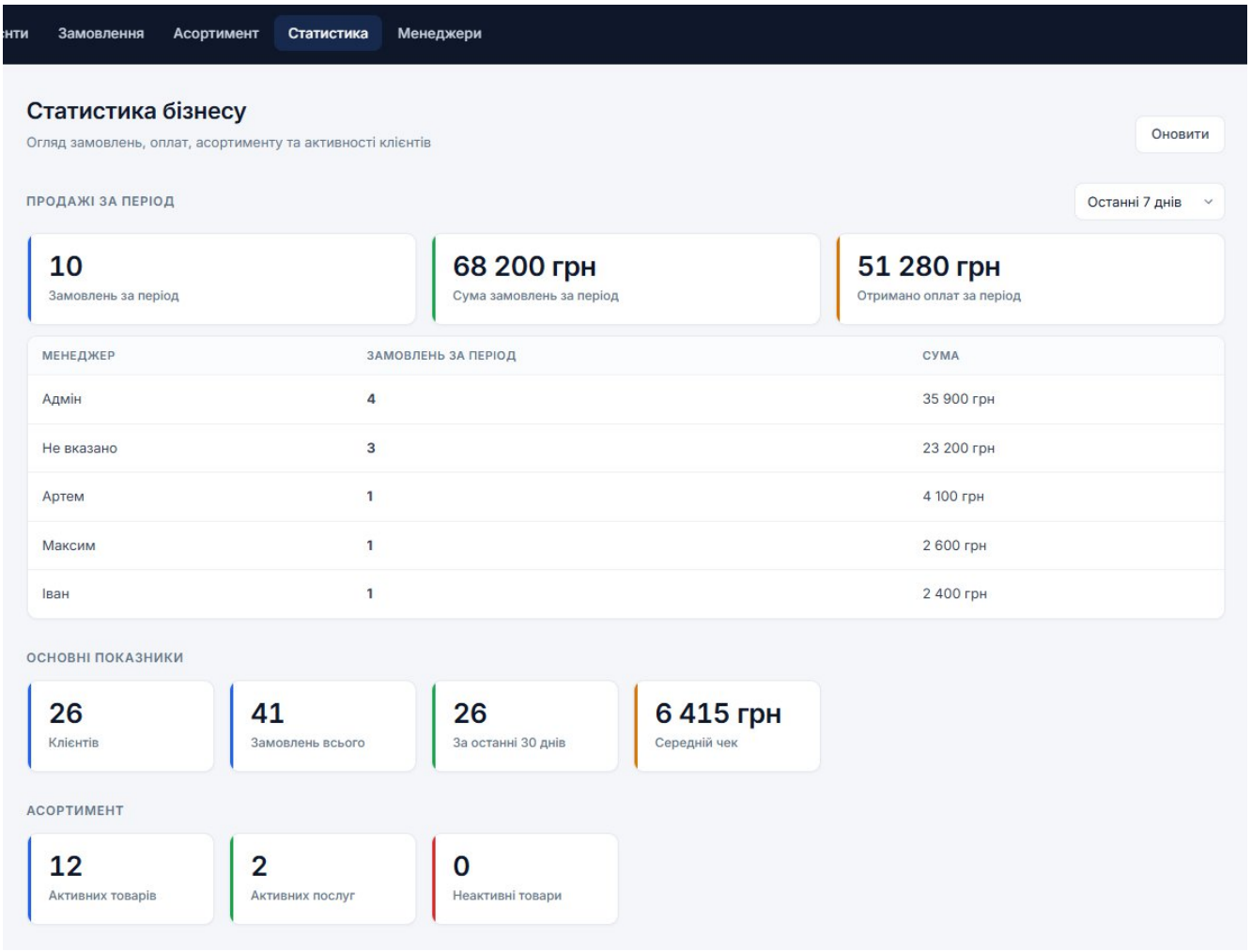


Рис. 3.8 Панель статистики та аналітики

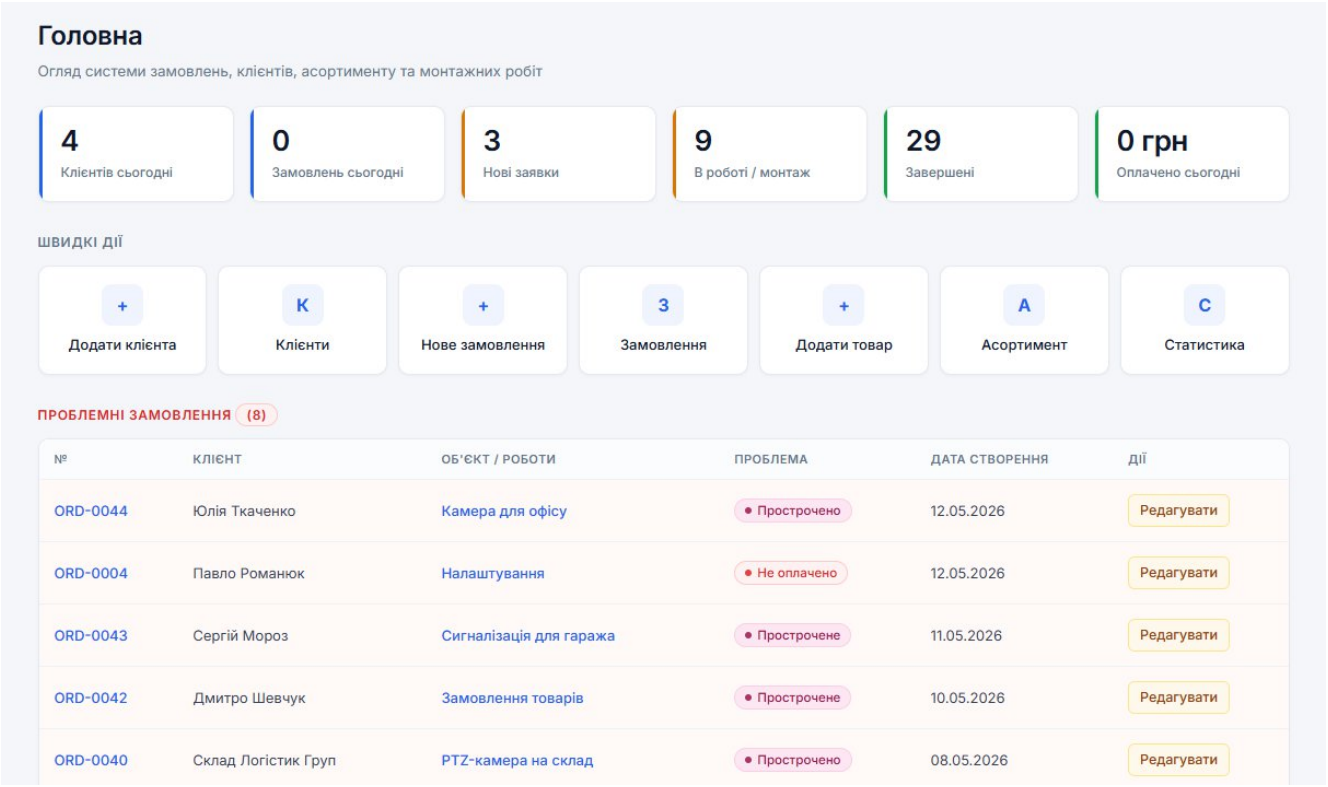


Рис. 3.9 Головна панель з оперативними показниками

3.5 Клієнтська частина: інтерфейс та клієнтський JavaScript

Клієнтська частина побудована без JavaScript-фреймворків — виключно на нативних Web API браузера. Система складається з 10 HTML-сторінок у /public/pages/ та 12 JS-модулів у /public/js/ за принципом «один модуль — одна сторінка». Стилізація зосереджена у /public/css/style.css.

Комунікація з API організована через Fetch API. Кожен JS-модуль визначає локальну функцію authHeader(), що повертає заголовки з поточним JWT-токеном:

Фрагмент коду 3.7 — clients.js: функція authHeader та fetch-запит із JWT

```
function authHeader() {
  return {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`
  };
}
```

```
async function loadClients(search = '') {
  const url = search
    ? `/api/clients?search=${encodeURIComponent(search)}`
    : '/api/clients';
  const res     = await fetch(url, { headers: authHeader() });
  const clients = await res.json();
  renderClients(clients);
}
```

Модуль nav.js підключається першим на кожній сторінці і виконує три дії: відображає бейдж ролі (Адмін / Менеджер) у навбарі; приховує елементи .admin-nav-link для менеджерів; будує аватар з ініціалів імені. Це єдина точка управління видимістю рольових елементів навігації.

Модуль modal.js реалізує єдине повторно використовуване модальне вікно через інжекцію HTML у body при першому виклику (lazy init). Надає три публічні функції: showConfirmModal() — базова функція, що повертає Promise<boolean>; confirmDelete(label) — сценарій видалення із застереженням; confirmDuplicate(clientInfo) — попередження про дублікат клієнта. Підхід замінює стандартні браузерні dialog-вікна і зберігає стилістичну єдність.

Пагінація реалізована клієнтсько: всі записи завантажуються одним запитом і зберігаються у масиві (allClients, allOrders). При переключенні сторінки функція renderClientsPage() або renderOrdersPage() обчислює потрібний зріз масиву і рендерить у таблицю DOM. Клієнтська фільтрація замовлень за статусом також здійснюється без додаткових запитів до API.

SVG-графіки на dashboard.html і statistics.html будуються нативно: функція renderOrdersChart() формує рядок SVG-розмітки на основі масивів ordersByDay[] і salesByDay[], отриманих від /api/statistics, і присвоює його до innerHTML контейнера. Дві polyline-лінії на єдиному SVG-полотні відображають кількість і суму замовлень з двома осями масштабу.

3.6 Тестування та результати роботи системи

Тестування проводилося методом функціонального тестування «чорного ящика»: перевірялася відповідність поведінки системи сформульованим вимогам без аналізу внутрішньої реалізації. Для наповнення тестової бази використовувався скрипт seedDemoData.js (команда npm run seed:demo), що автоматично створює 4 тестових користувачів (1 адмін + 3 менеджери), 14 позицій каталогу (12 товарів і 2 послуги) та набір клієнтів і замовлень за 2 місяці з різними статусами та сумами.

Тестування проводилося у двох режимах. У режимі браузера перевірялися інтерфейс, поведінка форм, навігація між сторінками, динаміка графіків та відображення бейджів. У режимі прямих HTTP-запитів (браузерні DevTools → Network) перевірялися ендпоінти API: коди відповідей, структура JSON, коректність бізнес-логіки.

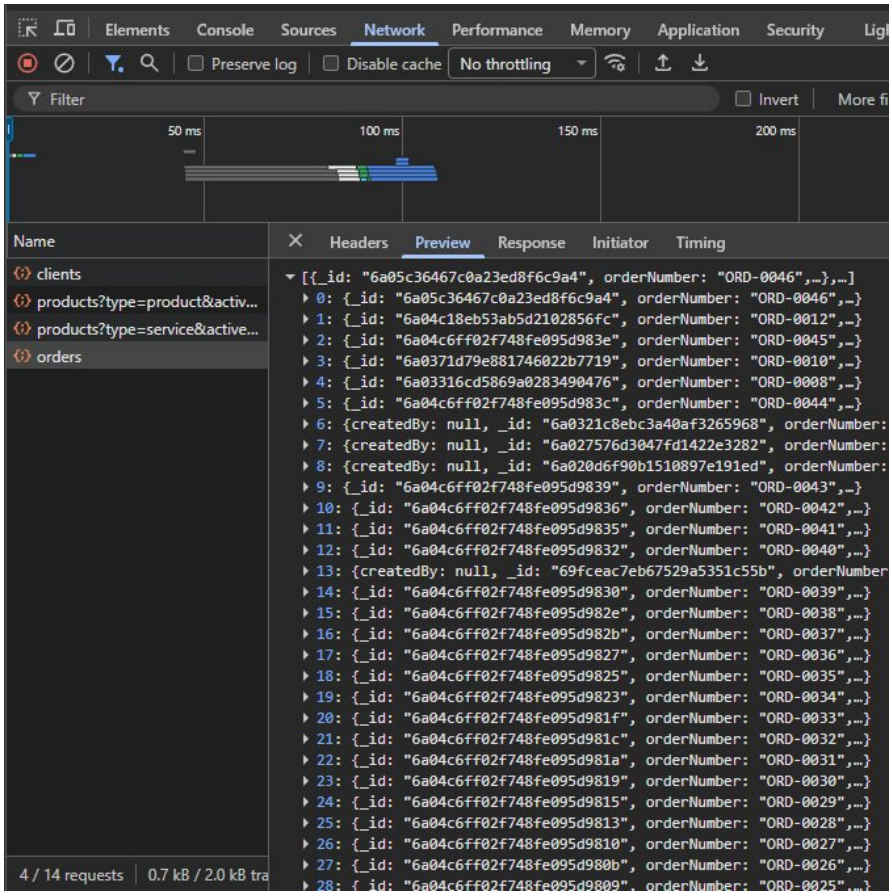


Рис. 3.10 Результат запиту до API у DevTools браузера

Таблиця 3.1

Результати функціонального тестування системи

№	Тестовий сценарій	Вхідні дані / дія	Очікуваний результат	Рез.
Т-01	Вхід з коректними даними	Коректний email та пароль	HTTP 200, JWT у відповіді, редирект на dashboard.html	✓
Т-02	Вхід з невірним паролем	Коректний email, невірний пароль	HTTP 401, «Невірний email або пароль»	✓
Т-03	Запит без токена до захищеного ендпоінту	GET /api/clients без заголовку Authorization	HTTP 401, «Немає токена, доступ заборонено»	✓
Т-04	Доступ менеджера до adminOnly маршруту	GET /api/users з токеном менеджера	HTTP 403, «Доступ заборонено. Тільки для адміністратора»	✓
Т-05	Перевірка дублікату клієнта	GET /api/clients/check-duplicate?phone=існуючий	HTTP 200, { duplicate: true, client: {...} }	✓
Т-06	Створення клієнта без обов'язкового поля	POST /api/clients без поля fullName	HTTP 400, «Поле ПІБ обов'язкове»	✓
Т-07	Створення клієнта з коректними даними	POST /api/clients з fullName, phone, email	HTTP 201, об'єкт нового клієнта з _id та createdAt	✓
Т-08	Автоматична нумерація замовлення	POST /api/orders (перше замовлення в БД)	HTTP 201, orderNumber = «ORD-0001»	✓

Продовження таблиці 3.1

Т-09	Фіксація історії статусу при зміні	PUT /api/orders/:id зі зміненим статусом	HTTP 200, statusHistory[] містить новий запис з датою та changedBy	✓
Т-10	Історія не змінюється без зміни статусу	PUT /api/orders/:id без зміни поля status	HTTP 200, довжина statusHistory[] не збільшилась	✓
Т-11	Автоматичний розрахунок paymentStatus	POST /api/orders: totalAmount=5000, paidAmount=2500	paymentStatus = «partial»	✓
Т-12	paymentStatus при повній оплаті	PUT /api/orders/:id: totalAmount=5000, paidAmount=5000	paymentStatus = «paid»	✓
Т-13	Деактивація позиції каталогу	PATCH /api/products/:id/toggle (isActive: true → false)	HTTP 200, isActive = false	✓
Т-14	Захист від самовидалення адміна	DELETE /api/users/:id (власний ID)	HTTP 400, «Не можна видалити власний акаунт»	✓
Т-15	Структура відповіді модуля статистики	GET /api/statistics з коректним JWT	HTTP 200, відповідь містить totalClients, topClients[], ordersByDay[]	✓

Усі 15 тестових сценаріїв пройдено успішно (✓). Результати підтверджують: коректну JWT-автентифікацію та рольовий доступ (Т-01–Т-04), логіку клієнтської бази зі збереженням дублікатів (Т-05–Т-07), автонумерацію та stateful-логіку замовлень (Т-08–Т-12), роботу каталогу (Т-13) та безпекові обмеження (Т-14). Модуль статистики (Т-15) повертає коректну структуру з усіма очікуваними полями.

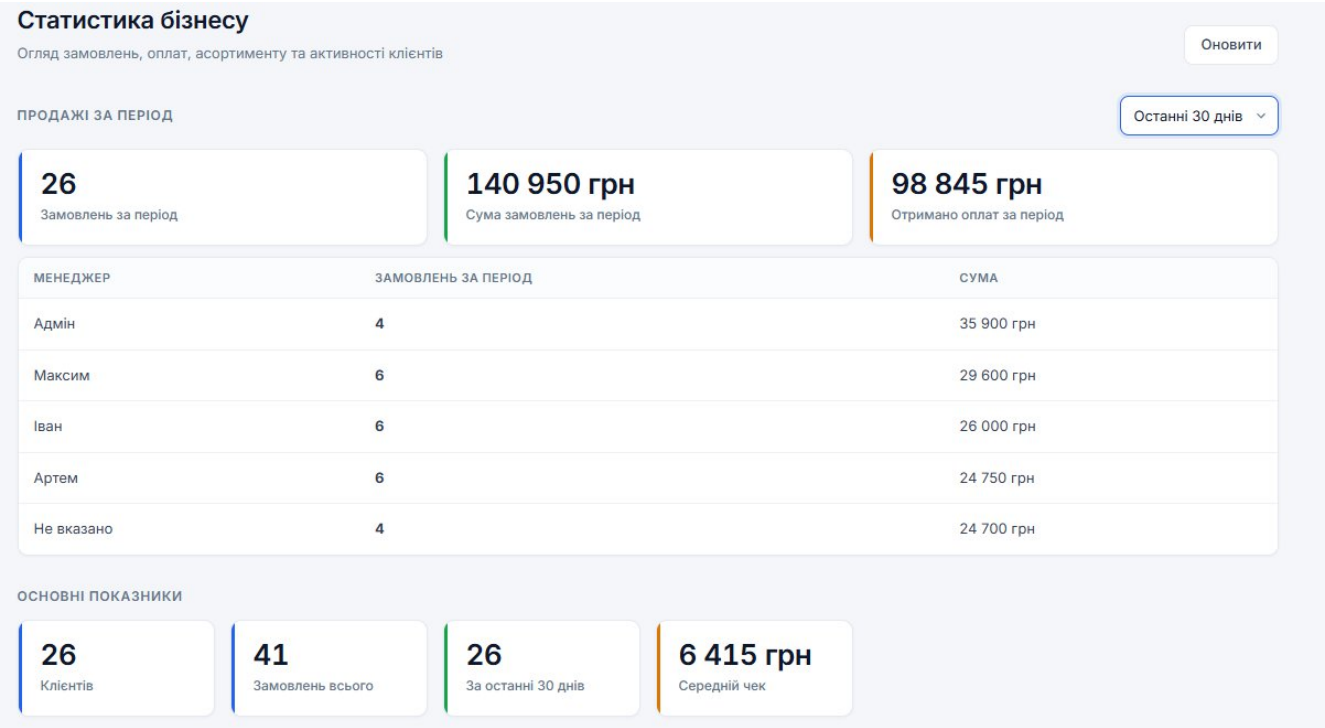


Рис. 3.11 Система після завантаження тестових даних через seedDemoData.js

У третьому розділі описано практичну реалізацію всіх модулів веб-платформи. На основі виконаної роботи можна зробити такі висновки:

– Точка входу server.js організована за принципом єдиної відповідальності: підключення залежностей, реєстрація маршрутів, запуск сервера. Бізнес-логіка повністю зосереджена у контролерах, що спрощує навігацію та підтримку.

– Модуль автентифікації реалізує stateless JWT-захист: сервер не зберігає сесії, кожен запит самодостатній. Рольова модель (admin / manager) обробляється двома незалежними middleware, що комбінуються за потребою.

– Модуль замовлень вирішує три нетривіальних завдання: автонумерація без атомарних лічильників, одночасний облік кількох товарів і послуги в одному замовленні, автоматичний розрахунок paymentStatus і фіксація історії статусів лише при реальній зміні.

– Модуль статистики агрегує великий обсяг показників одним запитом, обробляючи дані в пам'яті через Map-структури. Підхід є ефективним при обсягах малого бізнесу і не потребує складних агрегаційних пайплайнів MongoDB.

– Клієнтська частина реалізована на Vanilla JS без фреймворків, що мінімізує залежності. Спільні функції (nav.js, modal.js, authHeader) виокремлені для повторного використання на всіх сторінках системи.

– Функціональне тестування підтвердило коректну роботу всіх 15 сценаріїв, що охоплюють критичні шляхи: автентифікацію, рольовий доступ, валідацію, бізнес-логіку замовлень та безпекові обмеження.

Реалізована платформа повністю відповідає функціональним та нефункціональним вимогам, сформульованим у розділі 2, і готова до використання після розгортання на сервері.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі розроблено та науково обґрунтовано веб-платформу для управління замовленнями та клієнтами малого бізнесу з використанням JavaScript-технологій. Практичну реалізацію виконано у вигляді повністю функціонуючої системи, протестованої на прикладі умовного підприємства «Безпека Плюс». У процесі виконання роботи вирішено всі чотири завдання, сформульовані у вступі, та досягнуто поставленої мети. На підставі отриманих результатів зроблено такі висновки:

Проаналізовано сучасний стан і тенденції цифровізації малого бізнесу. Встановлено, що більшість підприємств сфери технічних послуг покладаються на ручний облік у таблицях, що при зростанні кількості активних замовлень понад 20–30 на місяць призводить до операційних помилок і втрат. Порівняльний аналіз існуючих CRM-систем (HubSpot, Pipedrive, Zoho CRM) та ERP-рішень (NetSuite, SAP Business One) показав, що жодна з них не задовольняє одночасно критеріям вартості, простоти освоєння та наявності специфічних функцій для обліку монтажних замовлень, що обґрунтовує доцільність власної розробки.

Спроєктовано трирівневу клієнт-серверну архітектуру з REST API (27 ендпоінтів, 6 груп ресурсів). Гібридний підхід MPA-навігація + SPA-поведінка через Fetch API дозволив відмовитися від важких JavaScript-фреймворків. Серверна частина організована за патерном MVC. Спроєктовано чотири Mongoose-моделі (User, Client, Order, Product); модель Order є центральною і містить вкладені масиви products[] та statusHistory[]. Визначено та задокументовано 17 функціональних і 10 нефункціональних вимог до системи.

Розроблено та реалізовано серверну частину платформи на Node.js/Express із системою JWT-автентифікації та рольового доступу (admin/manager) через middleware protect та adminOnly. Реалізовано CRUD-модулі для клієнтів (з перевіркою дублікатів), замовлень (автонумерація ORD-XXXX, автоматичний розрахунок paymentStatus, фіксація statusHistory[]) та каталогу товарів і послуг (з деактивацією без видалення). Розроблено модуль статистики, що формує 25+

аналітичних показників одним запитом, включаючи динаміку за 30 днів, рейтинги клієнтів і менеджерів та SVG-графіки без сторонніх бібліотек.

Проведено функціональне тестування системи за 15 сценаріями методом «чорного ящика». Усі сценарії пройдено успішно, що підтвердило коректну роботу механізмів автентифікації та рольового доступу, бізнес-логіки замовлень, обробки клієнтів та безпекових обмежень. Для генерації тестових даних розроблено seed-скрипт, що створює реалістичний набір із 4 користувачів, 14 позицій каталогу та замовлень за 2 місяці.

Розроблена веб-платформа є завершеним функціонуючим програмним продуктом. Вона повністю відповідає сформульованим функціональним та нефункціональним вимогам і готова до розгортання у виробничому середовищі. Архітектурні рішення, рольова модель доступу, структура REST API та схема бази даних можуть бути використані як основа для створення аналогічних систем для підприємств малого бізнесу у сферах надання технічних послуг, дрібнооптової торгівлі або сервісного обслуговування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Державна служба статистики України. Офіційний сайт. URL: https://stat.gov.ua/ (дата звернення: 15.05.2026).

2. Kraus S., Jones P., Kailer N., Weinmann A., Chaparro-Banegas N., Roig-Tierno N. Digital Transformation: An Overview of the Current State of the Art of Research. SAGE Open. 2021. Vol. 11, No. 3. DOI: 10.1177/21582440211047576. URL: https://doi.org/10.1177/21582440211047576

3. Brown E. *Web Development with Node and Express: Leveraging the JavaScript Stack*. 2nd ed. Sebastopol: O'Reilly Media, 2019. 332 p. ISBN 978-1492053514.

4. Frisbie M. *Professional JavaScript for Web Developers*. 4th ed. Hoboken: Wiley, 2019. 1240 p. ISBN 978-1119366447.

5. Casciaro M., Mammino L. *Node.js Design Patterns: Design and Implement Production-Grade Node.js Applications Using Proven Patterns and Techniques*. 3rd ed. Birmingham: Packt Publishing, 2020. 660 p. ISBN 978-1839214110.

6. Fielding R. T., Taylor R. N. Principled design of the modern Web architecture. ACM Transactions on Internet Technology. 2002. Vol. 2, No. 2. P. 115–150. DOI: 10.1145/514183.514185. URL: https://doi.org/10.1145/514183.514185

7. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing. 2010. Vol. 14, No. 6. P. 80–83. DOI: 10.1109/MIC.2010.145. URL: https://doi.org/10.1109/MIC.2010.145

8. Bradshaw S., Brazil E., Chodorow K. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. 3rd ed. Sebastopol: O'Reilly Media, 2019. 514 p. ISBN 978-1491954461.

9. Richards M., Ford N. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol: O'Reilly Media, 2020. 432 p. ISBN 978-1492043454.

10. Newman S. *Building Microservices: Designing Fine-Grained Systems*. 2nd ed. Sebastopol: O'Reilly Media, 2021. 612 p. ISBN 978-1492034025.

11. Haverbeke, M. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd ed. San Francisco: No Starch Press, 2018. 472 p. ISBN 978-1593279509.

12. Subramanian H., Raj P. Hands-On RESTful API Design Patterns and Best Practices. Birmingham : Packt Publishing, 2019. 378 p. ISBN 978-1788992664.

13. Node.js Foundation. *Node.js Documentation*. 2024. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 15.05.2026).

14. Express.js Team. *Express.js — Fast, Unopinionated, Minimalist Web Framework for Node.js*. 2024. URL: https://expressjs.com/ (дата звернення: 15.05.2026).

15. MongoDB Inc. *MongoDB Manual*. 2024. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 15.05.2026).

16. Automattic Inc. *Mongoose ODM v8 Documentation*. 2024. URL: https://mongoosejs.com/docs/ (дата звернення: 15.05.2026).

17. MDN Web Docs. *Fetch API*. Mozilla Foundation, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 15.05.2026).

18. OWASP Foundation. *OWASP Top Ten: The Ten Most Critical Web Application Security Risks*. 2023. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 15.05.2026).

19. Flanagan D. *JavaScript: The Definitive Guide*. 7th ed. Sebastopol: O'Reilly Media, 2020. 706 p. ISBN 978-1491952023.

20. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p. ISBN 978-0321127426.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Веб-платформа управління замовленнями та клієнтами малого бізнесу з використанням JavaScript

Виконав: Богдан САНЖАРОВСЬКИЙ, гр. ІСД-41

Керівник: Оксана ТКАЛЕНКО, к.т.н., доцент

Київ, 2026

Актуальність та мета роботи

Актуальність

Цифровізація малого бізнесу є одним із пріоритетних напрямів розвитку економіки України — малі та середні підприємства становлять понад 90% усіх діючих компаній, що визначає їх ключову роль в економіці країни. Підприємства сфери технічних послуг покладаються на Excel та ручний облік, що при 20–40 активних замовленнях призводить до операційних помилок, втрачених заявок та запізнілого виявлення заборгованості. Готові CRM-системи (HubSpot, Pipedrive, Zoho) є дорогими, надлишковими і не адаптованими під специфіку монтажних підприємств.

МЕТА

Розробити веб-платформу для управління замовленнями та клієнтами малого бізнесу на стеку

Node.js + Express + MongoDB

Об'єкт: процес управління замовленнями та клієнтами в малому бізнесі у сфері технічних послуг
Предмет: методи та засоби розробки веб-платформ на основі JavaScript-технологій для автоматизації операційних процесів

Завдання дослідження

01 Проаналізувати стан цифровізації малого бізнесу та визначити вимоги до спеціалізованих веб-рішень для управління замовленнями

02 Дослідити архітектурні підходи та спроєктувати архітектуру платформи, структуру бази даних і REST API

03 Розробити серверну частину на Node.js/Express із JWT-автентифікацією, рольовим доступом, CRUD-модулями та модулем статистики

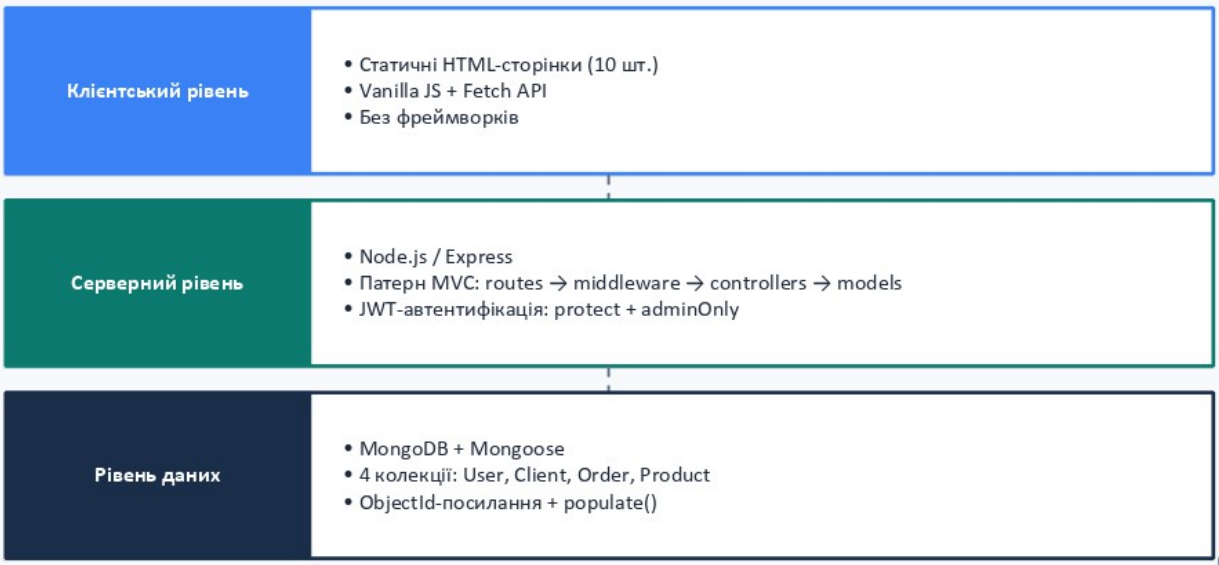
04 Провести функціональне тестування системи та підтвердити відповідність реалізованого функціоналу сформульованим вимогам

Порівняльний аналіз існуючих рішень

Система	Вартість	Обмеження для задач проєкту
HubSpot CRM	Безкоштовно / $45+/міс.	Немає обліку монтажних замовлень, зорієнтований на воронку продажів
Pipedrive	$14–99/міс.	Відсутня історія статусів, немає обліку товарів у замовленні
Zoho CRM	$14–52/міс.	Складне налаштування під специфічну логіку монтажних послуг
Google Sheets	Безкоштовно	Немає автоматизації, рольового доступу, прив'язки клієнт–замовлення
✓ Власна платформа	**Одноразова розробка**	**Повна відповідність: товари, послуги, статуси, оплата, аналітика**

Висновок: жодна з готових систем не задовольняє одночасно критеріям вартості, простоти та специфічного функціоналу

Архітектура системи



База даних та моделі

User	Client	Product	Order
name, email password (bcrypt) role: admin | manager	fullName, phone email, address createdBy → User	name, category price, type product | service isActive	— Центральна модель

Модель Order — ключові особливості:

- **products[]** — Вкладений масив товарів із кількістю — без окремих таблиць-зв'язок
- **statusHistory[]** — Журнал змін статусу: статус + дата + автор зміни (фіксується автоматично)
- **paymentStatus** — Автоматичний розрахунок: unpaid / partial / paid на основі totalAmount та paidAmount
- **orderNumber** — Автогенерація у форматі ORD-XXXX при кожному новому замовленні

REST API — специфікація

27 ендпоінтів • 6 груп ресурсів • 2 рівні захисту

/api/auth	Вхід, JWT-токен	public	
/api/products	CRUD + toggle isActive	protect	
/api/clients	CRUD + пошук + дубл.	protect	
/api/users	CRUD менеджерів + статистика	adminOnly	
/api/orders	CRUD + фільтри + статуси	protect	
/api/statistics	Зведена аналітика	protect	

admin — повний доступ до всіх модулів + управління обліковими записами менеджерів **manager** — робота з клієнтами, замовленнями та каталогом; без доступу до /api/users

Ключові модулі реалізації

Автентифікація

- bcrypt-хешування паролів (salt rounds = 10)
- JWT з payload: id, name, role (термін дії 8 год)
- Middleware protect + adminOnly — stateless-захист

Модуль замовлень

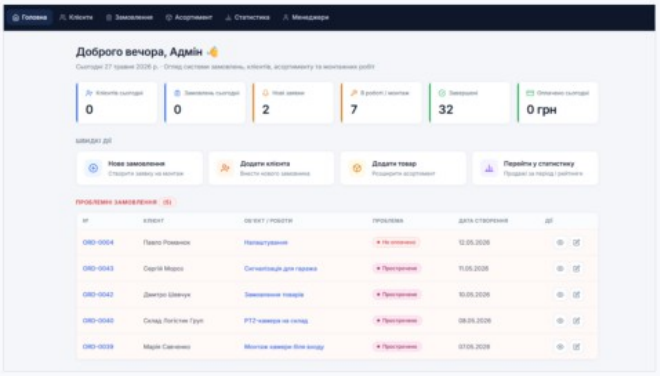
- Автонумерація ORD-XXXX при кожному створенні
- Одночасний облік кількох товарів і послуги
- Автоматичний розрахунок paymentStatus
- Фіксація statusHistory[] лише при реальній зміні

Модуль статистики

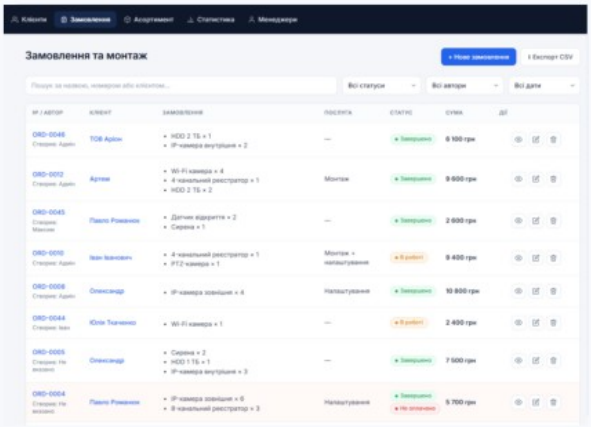
- 25+ показників одним запитом до API
- Динаміка замовлень і виручки за 30 днів
- Топ-5 клієнтів, топ-5 товарів, рейтинг менеджерів
- SVG-графіки нативно — без сторонніх бібліотек

Демонстрація інтерфейсу системи

Головна панель

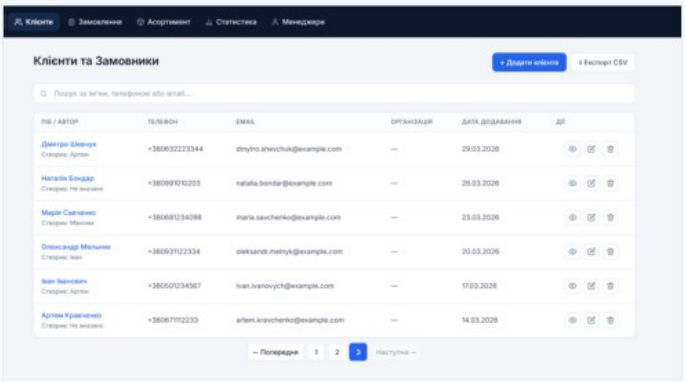


Управління замовленнями

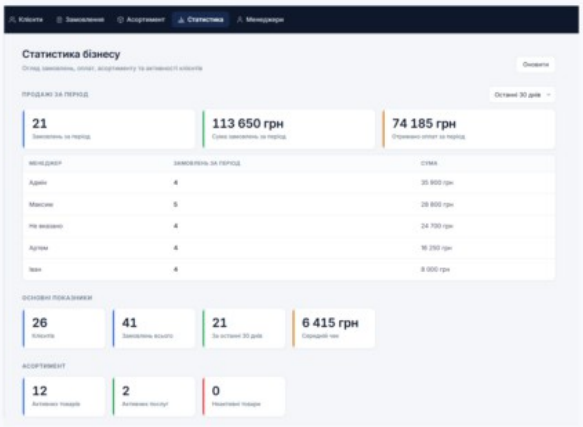


Демонстрація інтерфейсу системи (продовження)

Сторінка клієнтів



Модуль статистики



Тестування системи

15 тестових сценаріїв пройдено успішно

Тестування охопило всі ключові модулі системи: автентифікацію, управління клієнтами, замовленнями, каталогом та статистикою. Всі 15 сценаріїв підтвердили коректну роботу системи.

Охоплені сценарії:

- ✓ JWT-автентифікація та рольовий доступ (T-01–T-04)
- ✓ Логіка клієнтської бази: дубл. перевірка (T-05–T-07)
- ✓ Автонумерація та бізнес-логіка замовлень (T-08–T-12)
- ✓ Деактивація каталогу та захист від самовидалення (T-13–T-14)
- ✓ Структура відповіді модуля статистики (T-15)

Тестові дані (seed-скрипт npm run seed:demo):

4 тестових користувачі (1 адмін + 3 менеджери) • 14 позицій каталогу (12 товарів + 2 послуги) • Замовлення за 2 місяці з різними статусами та сумами

Висновки та практична значущість

Основні результати:

1. Підтверджено доцільність власної розробки — жодна готова CRM не задовольняє одночасно трьом критеріям
2. Спроєктовано трирівневу архітектуру з REST API (27 ендпоінтів, 6 груп ресурсів)
3. Реалізовано 4 Mongoose-моделі; Order — з вкладеними products[] та statusHistory[]
4. JWT + bcrypt забезпечують stateless-захист; рольова модель admin/manager
5. Всі 15 тестових сценаріїв пройдено успішно

Практична значущість

- Повністю функціонуюча система, протестована на прикладі «Безпека Плюс»
- Ведення клієнтської бази, замовлень, оплати, аналітики
- Архітектура та REST API можуть бути основою для аналогічних систем у сферах технічних послуг та дрібнооптової торгівлі

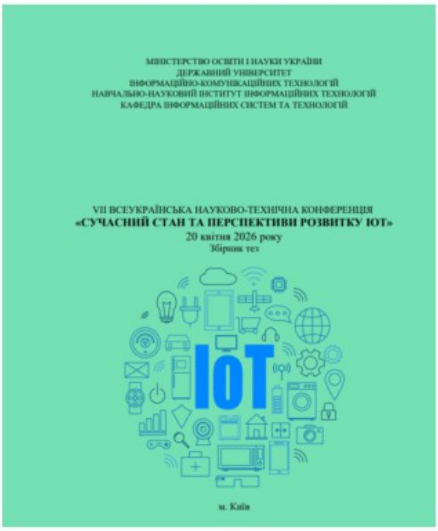
АПРОБАЦІЯ РЕЗУЛЬТАТІВ

Окремі результати дослідження щодо застосування стеку **Node.js, Express та MongoDB** для побудови веб-систем автоматизації процесів збору клієнтських даних та управління замовленнями було представлено у тезах доповіді **«Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот»** на

VII Міжнародній науково-технічній конференції «Сучасний стан та перспективи розвитку IoT»

20 квітня 2026 року

Державний університет інформаційно-комунікаційних технологій, м. Київ



Дякую за увагу!