МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРОЦЕНКО АНАСТАСІЯ СЕРГІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
_____ Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

ІНФОРМАЦІЙНА СИСТЕМА ІНТЕРНЕТ-МАГАЗИНУ ТОВАРІВ ЕНЕРГОНЕЗАЛЕЖНОСТІ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Ірина ФРИЗ, старший викладач кафедри
інформаційних технологій,
канд. фіз.-мат. наук
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Проценко А.С. Інформаційна система інтернет-магазину товарів енергонезалежності. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено ринок товарів енергонезалежності, проаналізовано існуючі вебрішення та сформовано вимоги до системи. За допомогою технологій Vue 3, Node.js, Fastify, PostgreSQL та Drizzle ORM розроблено інформаційну систему інтернет-магазину, що реалізує каталог, кошик, оформлення замовлень, JWT-автентифікацію та адміністративну панель.

Ключові слова: інформаційна система, інтернет-магазин, енергонезалежність, Vue 3, Node.js, Fastify, PostgreSQL, Drizzle ORM, REST API, JWT, Docker, SPA.

58 ст., 7 рис., 14 табл., 4 дод., 43 джерела.

ABSTRACT

Protsenko A.S. Information System of the Online Store of Energy Independence Products. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

In the qualification (bachelor's) work, the energy independence products market is investigated, existing web solutions are analyzed, and system requirements are defined. Using Vue 3, Node.js, Fastify, PostgreSQL, and Drizzle ORM technologies, an online store information system has been developed, implementing a product catalog, shopping cart, order placement, JWT authentication, and an administrative panel.

Keywords: information system, online store, energy independence, Vue 3, Node.js, Fastify, PostgreSQL, Drizzle ORM, REST API, JWT, Docker, SPA.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації суспільства електронна комерція посідає важливе місце в економічній діяльності підприємств і повсякденному житті споживачів. З кожним роком зростає кількість користувачів, які віддають перевагу онлайн-покупкам завдяки їхній зручності, доступності та економії часу. Це, у свою чергу, стимулює розвиток вебтехнологій та створення ефективних, функціональних і безпечних вебзастосунків для електронної торгівлі.

Актуальність теми дослідження зумовлена необхідністю розроблення сучасних вебзастосунків електронної комерції, які забезпечували б зручну взаємодію між користувачами та адміністраторами системи, а також відповідали актуальним вимогам до продуктивності, безпеки та масштабованості. Незважаючи на значну кількість існуючих рішень у сфері онлайн-торгівлі, залишається низка проблем, пов’язаних із оптимізацією користувацького досвіду, ефективною організацією взаємодії між клієнтською та серверною частинами, а також забезпеченням гнучкості систем для подальшого розвитку. Окремої уваги потребують питання інтеграції функціоналу керування товарами, обробки замовлень і персоналізації взаємодії з користувачем у межах єдиного програмного продукту.

Мета дослідження полягає у проєктуванні та реалізації інформаційної системи інтернет-магазину товарів енергонезалежності з використанням сучасних вебтехнологій, що забезпечують ефективну клієнт-серверну взаємодію, безпечну обробку даних, зручність користування, продуктивність і можливість подальшого масштабування системи відповідно до потреб користувачів та особливостей предметної області.

Для досягнення поставленої мети визначено такі завдання дослідження:

- проаналізувати сучасний стан та тенденції розвитку вебзастосунків електронної комерції;
- виявити основні вимоги до функціональності та архітектури систем електронної торгівлі;

- узагальнити підходи до організації клієнт-серверної взаємодії у вебзастосунках;
- дослідити методи забезпечення автентифікації та обробки користувацьких даних;
- розробити структуру та логіку функціонування вебзастосунку електронної торгівлі;
- проаналізувати результати реалізації та оцінити ефективність запропонованого рішення.

Об’єктом дослідження є робота вебзастосунків, що забезпечують функціонування електронної комерції. Він охоплює загальну логіку роботи онлайн-магазинів, включно з обробкою дій користувачів, управлінням товарним наповненням, формуванням інтерфейсу та підтримкою ключових процесів взаємодії в цифровому торговельному середовищі. Об’єкт розглядається як комплексна система, метою якої є забезпечення стабільної та зрозумілої роботи інтернет-платформи для кінцевого користувача.

Предметом дослідження є методи, підходи та програмні засоби, що забезпечують побудову клієнт-серверної архітектури вебзастосунків для електронної торгівлі, а також особливості реалізації їхніх функціональних можливостей. У межах дослідження розглядаються принципи організації взаємодії між клієнтською і серверною частинами системи, способи опрацювання та зберігання даних, механізми підтримки користувацької активності та виконання основних бізнес-процесів інтернет-магазину. Особлива увага приділяється питанням забезпечення надійності, безпеки та ефективності роботи вебзастосунку, а також закономірностям, що визначають якість користувацького досвіду та відповідність системи функціональним вимогам. Предмет дослідження охоплює як теоретичні аспекти побудови веборієнтованих інформаційних систем, так і практичні аспекти їхньої реалізації.

Теоретичне та практичне значення одержаних результатів полягає у систематизації підходів до створення вебзастосунків електронної комерції та можливості використання отриманих результатів при розробленні аналогічних

інформаційних систем. Практична цінність роботи полягає у створенні функціонального вебзастосунку, який може бути використаний як основа для подальшого розвитку комерційних платформ або навчальних проєктів, а також як приклад реалізації сучасних підходів до побудови клієнт-серверних систем.

Структура кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються теоретичні аспекти, аналіз предметної області та вибір інструментів реалізації. У другому розділі описується проєктування системи та обґрунтування обраних рішень. Третій розділ присвячений реалізації вебзастосунку, тестуванню та аналізу отриманих результатів. У висновках узагальнюються результати дослідження. Список використаних джерел містить 43 найменування, залучених у процесі виконання роботи. Основна частина роботи містить 14 таблиць та 7 рисунків, а загальний обсяг становить 58 сторінок. Додатки містять допоміжні матеріали, що деталізують основні проєктні рішення, зокрема, у них представлено таблиці порівняльного аналізу технологій розробки, прав доступу та маршрутів клієнтської частини, а також схему бази даних.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1. Аналіз предметної області ринку товарів енергонезалежності

Забезпечення безперебійного електропостачання є однією з найактуальніших проблем сучасного суспільства, яка набула особливої гостроти в умовах збройного конфлікту в Україні. Починаючи з 2022 року, систематичні ракетні та дронові удари по об'єктах критичної інфраструктури спричинили масштабні руйнування в енергетичному секторі країни, унаслідок чого мільйони громадян та підприємств зіткнулися з тривалими відключеннями електроенергії – так званими «блекаутами». У цих умовах попит на засоби автономного енергозабезпечення: портативні зарядні станції, генератори, сонячні панелі, інвертори та павербанки – зріс у рази, сформувавши принципово новий сегмент споживчого ринку.

Внаслідок масованих ракетних атак на об'єкти енергетичної інфраструктури протягом 2022-2024 років понад 70% теплових генеруючих потужностей було зруйновано або виведено з ладу, а загальні втрати генерації у 2024 році склали близько 9-10 ГВт [1]. Це спричинило значний дефіцит електроенергії в об'єднаній енергосистемі (ОЕС) країни, що призвело до введення графіків відключень – від 2 до 14 годин на добу для споживачів залежно від регіону та інтенсивності обстрілів [2].

Наведена ситуація обумовила стрімкий розвиток ринку товарів енергонезалежності. Відповідно до досліджень аналітичних компаній, обсяг цього ринку в Україні у 2023 році оцінювався на рівні 1,8-2,2 млрд доларів США, а темпи його зростання порівняно з 2021 роком склали понад 400% [3]. Прогнозується, що навіть після завершення активної фази конфлікту ринок збереже підвищену активність завдяки формуванню сталої культури енергетичної незалежності серед населення та підприємств.

Ринок товарів енергонезалежності охоплює широкий спектр пристроїв і систем, призначених для накопичення, перетворення та автономного вироблення електричної енергії. Основу асортименту сучасних інтернет-магазинів цього напряму формують портативні зарядні станції, генератори, сонячні панелі, інвертори, джерела безперебійного живлення, а також павербанки й портативні акумулятори.

Цільова аудиторія цього ринку є різнорідною та умовно поділяється на два основні сегменти: B2C і B2B. До B2C належать приватні споживачі, які купують обладнання для дому, квартири чи особистого користування. Для них важливими є доступна ціна, просте пояснення технічних характеристик, відгуки інших покупців і зручність онлайн-замовлення. До B2B сегмента належать підприємства та організації, що потребують більш потужних і надійних рішень для безперервної роботи бізнесу. Для них пріоритетними є технічні параметри, гарантійне обслуговування, офіційна документація та можливість закупівлі більших партій товару [4].

Урахування особливостей обох сегментів є важливим під час проєктування інформаційної системи інтернет-магазину, оскільки впливає на структуру каталогу, систему фільтрації, подання характеристик товарів, механізми оформлення замовлень і способи взаємодії з клієнтами.

Продаж товарів енергонезалежності в онлайн-середовищі має низку специфічних рис, які відрізняють цей напрям від багатьох інших категорій електронної комерції та безпосередньо визначають вимоги до функціональності майбутньої системи.

На відміну від товарів широкого вжитку, обладнання для автономного енергопостачання характеризується значною технічною складністю. Покупець перед здійсненням покупки, як правило, має потребу в розумінні таких параметрів, як потужність, ємність, тип акумулятора, вихідна напруга тощо. Отже, детальні технічні специфікації є критично важливим фактором.

Дослідження поведінки онлайн-покупців засвідчують, що понад 70 % клієнтів у сегменті складної техніки звертаються за консультацією перед

здійсненням покупки [5]. Тому наявність у системі зручних каналів зворотного зв'язку – форми запиту на консультацію, телефонної лінії – є обов'язковою умовою ефективного інтернет-магазину в даному сегменті.

В умовах нестабільного ринку ціни на товари енергонезалежності можуть змінюватися кілька разів на тиждень. Водночас на популярні позиції часто виникає дефіцит. Ці фактори обумовлюють вимоги до системи управління товарними запасами та актуальністю цін у режимі реального часу.

Таким чином, аналіз предметної області ринку товарів енергонезалежності дозволяє зробити висновок, що цей ринок є динамічним, технічно складним та має виражену специфіку, яка суттєво відрізняє його від інших сегментів електронної комерції. Зазначені особливості мають бути враховані під час формування функціональних вимог до інформаційної системи інтернет-магазину, що буде здійснено в підрозділі 1.4 даної роботи.

1.2. Огляд існуючих аналогів інтернет-магазинів енергонезалежності

Для виявлення переваг, недоліків та технологічних обмежень наявних рішень у сегменті онлайн-продажу товарів енергонезалежності здійснено порівняльний аналіз чотирьох платформ: двох великих маркетплейсів – Rozetka та Epicentr, а також двох спеціалізованих магазинів – Yasno та офіційного магазину EcoFlow Ukraine. Аналіз проводився за критеріями функціональності, технологічного стеку та якості взаємодії з користувачем.

Rozetka (rozetka.ua) – найбільший маркетплейс України із щомісячною аудиторією понад 45 мільйонів відвідувань [6-7]. Платформа побудована на власній монолітній архітектурі із використанням Java на серверному боці та Angular-based SPA для клієнтської частини. Серверний рендеринг (SSR) реалізований через Node.js-проміжний шар. CDN-розподіл статичних активів здійснюється через Cloudflare.

Сильні сторони платформи: широкий асортимент товарів з характеристиками, розвинена система відгуків і рейтингів, розгалужена мережа видачі

замовлень, підтримка онлайн-оплати та кредитування, адаптивний дизайн. Завантаження фільтрів у глибоко вкладених категоріях здійснюється окремими HTTP-запитами без кешування на стороні клієнта, що при слабкому інтернет-з'єднанні призводить до помітних затримок (800–1500 мс).

Epicentr (epicentrk.ua) – мережа будівельних гіпермаркетів, що розвиває онлайн-напрям [8]. Сайт функціонує на PHP-фреймворку Symfony із jQuery-фронтендом і поступово переходить до компонентного підходу на базі Vue.js для окремих сторінок («острівна архітектура»). Використовується СУБД MySQL та Elasticsearch для пошуку.

Серед переваг варто відзначити широкий асортимент генераторів та будівельного обладнання, можливість самовивозу з фізичних магазинів. Разом із тим є суттєві недоліки: змішана архітектура jQuery + частковий Vue.js призводить до неузгодженої поведінки інтерфейсу під час фільтрації та сортуванні. Відсутній функціонал порівняння товарів у категорії енергетичного обладнання. Адаптивність для мобільних пристроїв реалізована частково: ряд сторінок вимагає горизонтальної прокрутки на екранах шириною менше 400 px. Технічний опис генераторів і зарядних станцій не структурований, що ускладнює вибір покупця без попередньої технічної підготовки.

Yasno Shop (yasno.ua) – спеціалізований магазин компанії ДТЕК, орієнтований виключно на товари для автономного енергопостачання [9]. Платформа реалізована на базі CMS Shopify із підключеними сторонніми темами та низкою кастомних React-компонентів для ключових блоків сторінки.

Перевагою ресурсу є вузька спеціалізація, завдяки чому каталог має більш структурований вигляд порівняно з універсальними маркетплейсами. Також на сайті представлено інформаційні матеріали та консультаційні можливості, що сприяє кращому ознайомленню покупців із продукцією.

Разом із тим використання шаблонної SaaS-платформи накладає певні функціональні обмеження. Частина інструментів реалізована в базовому вигляді або потребує додаткових зовнішніх рішень. Система фільтрації переважно орієнтована на стандартні параметри та не завжди забезпечує максимально гнучкий

пошук за вузькоспеціалізованими технічними характеристиками. Інтерфейс окремих сторінок може поступатися індивідуально розробленим рішенням у питаннях адаптивності та зручності взаємодії на різних пристроях.

Оскільки сайт розміщується на серверах Shopify за межами України, для українських користувачів інколи можливе повільніше завантаження сторінок. Використання сторонньої SaaS-платформи також обмежує можливості повної інтеграції з власними ERP- та CRM-системами компанії.

EcoFlow Ukraine (ecoflowukraine.com) – авторизований український магазин продукції бренду EcoFlow, орієнтований на продаж зарядних станцій, сонячних панелей, аксесуарів та суміжних рішень для енергонезалежності [10]. Сайт побудований на платформі Хорошоп, що забезпечує готовий набір інструментів для електронної комерції: каталог товарів, кошик, особистий кабінет, онлайн-оплату та базову адаптивність.

Перевагами ресурсу є сучасний візуальний дизайн, детальний опис продукції, зручна структура каталогу, наявність офіційної гарантії, консультаційної підтримки та кількох способів оплати.

Разом із тим магазин має і певні обмеження. Каталог охоплює продукцію одного бренду, що звужує можливості вибору та порівняння з альтернативними виробниками. Крім того, функціональність підбору обладнання та аналітичні інструменти реалізовані переважно у стандартному форматі.

Проведений аналіз засвідчує, що жоден із розглянутих аналогів не задовольняє повною мірою потреби спеціалізованого інтернет-магазину товарів енергонезалежності. Універсальні маркетплейси (Rozetka, Epicentr) не підтримують специфічних технічних фільтрів і використовують застарілі підходи до клієнтської частини (jQuery, монолітні сторінки без реактивного оновлення), що призводить до повільного відгуку інтерфейсу при фільтрації. Shopify-рішення (Yasno Shop) обмежені функціональністю платформи та не масштабуються під власні бізнес-вимоги. Брендові магазини мають вузький асортимент і не покривають потреби мультивендорного ринку. Виявлені недоліки обґрунтовують доцільність

розробки спеціалізованої інформаційної системи на сучасному реактивному стеку.

1.3. Теоретичні засади розробки вебінформаційних систем

Вебінформаційна система (ВІС) – це програмно-апаратний комплекс, що забезпечує збір, зберігання, обробку та надання інформації через вебсередовище з використанням стандартних мережевих протоколів і технологій [11]. На відміну від традиційних настільних застосунків, ВІС працюють за клієнт-серверним принципом, де користувач взаємодіє із системою через веббраузер, а обробка даних виконується серверною частиною. Обмін даними між компонентами системи зазвичай здійснюється за протоколами HTTP або HTTPS.

У теорії веброзробки виділяють кілька фундаментальних принципів та архітектурних підходів до побудови інформаційних систем. Одним із базових є принцип розмежування відповідальності (Separation of Concerns), який передбачає поділ системи на незалежні компоненти, кожен з яких виконує окрему функцію. Такий підхід зменшує складність системи, підвищує її гнучкість і значно спрощує тестування та супровід.

Практичною реалізацією цього принципу є класична триланкова архітектура, що передбачає розмежування рівнів представлення, бізнес-логіки та даних [12]. Такий підхід забезпечує незалежність шарів, полегшує масштабування та тестування кожного з них. У сучасних вебзастосунках ця модель еволюціонувала у більш гнучку клієнт-серверну архітектуру, де рівень представлення реалізований у браузері як JavaScript-застосунок (SPA), а сервер відповідає за API та бізнес-логіку.

Концепція односторінкового застосунку (Single-Page Application, SPA) полягає в тому, що браузер завантажує один HTML-документ і далі динамічно оновлює його вміст без повного перезавантаження сторінки, використовуючи AJAX/Fetch-запити до серверного API [13]. SPA-підхід забезпечує плавний і швидкий користувацький досвід (UX), що є критично важливим для інтернет-

магазинів, де покупець взаємодіє з великою кількістю динамічного контенту – каталогами, фільтрами, кошиком.

REST (Representational State Transfer) – архітектурний стиль для побудови розподілених гіпермедіа-систем [14]. REST-API визначає набір принципів: клієнт-серверна взаємодія, відсутність стану, кешованість ресурсів, однорідний інтерфейс і використання стандартних HTTP-методів (GET, POST, PUT, DELETE). Дотримання цих принципів забезпечує масштабованість, незалежність клієнта від сервера, зручність тестування і документування API – якості, принципово важливі для комерційних систем.

Проєктування бази даних є ключовим етапом розробки будь-якої інформаційної системи. Реляційна модель даних базується на табличному поданні інформації та нормалізації – процесі усунення надлишковості й аномалій даних шляхом розбиття таблиць до нормальних форм (1НФ, 2НФ, 3НФ, BCNF) [15]. Реляційні СУБД забезпечують суворе дотримання ACID-властивостей транзакцій: атомарності, узгодженості, ізольованості та довговічності – що є обов'язковою умовою для фінансових та комерційних операцій в інтернет-магазині.

Об'єктно-реляційне відображення (Object-Relational Mapping) – технологія, що відображає об'єкти програмного застосунку на таблиці реляційної бази даних, дозволяючи розробнику взаємодіяти з даними через об'єктно-орієнтований API без написання SQL-запитів вручну [16]. Сучасні ORM, на відміну від класичних рішень, реалізують типобезпечні схеми та міграції, що значно знижує ризик помилок під час модифікації структури бази даних.

Безпека вебзастосунків регламентується рекомендаціями OWASP (Open Web Application Security Project). Серед найбільш критичних загроз виокремлюють: SQL-ін'єкції, міжсайтовий скриптинг (XSS), підробку запитів (CSRF), витік конфіденційних даних та порушення автентифікації [17]. Для їхнього запобігання застосовуються параметризовані запити, CSP-заголовки, HTTPS, та механізми токенізованої автентифікації – зокрема JSON Web Token, що забезпечує stateless-верифікацію особи користувача без необхідності зберігання сесій на сервері.

Адаптивний вебдизайн (Responsive Web Design, RWD) передбачає побудову інтерфейсів, що автоматично адаптуються до будь-якого розміру екрана [18]. Досягається це через використання відносних одиниць виміру, гнучких сіток (CSS Grid, Flexbox) та медіазапитів. В умовах, коли понад 55 % трафіку електронної комерції в Україні надходить з мобільних пристроїв [19], адаптивність є не перевагою, а базовою вимогою до будь-якої комерційної системи.

Розглянуті теоретичні засади формують концептуальну основу для проєктування інформаційної системи інтернет-магазину: триланкова архітектура визначає структуру застосунку; принципи REST – правила побудови API; теорія реляційних баз даних – підхід до проєктування схеми даних; стандарти безпеки OWASP – вимоги до захисту системи; концепція RWD – принцип проєктування інтерфейсу.

1.4. Постановка завдання та функціональних вимог

У процесі аналізу предметної області було встановлено потребу у створенні вебінформаційної системи інтернет-магазину товарів енергонезалежності, яка забезпечуватиме зручний вибір продукції, оформлення замовлень, адміністрування каталогу та взаємодію з користувачами. Необхідність розроблення такого рішення зумовлена зростанням попиту на резервні джерела живлення, а також наявністю недоліків у чинних онлайн-магазинах.

Основним завданням проєктування є створення системи, що поєднуватиме зрозумілий користувацький інтерфейс, структурований каталог товарів, засоби керування замовленнями та надійний адміністративний модуль.

Для реалізації поставленої мети визначено три ролі з різними рівнями доступу та наборами функцій.

Гість (неавторизований відвідувач) – будь-який користувач, який відвідує сайт без проходження процедури автентифікації. Для цієї ролі доступна публічна частина інтернет-магазину, що включає перегляд каталогу товарів, використання пошуку та фільтрації, перегляд детальної інформації про товари, а також

можливість додавання товарів до кошика та оформлення замовлення. При цьому доступ до персоналізованих функцій системи для гостя обмежений.

Авторизований клієнт – зареєстрований та автентифікований користувач системи. Після входу до облікового запису клієнт отримує доступ до розширеного функціоналу, зокрема персонального кошика, оформлення та відстеження замовлень, а також особистого кабінету. У профілі користувача зберігається історія покупок, персональні дані та інформація про попередні замовлення, що забезпечує більш зручний і персоналізований процес взаємодії з магазином.

Адміністратор – представник магазину з повними правами доступу. Управляє товарним каталогом, замовленнями, користувачами, а також контролює коректність роботи системи через захищену адміністративну панель.

Деталізований список прав для кожної ролі наведено в Додатку А. На основі прецедентів розглянемо детально вимоги за модулями системи.

Модуль каталогу та пошуку: система має забезпечувати перегляд товарів за ієрархічними категоріями; багатокритеріальну фільтрацію (за ціною, брендом, потужністю, ємністю); сортування за ціною, рейтингом; пагінацію.

Модуль картки товару: кожен товар має мати сторінку з технічними характеристиками у структурованому вигляді, зображенням, описом, рейтингом, кнопкою додавання у кошик, інформацією про наявність.

Модуль кошика та замовлення: система має реалізовувати кошик покупок із збереженням вмісту між сесіями користувачів; форму оформлення замовлення з введенням даних доставки та вибором способу оплати; відстеження статусу замовлення.

Модуль автентифікації: реєстрація та вхід за email і паролем; захищений вихід із системи; збереження сесії.

Адміністративний модуль: панель управління має забезпечувати CRUD-операції над товарами (створення, читання, оновлення, видалення); управління категоріями; зміну статусів замовлень; перегляд списку зареєстрованих користувачів та списку повідомлень у службу підтримки.

Окрім функціональних вимог, важливе значення мають нефункціональні, що визначають якісні характеристики системи. До них належить продуктивність – швидке завантаження сторінок і обробка запитів користувачів. Безпека передбачає захист персональних даних, автентифікацію та розмежування прав доступу. Надійність означає стабільну роботу системи та збереження цілісності даних. Зручність використання передбачає інтуїтивний інтерфейс і просту навігацію. Масштабованість забезпечує можливість розширення функціоналу та зростання кількості користувачів без ускладнення архітектури. Дотримання зазначених вимог є необхідною умовою створення конкурентоспроможної та ефективної інформаційної системи.

Отже, необхідно вирішити завдання, пов’язані з вибором технологій, проєктуванням архітектури та бази даних, розробкою й інтеграцією клієнтської та серверної частин, а також тестуванням системи. Важливими аспектами є забезпечення зручності користування, надійності та можливості подальшого розвитку системи. Виконання цих завдань дасть змогу створити ефективну вебінформаційну систему інтернет-магазину, що відповідатиме сучасним вимогам електронної комерції.

1.5. Вибір технологій для реалізації системи

Вибір технологічного стеку є фундаментальним рішенням під час проєктування інформаційної системи, оскільки він визначає продуктивність, масштабованість, безпеку та можливість підтримки продукту протягом усього його життєвого циклу. У цьому підрозділі обґрунтовано вибір кожної із застосованих технологій на підставі порівняльного аналізу альтернатив, вимог предметної галузі та специфіки розроблюваної системи.

Vue 3 як фреймворк клієнтської частини. Для реалізації фронтенду обрано Vue 3 із Composition API. Основним аргументом на користь Vue 3 є його реактивна система на базі Proxy-об’єктів, яка забезпечує автоматичне відстеження залежностей та точкове оновлення DOM без необхідності ручного

визначення спостережуваних властивостей. Це критично важливо для інтернет-магазину, де каталог, кошик та фільтри є взаємозалежними реактивними сутностями.

Composition API – є центральним нововведенням Vue 3, що дозволяє організовувати логіку компонента за функціональним принципом, а не за типами опцій (data, methods, computed). Це суттєво покращує читабельність і тестованість великих компонентів, а також дає змогу виносити повторювану логіку у composable-функції – аналог React Hooks, але з більш явним зв'язуванням реактивних залежностей. Вбудована підтримка TypeScript у Vue 3 через defineComponent та <script setup> забезпечує статичну типізацію без зайвого синтаксичного навантаження [20].

Для обґрунтування вибору технології клієнтської частини було проведено порівняльний аналіз найбільш поширених SPA-фреймворків, що використовуються у сучасній веброзробці. Під час оцінювання враховувалися такі критерії, як складність освоєння, розмір бандлу, підтримка TypeScript, можливості керування станом застосунку та підтримка серверного рендерингу. Angular надмірно великий для проєктів без корпоративного масштабу [21]; React, попри потужну екосистему, вимагає суттєво більш шаблонного коду для організації стану та маршрутизації [22]. У таблиці Б.1 Додатку Б наведено порівняння Vue 3 з провідними альтернативами за ключовими критеріями вибору для даного проєкту.

Vite як інструмент збірки. Збірник Vite обрано замість Webpack або Parcel з кількох причин. По-перше, Vite використовує нативні ES Modules у браузері під час розробки, що дозволяє запускати dev-сервер практично миттєво незалежно від розміру проєкту – на відміну від Webpack, час старту якого зростає лінійно з кількістю модулів. По-друге, Vite реалізує Hot Module Replacement через WebSocket з точковим оновленням лише зміненого модуля без перезбірки всього графу залежностей, що скорочує час відгуку при розробці до 20–100 мс. По-третє, для робочої збірки Vite використовує Rollup із механізмами усунення зайвого коду та розділенням коду, забезпечуючи оптимальні розміри фінального

бандлу через розподіл на окремі сегменти (сторонні бібліотеки, маршрутизація, сховище стану тощо) [23].

Pinia як менеджер стану. Для централізованого управління станом застосунку обрано Pinia – офіційно рекомендований менеджер стану для Vue 3, що прийшов на зміну Vuex. Ключова відмінність від Vuex полягає у відмові від концепції мутацій: у Pinia стан змінюється безпосередньо в діях, що спрощує код та знижує кількість шаблонного коду приблизно вдвічі. Pinia підтримує інструменти розробника Vue DevTools, мову TypeScript без додаткових налаштувань та рендеринг на стороні сервера (Server-Side Rendering). У системі Pinia може використовуватися для трьох ключових сховищ: authStore (збереження JWT-токена та даних поточного користувача), cartStore (стан кошика з обчислюваними полями – загальна сума, кількість позицій) та wishlistStore (список відкладених товарів із синхронізацією між локальним сховищем браузера та API після авторизації) [24].

Tailwind CSS як CSS-фреймворк. Для стилізації інтерфейсу обрано функціональний фреймворк Tailwind CSS замість компонентно-орієнтованих бібліотек (Bootstrap, Vuetify). Підхід utility-first дозволяє безпосередньо в HTML-шаблоні описувати зовнішній вигляд елемента через набір атомарних CSS-класів (flex, gap-4, text-sm тощо), що унеможливлює конфлікти стилів між компонентами та повністю виключає потребу в іменуванні CSS-класів (проблема BEM-методології). Tailwind вбудовано виконує очищення – вилучення невикористаних стилів на етапі робочої збірки, що зменшує кінцевий розмір CSS-файлу до 5–15 КВ. Вбудований принцип mobile-first із префіксами точок зупину (sm:, md:, lg:) спрощує реалізацію адаптивного дизайну [25].

Node.js та Fastify як серверна платформа. Серверна частина побудована на платформі Node.js із використанням фреймворку Fastify. Node.js обрано завдяки однорідності мови (JavaScript/TypeScript) між клієнтською та серверною частинами, що знижує когнітивне навантаження на розробника та спрощує спільне використання типів і допоміжних інструментів. Асинхронна подієво-орієнтована модель Node.js оптимально підходить для завдань із інтенсивним введенням-виведенням (I/O) – обробки HTTP-запитів та взаємодії з базою даних, де

переважна більшість часу витрачається на очікування відповіді від зовнішніх ресурсів, а не на обчислення [26].

Fastify обрано як найбільш продуктивний мінімалістичний Node.js-фреймворк. Вбудована перевірка JSON-схем через Ajv дозволяє декларативно описувати структуру вхідних і вихідних даних кожного маршруту, а серіалізація відповідей через fast-json-stringify виконується до 5 разів швидше за стандартний JSON.stringify. Порівняння Node.js-фреймворків наведено у таблиці Б.2 Додатку Б [28-30].

PostgreSQL як система управління базами даних. Для зберігання даних обрано реляційну СУБД PostgreSQL. Вибір реляційної моделі є принциповим для інтернет-магазину, де дані мають чітко визначену структуру та численні зв'язки: замовлення → позиції замовлення → товари → категорії → користувачі. PostgreSQL забезпечує повне дотримання ACID-властивостей транзакцій, що є обов'язковою умовою для операцій оформлення замовлення, де неатомарне виконання (списання залишку + створення запису замовлення) може призвести до неузгодженості даних. Додатковими аргументами є підтримка JSONB-типу для зберігання динамічних технічних характеристик товарів, повнотекстовий пошук, матеріалізовані представлення (views) та розвинена система індексування (B-tree, GIN, GiST) [31]. Порівняння з альтернативними СУБД наведено у таблиці Б.3 Додатку Б [32-33].

Drizzle ORM. Для взаємодії з PostgreSQL обрано Drizzle ORM – відносно новий орієнтований на TypeScript інструмент об'єктно-реляційного відображення (ORM), що реалізує принцип пріоритету SQL: схеми таблиць описуються мовою TypeScript із генерацією SQL-міграцій, а запити будуються через типобезпечний конструктор запитів, що синтаксично близький до SQL. На відміну від Prisma (генерує власну абстракцію запитів) або Sequelize (слабка типізація), Drizzle не приховує SQL за непрозорою абстракцією – розробник завжди точно розуміє, який запит буде виконано. Це критично важливо для оптимізації продуктивності на рівні запитів. Drizzle також підтримує версійні перенесення

структури даних (міграції) через drizzle-kit, що забезпечує відтворюваність стану бази даних у будь-якому середовищі [34].

Docker та Docker Compose. Для ізоляції всіх компонентів системи у контейнерах обрано Docker та Docker Compose. Контейнеризація гарантує ідентичність середовища виконання на машинах розробника, у конвеєрі безперервної інтеграції та розгортання (CI/CD-пайплайні) та на робочому сервері, усуваючи клас помилок типу «на моєму комп'ютері працює» [35]. Docker Compose дозволяє декларативно описати топологію з трьох сервісів (клієнтська частина, серверна частина, база даних) у єдиному файлі docker-compose.yml, управляти залежностями між ними та налаштовувати ізольовану мережу між контейнерами [36].

У першому розділі проведено аналіз предметної області ринку товарів енергонезалежності та визначено особливості функціонування інтернет-магазинів цього напряму. Досліджено існуючі аналоги, їхні переваги та недоліки, що дозволило обґрунтувати доцільність розробки власної вебінформаційної системи. Розглянуто теоретичні засади сучасних вебзастосунків, сформовано функціональні та нефункціональні вимоги, визначено ролі користувачів і основні модулі системи. На основі аналізу обрано технологічний стек, що забезпечує продуктивність, масштабованість і безпечний супровід системи та є узгодженим і орієнтованим на вимоги розроблюваного інтернет-магазину товарів енергонезалежності.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СТРУКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Архітектура інформаційної системи

Архітектура системи визначає спосіб організації та взаємодії її структурних компонентів. Для проєктованого інтернет-магазину обрано трирівневу клієнт-серверну архітектуру, що є загальновизнаним стандартом для сучасних вебінформаційних систем та безпосередньо реалізує теоретичний принцип розмежування відповідальності (Separation of Concerns), описаний у підрозділі 1.3. Такий підхід дозволяє забезпечити модульність системи, спростити її супровід та зменшити залежність між окремими компонентами.

Система розподілена на три логічних рівні. Перший – рівень представлення – реалізований у вигляді Vue 3 SPA, що виконується в браузері користувача. Другий – рівень бізнес-логіки – реалізований як Fastify REST API на Node.js, що розгортається як окремий серверний процес. Третій – рівень даних – представлений СУБД PostgreSQL, що відповідає виключно за зберігання та цілісність даних.

Потік даних у системі відбувається таким чином: користувач взаємодіє з інтерфейсом у браузері → Vue-компонент або сховище Pinia ініціює HTTP-запит через Fetch API → Fastify-сервер приймає запит, перевіряє автентифікацію та права доступу через проміжне програмне забезпечення, перевіряє відповідність тіла запиту через JSON-схему → сервіс виконує бізнес-логіку і звертається до репозиторію → репозиторій формує типобезпечний запит до PostgreSQL через Drizzle ORM → база даних повертає результат → дані серіалізуються та передаються клієнту у форматі JSON → сховище Pinia оновлює реактивний стан → Vue автоматично оновлює відображення залежних компонентів.

2.2. Проєктування бази даних

Проєктування бази даних є одним з ключових етапів розробки інформаційної системи, оскільки від якості та повноти схеми даних залежать цілісність інформації, продуктивність запитів і простота подальшого масштабування. Для інформаційної системи інтернет-магазину товарів енергонезалежності обрано реляційну модель даних на основі СУБД PostgreSQL.

Загальна схема бази даних охоплює 13 таблиць та 3 перелічуваних типи (ENUM), що у сукупності описують усі сутності та бізнес-процеси системи: управління користувачами, товарний каталог, адресний довідник, підсистему замовлень, список бажань, систему повідомлень та журнал адміністративних дій. Спроєктована схема наведена в Додатку Б.

Таблиця users – облікові записи користувачів. Центральною сутністю системи є таблиця users, що зберігає облікові дані та контактну інформацію всіх зареєстрованих користувачів. Поле email оголошено з обмеженням UNIQUE, що гарантує неможливість реєстрації двох акаунтів на одну адресу на рівні бази даних – захист від дублювання навіть за умови паралельних запитів (race condition). Додатково це поле використовується як основний ідентифікатор користувача під час авторизації та відновлення доступу до системи. Пароль зберігається виключно у вигляді bcrypt-хешу у полі password_hash – оригінальний пароль у базі не фіксується ніколи [37], що підвищує рівень безпеки та відповідає сучасним практикам захисту персональних даних. Роль користувача визначається полем role типу user_role, що приймає значення customer (за замовчуванням) або admin, що дозволяє реалізувати розмежування прав доступу в системі. Таким чином, таблиця забезпечує базову модель автентифікації та авторизації користувачів у межах інформаційної системи. Структуру сутності описано в таблиці 2.1.

Таблиця 2.1 Структура таблиці users

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Унікальний ідентифікатор користувача
email	varchar(255)	UNIQUE, NOT NULL	Електронна адреса (логін)
password_hash	varchar(255)	NOT NULL	bcrypt-хеш пароля
first_name	varchar(100)	NOT NULL	Ім’я користувача
last_name	varchar(100)	NOT NULL	Прізвище користувача
phone	varchar(30)	NULL	Номер телефону (необов’язково)
role	user_role	DEFAULT 'customer'	Роль: customer або admin
created_at	timestamp	DEFAULT now()	Дата та час реєстрації

Таблиця categories – категорії товарів. Таблиця categories реалізує ієрархічний каталог категорій. Ключовою особливістю є самозв’язок через поле parent_id, що є зовнішнім ключем на ту саму таблицю (categories.id). Це дозволяє будувати дерево категорій довільної глибини без зміни схеми, наприклад, категорія «Генератори» може містити підкатегорії «Бензинові генератори» та «Дизельні генератори», а кожна підкатегорія – власні вкладені рівні. Для категорій верхнього рівня поле parent_id дорівнює NULL. Поле filters типу JSONB зберігає конфігурацію специфічних фільтрів, доступних у даній категорії (наприклад, для «Зарядних станцій» – фільтри за ємністю та потужністю інвертора). Поля name та slug мають обмеження UNIQUE для унеможливлення дублювання. Такий підхід забезпечує гнучкість структури каталогу та дозволяє масштабувати систему без зміни логіки бази даних. Крім того, використання ієрархічної моделі спрощує навігацію користувача та обробку категорій. Структуру сутності наведено в таблиці 2.2.

Таблиця 2.2 Структура таблиці categories

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Унікальний ідентифікатор категорії
name	varchar(100)	UNIQUE, NOT NULL	Назва категорії
slug	varchar(120)	UNIQUE, NOT NULL	URL-ідентифікатор категорії
description	text	NULL	Опис категорії
image	varchar(500)	NULL	Шлях до зображення
parent_id	int4	FK → categories	Посилання на батьківську категорію (самозв'язок)
filters	jsonb	NULL	Конфігурація фільтрів

Таблиця manufacturers – виробники. Таблиця manufacturers винесена як окрема сутність (замість зберігання назви виробника у таблиці products як текстовий рядок), що забезпечує нормалізацію: зміна назви бренду вимагає оновлення лише одного рядка, а не всіх товарів цього виробника. Поля name та slug мають обмеження UNIQUE. Структуру сутності наведено в таблиці 2.3.

Таблиця 2.3 Структура таблиці manufacturers

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Унікальний ідентифікатор виробника
name	varchar(200)	UNIQUE, NOT NULL	Назва виробника
slug	varchar(220)	UNIQUE, NOT NULL	URL-ідентифікатор виробника

Таблиця products – товарний каталог. Таблиця products є центральним елементом товарного каталогу і пов'язана зовнішніми ключами з таблицями categories та manufacturers. Поле price визначено типом numeric(10,2) для точного зберігання грошових значень без похибок округлення. Поле stock відстежує

залишок товару на складі. Поле attributes типу JSONB забезпечує гнучке зберігання технічних характеристик товарів залежно від категорії, що дозволяє уникнути надмірної кількості NULL-полів у таблиці. Поле featured використовується для позначення рекомендованих товарів, які відображаються на головній сторінці. Структуру сутності наведено в таблиці 2.4.

Таблиця 2.4 Структура таблиці products

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Унікальний ідентифікатор товару
name	varchar(255)	NOT NULL	Назва товару
slug	varchar(280)	UNIQUE, NOT NULL	URL-ідентифікатор товару
description	text	NULL	Повний опис товару
price	numeric(10,2)	NOT NULL	Ціна продажу
stock	int4	DEFAULT 0	Залишок на складі
image	varchar(500)	NULL	Шлях до зображення
category_id	int4	FK → categories	Належність до категорії
manufacturer_id	int4	FK → manufacturers	Виробник товару
featured	bool	DEFAULT false	Прапор «рекомендованого» товару
rating	numeric(2,1)	NULL	Середній рейтинг (0.0–5.0)
attributes	jsonb	NULL	Технічні характеристики
created_at	timestamp	DEFAULT now()	Дата додавання товару

Адресна підсистема (streets, buildings, rooms, addresses). Адресна інформація в системі нормалізована у чотири взаємопов’язані таблиці, що дозволяє уникнути дублювання даних та підвищити ефективність їх зберігання. Таблиця streets містить унікальний перелік вулиць, таблиця buildings пов’язана зі streets зовнішнім ключем і зберігає номери будинків, а таблиця rooms містить

номери квартир або офісів із прив’язкою до конкретного будинку. Таблиця addresses формує повну адресу як комбінацію street_id, building_id та необов’язкового room_id, що забезпечує гнучкість роботи як із приватними будинками, так і з багатоквартирними спорудами. Такий підхід дозволяє зберігати записи про вулиці та будинки лише один раз і повторно використовувати їх через систему зовнішніх ключів. У результаті таблиця addresses містить переважно числові ідентифікатори замість дубльованих текстових рядків, що суттєво скорочує обсяг даних, спрощує підтримку цілісності інформації та підвищує ефективність пошуку й групування замовлень за географічним принципом. Спроєктовані структури наведено в таблицях 2.5–2.8.

Таблиця 2.5 Структура таблиці streets

Поле	**Тип даних**	**Обмеження**	**Призначення**
id	serial4	PK, UNIQUE, NOT NULL	Ідентифікатор вулиці
name	varchar(255)	UNIQUE, NOT NULL	Назва вулиці

Таблиця 2.6 Структура таблиці buildings

Поле	**Тип даних**	**Обмеження**	**Призначення**
id	serial4	PK, NOT NULL	Ідентифікатор будинку
street_id	int4	FK → streets	Вулиця, на якій розташований будинок
number	varchar(20)	NOT NULL	Номер будинку

Таблиця 2.7 Структура таблиці rooms

Поле	**Тип даних**	**Обмеження**	**Призначення**
id	serial4	PK, NOT NULL	Ідентифікатор приміщення
building_id	int4	FK → buildings	Будинок, у якому розташоване приміщення
number	varchar(20)	NOT NULL	Номер квартири/офісу

Таблиця 2.8 Структура таблиці addresses

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Ідентифікатор адреси
street_id	int4	FK → streets	Вулиця
building_id	int4	FK → buildings	Будинок
room_id	int4	FK → rooms, NULL	Приміщення (необов’язково)

Таблиця orders – замовлення. Таблиця orders є центральним елементом підсистеми продажів, оскільки містить основну інформацію про оформлені замовлення користувачів. Поле user_id оголошено як NULL-дозволене, що дозволяє зберігати гостьові замовлення без прив’язки до облікового запису: у такому разі контактна інформація покупця фіксується у полях guest_first_name, guest_last_name та guest_phone. Для авторизованих клієнтів ці поля залишаються NULL, а дані автоматично беруться з таблиці users. Поле status типу ENUM order_status використовується для відстеження поточного стану замовлення та проходить ланцюжок pending → processing → shipped → delivered, при цьому статус cancelled може бути встановлений на будь-якому етапі. Поле total_price зберігає суму замовлення на момент оформлення, що дозволяє зберегти коректність фінансових даних навіть у разі зміни цін на товари в майбутньому. Зв’язок із адресою доставки реалізовано через зовнішній ключ address_id, що спрощує повторне використання адрес і підтримує цілісність даних. Структуру таблиці наведено в таблиці 2.9.

Таблиця 2.9 Структура таблиці orders

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Унікальний ідентифікатор замовлення
user_id	int4	FK → users, NULL	Покупець (NULL – гостьове замовлення)
status	order_status	DEFAULT 'pending'	Статус виконання замовлення
total_price	numeric(10,2)	NOT NULL	Загальна сума замовлення

Поле	Тип даних	Обмеження	Призначення
payment_method	payment_method	NOT NULL	Спосіб оплати
address_id	int4	FK → addresses	Адреса доставки
guest_first_name	varchar(100)	NULL	Ім’я гостя (для неавторизованих)
guest_last_name	varchar(100)	NULL	Прізвище гостя
guest_phone	varchar(30)	NULL	Телефон гостя
created_at	timestamp	DEFAULT now()	Дата та час оформлення замовлення

Таблиця order_items – позиції замовлення. Таблиця order_items реалізує зв’язок «багато-до-багатьох» між замовленнями і товарами через проміжну сутність та використовується для зберігання складу кожного замовлення. Структуру таблиці наведено в таблиці 2.10.

Таблиця 2.10 Структура таблиці order_items

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Ідентифікатор позиції
order_id	int4	FK → orders	Замовлення, до якого належить позиція
product_id	int4	FK → products	Товар у замовленні
quantity	int4	NOT NULL	Кількість одиниць товару
unit_price	numeric(10,2)	NOT NULL	Ціна одиниці товару на момент замовлення

Кожен рядок описує окрему товарну позицію: яким товаром (product_id), у якій кількості (quantity) та за якою ціною (unit_price) він був доданий до замовлення. Такий підхід дозволяє одному замовленню містити декілька товарів, а одному товару – бути присутнім у багатьох замовленнях. Зберігання unit_price є принципово важливим, оскільки ціна товару може змінюватися після оформлення замовлення. Фіксація вартості на момент продажу забезпечує

коректність фінансової звітності, дозволяє точно відновити історичні дані про замовлення та уникнути помилок під час формування статистики продажів.

Таблиця wishlist – список бажань. Таблиця wishlist реалізує прив'язку товарів до облікового запису користувача для збереження вподобаних позицій. Зв'язки з таблицями users та products реалізовані через зовнішні ключі. Поле created_at дозволяє сортувати список бажань за датою додавання та аналізувати попит на товари, які часто додаються до списку бажань, але рідко купуються – корисна метрика для маркетингових рішень в майбутньому. Структуру таблиці наведено в таблиці 2.11.

Таблиця 2.11 Структура таблиці wishlist

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Ідентифікатор запису
user_id	int4	FK → users	Користувач, що додав товар
product_id	int4	FK → products	Відкладений товар
created_at	timestamp	DEFAULT now()	Дата додавання до списку бажань

Таблиця audit_logs – журнал адміністративних дій. Таблиця audit_logs фіксує всі значущі операції, виконані адміністраторами системи: додавання, редагування та видалення товарів, категорій, виробників; зміни статусів замовлень. Поле details типу JSONB дозволяє зберігати структуровані дані про конкретні зміни – наприклад, старе і нове значення ціни товару. Зовнішній ключ admin_id посилається на таблицю users (де role = 'admin'), що гарантує можливість ідентифікувати виконавця кожної дії. Таблиця не підтримує операцій DELETE або UPDATE – записи журналу є незмінними після створення, що забезпечує достовірність аудиту. Структуру таблиці наведено в таблиці 2.12.

Таблиця 2.12 Структура таблиці audit_logs

Поле	Тип даних	Обмеження	Призначення
id	serial4	PK, NOT NULL	Ідентифікатор запису журналу
admin_id	int4	FK → users	Адміністратор, що виконав дію
action	varchar(50)	NOT NULL	Тип дії (CREATE, UPDATE, DELETE)
entity	varchar(50)	NOT NULL	Сутність, над якою виконано дію
entity_id	int4	NULL	Ідентифікатор конкретного запису
details	jsonb	NULL	Деталі зміни (старе/нове значення)
created_at	timestamp	DEFAULT now()	Дата та час операції

Таблиця messages – повідомлення від відвідувачів. Таблиця messages зберігає звернення, надіслані через форму зворотного зв'язку на сайті. Оскільки повідомлення можуть надходити як від авторизованих користувачів, так і від гостей, таблиця не має зовнішнього ключа на users, а email відправника фіксується безпосередньо в окремому полі. Структуру таблиці наведено в таблиці 2.13.

Таблиця 2.13 Структура таблиці messages

Поле	Тип даних	Обмеження	Призначення
message_id	serial4	PK, NOT NULL	Ідентифікатор повідомлення
email	varchar(255)	NOT NULL	Email відправника
message	text	NOT NULL	Текст повідомлення
created_at	timestamp	DEFAULT now()	Дата надходження

Перелічувані типи (ENUM). Для полів зі строго визначеним набором допустимих значень у PostgreSQL застосовано перелічувані типи ENUM. Їхнє використання порівняно зі зберіганням текстових рядків або цілих чисел

забезпечує три переваги: гарантію цілісності даних на рівні СУБД (неможливо записати значення поза переліком), читабельність SQL-запитів і логів, а також незначну оптимізацію зберігання [38]. Усі три ENUM-типи системи наведено у таблиці 2.14.

Таблиця 2.14 Перелічувані типи ENUM бази даних

Тип ENUM	**Значення**	**Таблиця**	**Призначення**
user_role	customer, admin	users	Розмежування ролей доступу в системі
order_status	pending, processing, shipped, delivered, cancelled	orders	Відстеження стану виконання замовлення
payment_method	cash, terminal, online	orders	Спосіб оплати при оформленні замовлення

Зв'язки між таблицями та обмеження цілісності. Усі міжтабличні зв'язки реалізовано через зовнішні ключі (FOREIGN KEY) із відповідними обмеженнями посилальної цілісності. У системі присутні два типи зв'язків. Зв'язки «один-до-багатьох» (1:M): users → orders, users → wishlist, users → audit_logs; categories → products; categories → categories (самозв'язок); manufacturers → products; orders → order_items; streets → buildings; buildings → rooms. Зв'язок «багато-до-багатьох» (M:M) між orders та products реалізовано через проміжну таблицю order_items, що є стандартним рішенням для реляційної моделі [39]. Кожна таблиця містить первинний ключ типу serial4 (автоінкрементне ціле число), що спрощує генерацію ідентифікаторів та забезпечує ефективну індексацію за первинним ключем. Унікальні обмеження (UNIQUE) встановлено для полів email у таблиці users, а також name та slug у таблицях categories, manufacturers – для запобігання дублюванню на рівні СУБД, незалежно від логіки застосунку.

Нормалізація та масштабованість схеми. Спроєктована схема відповідає сформульованим раніше вимогам. Нормалізація адресної підсистеми (вулиці → будинки → приміщення → адреси) є прикладом усунення часткових залежностей

і мінімізації дублювання текстових даних. Використання JSONB-полів (attributes у products, filters у categories, details у audit_logs) є виваженим відступом від суворої реляційної моделі на користь гнучкості: ці поля містять дані, структура яких суттєво відрізняється залежно від категорії товару або типу адміністративної дії, і їхня нормалізація призвела б до надмірного ускладнення схеми. PostgreSQL підтримує індексування JSONB-полів за допомогою GIN-індексів, що дозволяє ефективно виконувати пошук у межах цих структур. Схема є розширюваною: додавання нових сутностей (наприклад, таблиці відгуків, системи знижок або підписок) не вимагає модифікації наявних таблиць і може бути реалізовано через нові таблиці з зовнішніми ключами на наявні сутності.

2.3. Проєктування серверної логіки

Серверна частина системи реалізована як автономний Node.js-застосунок, розміщений у директорії server/ монорепозиторію. Проєктування серверної логіки базується на маршрутно-орієнтованій архітектурі, де кожен логічний ресурс системи оформлений у вигляді окремого модуля-плагіну Fastify із відповідним URL-префіксом.

Загальна структура серверної частини. Код організовано у два шари: db/ – підключення до PostgreSQL, схема Drizzle ORM і тестові дані; routes/ – вісім модулів, що покривають всю функціональність. Кореневий index.js реєструє плагіни, middleware та запускає сервер. Розглянемо спроєктовану структуру директорій серверної частини детальніше (рис. 2.1).

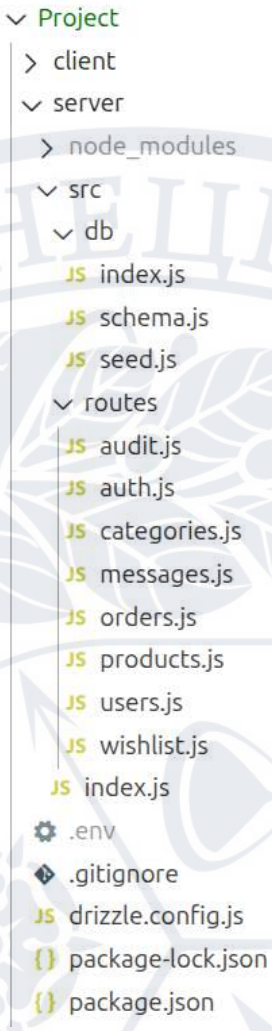


Рисунок 2.1 – Структура директорій серверної частини системи

Ініціалізація та порядок завантаження сервера. Сервер ініціалізується у строго визначеній послідовності. Спершу підключається CORS-плагін для обробки cross-origin запитів, далі – JWT-плагін із секретним ключем із змінної оточення. На екземпляр Fastify декоруються три middleware-функції: authenticate (обов'язкова перевірка токена), optionalAuth (для підтримки гостьового checkout) та authorizeAdmin (перевірка JWT і ролі). Завершує ініціалізацію реєстрація восьми груп маршрутів із відповідними префіксами.

Система автентифікації та авторизації. Автентифікація будується на принципі stateless JWT: сервер не зберігає сесії, а кожен захищений запит верифікується самостійно за підписом токена. JWT-токен містить мінімально необхідний payload – ідентифікатор, email та роль користувача – і має термін дії 7 днів. Авторизація реалізується через рольову модель (RBAC): звичайні

маршрути доступні будь-якому автентифікованому користувачу, адміністративні – лише тим, у чийому токені роль дорівнює admin. Middleware-декоратори підключаються до маршрутів декларативно через опцію preHandler, що дозволяє чітко бачити рівень захисту кожного ендпоінта безпосередньо в його описі [40].

Процес реєстрації включає перевірку унікальності email на рівні БД (не лише на рівні застосунку), хешування пароля алгоритмом bcrypt та автоматичне повернення JWT-токена разом з даними новоствореного користувача. При вході система порівнює пароль з хешем через bcrypt.compare і у разі помилки повертає однакове повідомлення незалежно від того, чи невірний email, чи пароль – це унеможливлює атаки перебором за існуванням акаунту.

Структура маршрутів API охоплюватиме 32 ендпоінти, розподілених за вісьмома ресурсами з чітко визначеним рівнем доступу – від публічних до суворо адміністративних, із підтримкою опційної автентифікації там, де потрібна гнучкість відображення.

Логіка обробки запитів на прикладі каталогу товарів. Маршрут GET /api/products – найскладніший маршрут системи. Для кожного відомого query-параметра (categoryId, search, minPrice тощо) до SQL-запиту додаватиметься відповідна умова. Невідомі параметри інтерпретуватимуться як динамічні атрибутні фільтри й транслюватимуться в умови вибірки по JSONB-полю attributes. Це дозволить підкатегоріям мати специфічні фільтри без змін у серверному коді. Фільтрація за батьківською категорією автоматично розширюватиметься на підкатегорії через підзапит.

Логіка оформлення замовлення. Маршрут POST /api/orders реалізує найбільш складний бізнес-сценарій і застосовує кілька рівнів перевірок. Спершу – семантична валідація структури товарів, адресних полів, методу оплати та контактних даних гостя. Далі – нормалізація адреси за стратегією «знайти або створити» для кожного рівня ієрархії, що запобігає дублюванню записів. Після цього ціни отримуватимуться безпосередньо з БД (клієнтські ціни ігноруються), перевіряються залишки. Фінальний запис до orders та order_items разом із декрементом залишків виконуватиметься в межах однієї транзакції.

Журнал аудиту. Всі адміністративні операції над ресурсами (створення, редагування, видалення товарів і категорій, зміна статусів замовлень) автоматично фіксуються в таблиці audit_logs після успішного виконання. Запис містить ідентифікатор адміністратора, тип дії, назву сутності, ідентифікатор конкретного запису та JSONB-деталі змін. Таблиця призначена виключно для запису – жодних операцій оновлення або видалення записів журналу не передбачаються.

Обробка помилок. Серверна логіка реалізує чотири рівні захисту від некоректних даних: перевірку наявності обов'язкових полів з поверненням 400; семантичну перевірку значень і форматів (email, телефон, enum-значення) з поверненням 400; перевірку бізнес-правил (унікальність email, дублікати у списку бажань, достатність залишків) з поверненням 409 або 422; та перевірку прав доступу через middleware з поверненням 401 або 403. Відповіді про помилки уніфіковані у форматі { error: "Повідомлення" }, що спрощує їхню обробку на клієнті.

2.4. Проєктування клієнтської частини

Клієнтська частина системи спроєктована як Single Page Application (SPA) на основі Vue 3 і розміщена у директорії client. Вся взаємодія з сервером відбувається через REST API без перезавантаження сторінки, що забезпечує плавну та реактивну поведінку інтерфейсу. Структура клієнтської частини розподілена на п'ять логічних шарів: централізований HTTP-клієнт, маршрутизація, глобальне управління станом, перевикористовувані компоненти та сторінки-відображення маршрутів. Розглянемо спроєктовану структуру директорій клієнтської частини детальніше (рис. 2.2).

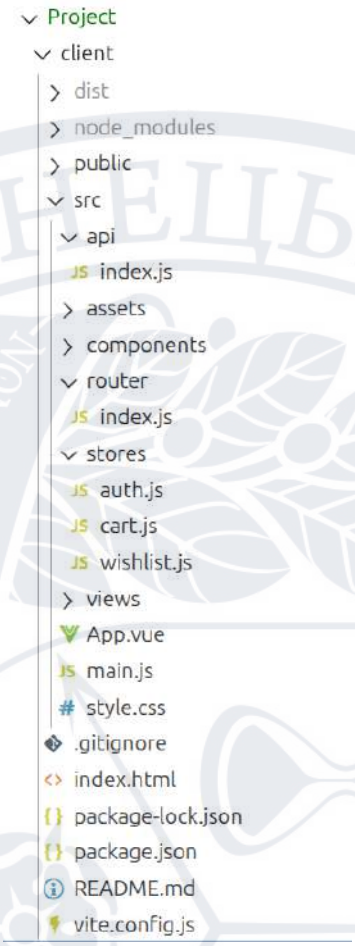


Рисунок 2.2 – Структура директорій клієнтської частини системи

Централізований HTTP-клієнт. Мережева взаємодія між клієнтом і сервером централізована у файлі api/index.js, який виступає єдиною точкою доступу до REST API. Реалізована функція-обгортка автоматично додає заголовки авторизації (Bearer JWT), виконує серіалізацію тіла запиту та забезпечує уніфіковану обробку відповідей і помилок. Централізація мережевого шару унеможливлює дублювання логіки в компонентах і дозволяє змінювати базову URL або схему автентифікації в одному місці без модифікації решти коду.

Маршрутизація та навігаційні guards. Vue Router 4 налаштовується в режимі HTML5 History API для формування семантичних URL без символу #. Кожен маршрут декларуватиметься з посиланням на компонент-сторінку та мета-атрибутом рівня доступу. Контроль переходів забезпечуватиме глобальний guard beforeEach, що звірятиме мета-атрибут із поточним станом автентифікації: маршрути з requiresAuth перенаправлятимуть неавторизованих користувачів на /register; з requiresAdmin – блокуватимуть доступ для не-адміністраторів; з

позначкою guest – перенаправлятимуть вже авторизованих на головну сторінку, унеможливлюючи повторний вхід. Повний перелік маршрутів наведено в Додатку Г.

Управління глобальним станом через Pinia. Глобальний стан структурується засобами Pinia і розподіляється між трьома незалежними сторами за предметними областями.

Стор автентифікації (auth.js) виступає центральним сховищем сесії: JWT-токен зберігатиметься одночасно у реактивному стані та localStorage для відновлення сесії між перезавантаженнями. При ініціалізації застосунку викликатиметься fetchUser(), який перевірятиме актуальність токена через GET /api/auth/me; за недійсного токена виконуватиметься автоматичний logout. Стор кошика (cart.js) реалізовуватиме повністю клієнтський кошик із персистентністю через localStorage – незалежно від стану автентифікації. Стор списку бажань (wishlist.js) підтримуватиме два режими завантаження: легке – лише масив ідентифікаторів для відображення стану в картках каталогу, та повне – з даними товарів для відповідної сторінки.

Тематизація та глобальні стилі. Дві колірні теми реалізовуватимуться через CSS-змінні, оголошені у двох наборах: базовий набір токенів та їхнє перевизначення у селекторі .dark. Перемикання зводитиметься до додавання або видалення класу dark на елементі html без участі JavaScript у зміні кольорів. Вибір теми зберігатиметься у localStorage і відновлюватиметься при завантаженні. Tailwind CSS застосовуватиметься в режимі utility-first, що унеможливлює конфлікти імен класів між компонентами.

Навігаційна панель. Фіксована верхня панель забезпечуватиме постійний доступ до ключових розділів. Вона інтегруватиме логотип як точку повернення на головну сторінку, адаптивне меню категорій, рядок пошуку з підтримкою швидких підказок, перемикач тем, а також індикатори стану кошика та профілю. Панель проєктується за принципом mobile-first із збереженням повної навігаційної логіки в обох форматах відображення.

Проєктування ключових сторінок. Головна сторінка (Home.vue) структурується із семи тематичних секцій: hero-блок із навігацією категорій, блок Flash Sale з таймером, бестселери, промо-банер, сітка нових товарів, блок нових надходжень і секція переваг. Три незалежні API-запити виконуватимуться паралельно через Promise.all для мінімізації часу завантаження.

Сторінка каталогу (Products.vue) реалізовуватиме повну фасетну фільтрацію: дворівневу навігацію за категоріями, ціновий слайдер із динамічним діапазоном, множинний вибір виробників і динамічні атрибутні фільтри, що десеріалізуватимуться з поля filters обраної категорії. Зміна будь-якого фільтра реактивно оновлюватиме список через watch-спостерігачі Vue 3.

Сторінка товару (ProductDetail.vue) використовуватиме обчислювану властивість availableStock = product.stock – кількість у кошику для коректного відображення фактичного залишку з урахуванням вже доданих позицій.

Сторінка кошика (Cart.vue) суміщатиме управління позиціями та форму оформлення в одному компоненті. Форма структурується на три логічні блоки – контактні дані, адреса доставки, спосіб оплати – і попередньо заповнюватиметься збереженими даними авторизованого користувача. Адміністративні сторінки реалізовуватимуть CRUD-операції через таблиці з модальними вікнами для повного редагування та inline-редагуванням для швидкої зміни статусів.

Адаптивність та UX-рішення. Адаптивний дизайн реалізовуватиметься через breakpoint-утиліти Tailwind CSS за принципом mobile-first: сітки товарів змінюватимуться від одної колонки на мобільних до чотирьох на desktop; форми від двоколонкового до одноколонкового макету. Серед ключових UX-рішень передбачаються: збереження стану кошика між сесіями; підтримка гостьового оформлення замовлення; пошук із debounce і кешуванням; стани порожнього вмісту з CTA-підказками на всіх списках; блокування повторних кліків під час обробки запиту.

Таким чином спроєктовано повну архітектуру інформаційної системи інтернет-магазину товарів енергонезалежності. Визначено трирівневу клієнт-

серверну структуру системи, розроблено схему бази даних PostgreSQL із механізмами забезпечення цілісності та масштабованості, спроєктовано серверну логіку на основі Fastify і REST API з підтримкою JWT-автентифікації та рольової авторизації, а також реалізовано структуру клієнтської SPA-частини на Vue 3 з глобальним управлінням станом, адаптивним інтерфейсом і підтримкою сучасних UX-рішень.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Реалізація серверної частини

Серверна частина інформаційної системи реалізована як Node.js-застосунок з використанням ES Modules. Код розміщено у директорії server/ та поділено на два основні шари: підключення до бази даних (db/) і маршрути з обробниками (routes/). Точка входу – src/index.js – відповідає за ініціалізацію компонентів і запуск HTTP-сервера.

Конфігурація середовища винесена у файл .env:

```
DATABASE_URL=postgres://user:password@localhost:5432/energyshop
JWT_SECRET=your_jwt_secret_key
PORT=3000
HOST=0.0.0.0
```

Використання змінних середовища дозволяє запускати застосунок у різних середовищах без змін у коді, змінюючи лише конфігурацію.

Підключення до бази даних. Модуль db/index.js реалізує ініціалізацію з'єднання з PostgreSQL та створення екземпляра Drizzle ORM, що використовується у всіх маршрутних обробниках:

```
import { drizzle } from "drizzle-orm/postgres-js";
import postgres from "postgres";
import * as schema from "./schema.js";

const client = postgres(process.env.DATABASE_URL);
export const db = drizzle(client, { schema });
```

Бібліотека postgres створює пул з'єднань через DATABASE_URL. Drizzle отримує пул і schema та формує ORM-об'єкт. Експортований db використовується у всіх модулях без створення нових підключень.

Ініціалізація Fastify та реєстрація плагінів. У файлі src/index.js створюється екземпляр Fastify та реєструються системні плагіни. CORS дозволяє

запити з будь-якого домену для роботи фронтенду, а JWT-плагін використовує секрет із .env для автентифікації:

```
await app.register(cors, { origin: true });

await app.register(fjwt, {
  secret: process.env.JWT_SECRET,
});
```

Middleware-декоратори автентифікації та авторизації. Після реєстрації плагінів до екземпляра Fastify додаються три middleware-функції у вигляді декораторів, що реалізують систему контролю доступу:

```
app.decorate("authenticate", async (request, reply) => {
  try {
    await request.jwtVerify();
  } catch {
    reply.code(401).send({ error: "Unauthorized" });
  }
});

app.decorate("optionalAuth", async (request) => {
  try {
    await request.jwtVerify();
  } catch {
  }
});

app.decorate("authorizeAdmin", async (request, reply) => {
  try {
    await request.jwtVerify();
    if (request.user.role !== "admin") {
      reply.code(403).send({ error: "Forbidden" });
    }
  } catch {
    reply.code(401).send({ error: "Unauthorized" });
  }
});
```

Декоратор authenticate верифікує JWT через request.jwtVerify(), повертаючи 401 при помилці або записуючи дані в request.user при успіху. optionalAuth працює аналогічно, але не блокує запит у разі невдачі, дозволяючи доступ гостям. authorizeAdmin забезпечує дворівневий захист: спочатку перевіряє токен (401), а потім наявність ролі 'admin', повертаючи 403 за її відсутності. Усі перевірки

підключаються декларативно через onRequest або preHandler прямо у визначенні маршруту, що робить архітектуру безпеки прозорою та зрозумілою.

Реєстрація маршрутних модулів. Після ініціалізації middleware всі вісім маршрутних модулів реєструються як Fastify-плагіни з відповідними URL-префіксами, наприклад:

```
await app.register(authRoutes,     { prefix: "/api/auth" });
await app.register(productRoutes,  { prefix: "/api/products" });
```

Префікс автоматично додається до всіх маршрутів модуля, забезпечуючи ізоляцію та структурованість API.

Запуск HTTP-сервера. Завершальний блок index.js запускає HTTP-сервер на параметрах, отриманих зі змінних середовища:

```
const port = Number(process.env.PORT) || 3000;
const host = process.env.HOST || "0.0.0.0";

try {
  await app.listen({ port, host });
} catch (err) {
  app.log.error(err);
  process.exit(1);
}
```

Хост 0.0.0.0 означає прослуховування на всіх мережевих інтерфейсах, що необхідно для доступу до сервера з Docker-контейнера або іншої машини в локальній мережі. При помилці запуску (зайнятий порт, недоступна база даних) застосунок фіксує деталі помилки через вбудований logger та завершує процес з кодом 1, що сигналізує операційній системі або Docker про невдалий запуск.

Реалізація маршрутів автентифікації. Модуль routes/auth.js оголошує три маршрути: реєстрацію нового користувача (POST /register), вхід в систему (POST /login) та отримання даних поточного користувача (GET /me). Усі маршрути реалізовані як методи об’єкта Fastify-плагіна, що приймає екземпляр app.

Реєстрація (POST /api/auth/register). Обробник послідовно виконує такі кроки:

1. Валідація обов'язкових полів (повертає код 400 Bad Request при помилці).
2. Перевірка унікальності email через явний запит до БД (повертає 409 Conflict, запобігаючи необробленим помилкам бази даних).
3. Хешування пароля функцією bcrypt.hash (cost factor 10) для неможливості його відновлення.
4. Збереження в БД за допомогою Drizzle ORM. Метод .returning() одразу повертає дані створеного користувача, гарантовано виключаючи хеш пароля (passwordHash), щоб він не потрапив до клієнта.
5. Генерація та повернення JWT-токена з терміном дії 7 днів.

Вхід в систему (POST /api/auth/login). Ключова логіка полягає у перевірці наявності користувача в базі та порівнянні паролів через bcrypt.compare. Як при неіснуючому email, так і при неправильному паролі навмисно повертається ідентичне повідомлення Invalid credentials (код 401). Ця уніфікація запобігає перебору (enumeration) зареєстрованих адрес зловмисниками. У разі успіху генерується JWT-токен.

Отримання даних поточного користувача (GET /api/auth/me). Маршрут захищений декоратором authenticate. Ідентифікатор користувача береться з payload верифікованого токена (request.user.id) і використовується для SELECT-запиту до бази даних. Це критично важливо: система повертає свіжі дані з БД, а не просто дані з токена (на випадок, якщо роль користувача змінилася після авторизації). Поле passwordHash знову ж таки виключається з вибірки.

Реалізована серверна частина забезпечує повний ланцюжок ініціалізації: від конфігурації та БД до запуску HTTP-сервера. Модуль автентифікації реалізує багаторівневу валідацію, безпечне зберігання паролів та JWT-авторизацію, що є надійним фундаментом для захисту всієї системи.

3.2. Реалізація схеми та міграцій бази даних

Схема бази даних реалізована декларативно у файлі server/src/db/schema.js засобами Drizzle ORM. Такий підхід означає, що структура таблиць описується як JavaScript-об’єкти з повною типізацією TypeScript – Drizzle транслює їх у відповідний DDL PostgreSQL при генерації міграцій. Це усуває розрив між кодом застосунку та станом бази даних: схема є єдиним джерелом правди для обох.

Оголошення ENUM-типів. На початку файлу за допомогою функції pgEnum оголошуються три перелічувані типи, що відповідають типам PostgreSQL, спроєктованим у підрозділі 2.3:

```
export const userRoleEnum = pgEnum("user_role", ["customer",
"admin"]);

export const orderStatusEnum = pgEnum("order_status", [
  "pending", "processing", "shipped", "delivered", "cancelled"]);

export const paymentMethodEnum = pgEnum("payment_method", [
  "cash", "terminal", "online"]);
```

Функція pgEnum генерує тип безпосередньо у схемі PostgreSQL, а не емулює його на рівні застосунку. Це означає, що база даних відхиляє будь-яке значення, що не входить до переліку, ще до того, як воно потрапить у бізнес-логіку.

Таблиця users. Оголошення таблиці користувачів демонструє типовий патерн Drizzle: параметри кожної колонки визначаються через ланцюгові виклики методів, що встановлюють типи даних, обмеження та значення за замовчуванням:

```
export const users = pgTable("users", {
  id:           serial("id").primaryKey(),
  email:        varchar("email", { length: 255
}).notNull().unique(),
  passwordHash: varchar("password_hash", { length: 255
}).notNull(),
  firstName:    varchar("first_name",  { length: 100 }).notNull(),
  lastName:     varchar("last_name",   { length: 100 }).notNull(),
  phone:        varchar("phone",       { length: 30  }),
```

```
  role:
userRoleEnum("role").default("customer").notNull(),
  createdAt:    timestamp("created_at").defaultNow().notNull(),
});
```

Методи .unique(), .notNull() та .defaultNow() забезпечують автоматичну трансляцію у відповідні SQL-обмеження: UNIQUE, NOT NULL та DEFAULT now(). Важливою особливістю реалізації є розмежування іменування: використання стилю camelCase для об'єктів JavaScript (passwordHash) та snake_case для колонок бази даних (password_hash). Таке відображення, реалізоване через аргументи методів, дозволяє дотримуватися конвенцій іменування обох середовищ, забезпечуючи високу читабельність коду та сумісність зі стандартами PostgreSQL [42].

Конфігурація Drizzle Kit та процес міграцій. Файл drizzle.config.js налаштовує утиліту drizzle-kit, що відповідає за генерацію та застосування міграцій. Основне призначення цього модуля полягає у забезпеченні синхронізації між декларативною схемою даних у коді та фізичною структурою бази даних PostgreSQL. Через параметр schema визначається шлях до файлів оголошення сутностей, а параметр out вказує на директорію для зберігання SQL-файлів міграцій. Такий підхід дозволяє реалізувати версіонування бази даних: система аналізує зміни у вихідному коді, порівнює їх із поточним станом та формує інкрементальні оновлення, що забезпечує стабільність структури під час розгортання в різних середовищах:

```
export default defineConfig({
  dialect:       "postgresql",
  schema:        "./src/db/schema.js",
  out:           "./drizzle",
  dbCredentials: { url: process.env.DATABASE_URL },
});
```

Утиліта аналізує файл схеми, порівнює його з поточним станом і формує нові інструкції лише для внесених змін. Після цього всі оновлення послідовно вносяться в базу даних, де кожна операція реєструється у службовій таблиці для запобігання помилкам. Такий механізм забезпечує цілісність даних і дозволяє

легко розгортати проєкт у будь-якому середовищі, підтримуючи повну відповідність між кодом застосунку та структурою збереженої інформації.

3.3. Реалізація маршрутних модулів

Модуль products.js: динамічна фільтрація каталогу. Маршрут GET /api/products є одним із найскладніших обробників у системі, оскільки забезпечує гнучкий механізм пошуку та відбору товарів за множиною параметрів. Він підтримує одночасне застосування до семи різних типів фільтрів, які формуються на основі вхідних параметрів запиту та поступово накопичуються в масиві умов conditions. У подальшому всі сформовані умови об’єднуються за допомогою логічного оператора AND і передаються в єдиний SQL-запит, що дозволяє виконувати комплексну фільтрацію на рівні бази даних без додаткової обробки на стороні сервера.

Фільтрація за категорією враховує ієрархічну структуру каталогу: під час передачі ідентифікатора батьківської категорії система автоматично включає до вибірки товари з усіх її підкатегорій:

```
if (categoryId) {
  const subcats = await db
    .select({ id: categories.id })
    .from(categories)
    .where(eq(categories.parentId, Number(categoryId)));

  const catIds = [Number(categoryId), ...subcats.map((c) =>
c.id)];
  conditions.push(inArray(products.categoryId, catIds));
}
```

Окрему увагу заслуговує механізм динамічних атрибутних фільтрів. Будь-який параметр запиту, що не відповідає зарезервованим іменам (categoryId, search, featured тощо), інтерпретується як фільтр за JSONB-полем attributes таблиці products. Для одного значення застосовується порівняння рівності, для декількох (розділених комою) – оператор ANY:

```
for (const [key, value] of Object.entries(rest)) {
  const vals = String(value).split(",");
  if (vals.length === 1) {
    conditions.push(
      sql`${products.attributes}->>${sql.raw(`'${key}'`)} =
${vals[0]}`
    );
  } else {
    conditions.push(
      sql`${products.attributes}->>${sql.raw(`'${key}'`)} =
ANY(${vals})`
    );
  }
}
```

Завдяки цьому рішенню кожна підкатегорія може мати власні специфічні фільтри без жодних змін у серверному коді: конфігурація фільтрів зберігається у JSONB-полі filters таблиці categories, а клієнт динамічно формує відповідні параметри запиту.

Фінальний запит виконується паралельно для двох операцій – отримання сторінки товарів та підрахунку загальної кількості записів – через Promise.all. Це скорочує час відповіді порівняно з послідовним виконанням запитів:

```
const [items, countResult] = await Promise.all([
  db.select().from(products)

.where(where).limit(Number(limit)).offset(offset).orderBy(orderCla
use),
  db.select({ count: sql`count(*)::int` })
    .from(products).where(where),
]);

return {
  items,
  total:      countResult[0].count,
  page:       Number(page),
  totalPages: Math.ceil(countResult[0].count / Number(limit)),
};
```

Відповідь містить масив товарів поточної сторінки, загальну кількість записів, номер сторінки та кількість сторінок. Така структура дозволяє клієнту реалізувати повноцінну пагінацію без потреби у додаткових запитах до сервера.

Модуль categories.js: ієрархічне дерево та аудит. Маршрут GET /api/categories/tree формує дворівневу ієрархію категорій в пам'яті застосунку замість рекурсивного SQL-запиту. Це обґрунтовано невеликою кількістю категорій (десятки, а не тисячі записів):

```
app.get("/tree", async () => {
  const all = await db.select().from(categories);
  const parents = all.filter((c) => !c.parentId);
  return parents.map((p) => ({
    ...p,
    children: all.filter((c) => c.parentId === p.id),
  }));
});
```

Усі адміністративні маршрути (POST, PUT, DELETE) після успішної операції з базою даних автоматично записують подію до журналу аудиту. Підхід однаковий для всіх ресурсів: спочатку виконується основна операція з поверненням .returning(), потім – вставка запису до таблиці audit_logs:

```
await db.insert(auditLogs).values({
  adminId: request.user.id,
  action:   "create",        // "update" | "delete" |
"update_status"
  entity:   "product",
  entityId: product.id,
  details:  { name: product.name },
});
```

Поле details типу JSONB зберігає контекстно-залежну інформацію: для операції create – назву створеного об'єкта, для update – перелік змінених полів, для update_status – новий статус замовлення. Такий підхід забезпечує повну простежуваність адміністративних дій без змін у структурі таблиці.

Модуль orders.js: оформлення замовлення. Маршрут POST /api/orders є критично важливим бізнес-процесом і реалізує найбільш складний ланцюжок операцій. Він захищений декоратором optionalAuth, що підтримує як авторизованих користувачів, так і гостей. Обробник складається з чотирьох послідовних етапів.

Перший етап – валідація вхідних даних: перевірка наявності товарів у масиві items, обов'язковості адресних полів та відповідності paymentMethod допустимому переліку ENUM. Для гостей додатково перевіряється наявність і формат контактних полів.

Другий етап – нормалізація адреси через функцію resolveAddress, що реалізує підхід «знайти або створити» для кожного рівня адресної ієрархії:

```
async function resolveAddress({ street, building, room }) {
  let [streetRow] = await db.select().from(streets)
    .where(eq(streets.name, street)).limit(1);
  if (!streetRow) {
    [streetRow] = await db.insert(streets).values({ name: street
}).returning();
  }
  let [buildingRow] = await db.select().from(buildings)
    .where(and(eq(buildings.streetId, streetRow.id),
               eq(buildings.number, building))).limit(1);
  if (!buildingRow) {
    [buildingRow] = await db.insert(buildings)
      .values({ streetId: streetRow.id, number: building
}).returning();
  }
  return addressRow.id;
}
```

Функція послідовно проходить чотири рівні: вулиця → будинок → кімната (необов'язково) → адреса. На кожному рівні спочатку виконується SELECT, і лише за відсутності запису – INSERT. Це гарантує унікальність адресних записів: якщо десятки клієнтів оформлять замовлення на ту саму адресу, в таблиці addresses буде лише один запис.

Третій етап – перевірка залишків та розрахунок суми. Ціни отримуються безпосередньо з бази даних (значення від клієнта ігноруються), після чого для кожної позиції перевіряється достатність залишку:

```
for (const item of items) {
  const product = priceMap[item.productId];
  if (!product) {
    return reply.code(400).send({ error: `Product
${item.productId} not found` });
  }
  if (product.stock < item.quantity) {
    return reply.code(400).send({
```

```
      error: `Insufficient stock for ${product.name}`
    });
  }
  totalPrice += Number(product.price) * item.quantity;
}
```

Четвертий етап – збереження замовлення. Залежно від наявності авторизації в об’єкт orderData записується або userId, або гостьові контактні поля. Після вставки замовлення та позицій виконується декремент залишків товарів на складі, який реалізований через атомарний SQL-вираз stock - quantity безпосередньо на рівні бази даних, що унеможливлює ситуацію гонки (race condition) між паралельними запитами на купівлю одного товару та забезпечує цілісність даних у системі:

```
const [order] = await
db.insert(orders).values(orderData).returning();

await db.insert(orderItems).values(
  orderItemsData.map((oi) => ({ ...oi, orderId: order.id }))
);

for (const item of items) {
  await db.update(products)
    .set({ stock: sql`${products.stock} - ${item.quantity}` })
    .where(eq(products.id, item.productId));
}
```

Модуль wishlist.js: оптимізоване завантаження. Модуль реалізує два режими отримання даних списку бажань. Маршрут GET /api/wishlist повертає повні об’єкти із вкладеними даними товарів через INNER JOIN – він використовується на сторінці обраного. Маршрут GET /api/wishlist/ids повертає лише масив ідентифікаторів:

```
app.get("/ids", { preHandler: [app.authenticate] }, async
(request) => {
  const rows = await db
    .select({ productId: wishlist.productId })
    .from(wishlist)
    .where(eq(wishlist.userId, request.user.id));

  return rows.map((r) => r.productId);
});
```

Такий поділ є свідомим UX-рішенням: при завантаженні каталогу клієнт спочатку отримує лише масив id і відразу може відображати стан кнопок «вже в обраному» на картках товарів – без очікування завантаження повних даних усіх відкладених товарів. Після додавання товару до списку бажань перевіряється дублікат на рівні сервера, і у разі повторного запиту повертається 409 Conflict.

Модулі messages.js, users.js та audit.js. Модуль messages.js реалізує публічну точку доступу для форми зворотного зв'язку з валідацією формату email через регулярний вираз та захищений адміністративний маршрут перегляду повідомлень. Модуль users.js реалізує функціонал редагування профілю (PUT /api/users/profile) з явним зазначенням полів, що дозволено оновлювати (firstName, lastName, phone), поле email залишається незмінним. Адміністративні маршрути повертають повні дані користувача разом з його історією замовлень із деталізацією товарними позиціями через INNER JOIN. Модуль audit.js надає єдиний маршрут GET /api/audit з пагінацією та INNER JOIN на таблицю users для відображення email адміністратора поруч із кожним записом журналу.

3.4. Клієнтська частина

Точка входу та ініціалізація (main.js, App.vue). Файл main.js виконує послідовну реєстрацію плагінів:

```
const app = createApp(App)
app.use(createPinia())
app.use(router)
app.mount('#app')
```

Послідовність ініціалізації Pinia перед Router зумовлена тим, що навігаційні фільтри в router/index.js використовують стан authStore. При монтуванні App.vue виконується відновлення сесії користувача через запит GET /api/auth/me із використанням збереженого токена, а у випадку помилки – автоматичне завершення сеансу. Також відновлюється обрана користувачем

колірна тема шляхом зчитування значення з localStorage та застосування класу dark до елемента html.

Централізований HTTP-клієнт (api/index.js). Функція request є єдиною точкою мережевої взаємодії:

```
async function request(path, options = {}) {
  const token = localStorage.getItem("token");
  const headers = { "Content-Type": "application/json",
...options.headers };
  if (token) headers.Authorization = `Bearer ${token}`;

  const res = await fetch(`${BASE}${path}`, { ...options, headers
});
  if (res.status === 204) return null;
  const data = await res.json();
  if (!res.ok) throw { status: res.status, ...data };
  return data;
}
```

Функція автоматично додає до кожного HTTP-запиту заголовок авторизації Authorization: Bearer <token> за наявності токена в localStorage. Окремо реалізовано обробку відповіді зі статусом 204 No Content, для якої повертається значення null без спроби десеріалізації порожнього тіла відповіді. У випадку отримання non-ok статусу генерується об’єкт помилки, що містить HTTP-статус та серверне повідомлення, завдяки чому компоненти інтерфейсу можуть напряму відображати релевантний текст помилки без додаткової логіки перевірки.

Маршрутизація та контроль доступу (router/index.js). Захист реалізований через єдиний глобальний навігаційний guard:

```
router.beforeEach((to) => {
  const auth = useAuthStore();
if (to.meta.requiresAuth && !auth.isAuthenticated)
return "/register";
if (to.meta.requiresAdmin && auth.user?.role !== "admin")
return "/";
if (to.meta.guest && auth.isAuthenticated)
return "/";
});
```

Навігаційний guard обробляє маршрути requiresAuth, requiresAdmin і guest, виконуючи відповідні перенаправлення залежно від стану авторизації та ролі

користувача. Також реалізовано автоматичний перехід до верхньої частини сторінки або плавний скрол до якоря за хешем URL.

Pinia-стор автентифікації (stores/auth.js). Токен синхронно зчитується з localStorage, завдяки чому стан isAuthenticated визначається ще до виконання асинхронних запитів:

```
const token = ref(localStorage.getItem("token"));
const isAuthenticated = computed(() => !!token.value);

async function fetchUser() {
  if (!token.value) return;
  try   { user.value = await api.get("/auth/me"); }
  catch { logout(); }
}
function logout() {
  token.value = null; user.value = null;
  localStorage.removeItem("token");
}
```

Функція fetchUser() виконує перевірку токена через сервер, а при помилці logout() очищує реактивний стан і localStorage. Доступ до маршрутів requiresAdmin контролюється через обчислювану властивість isAdmin.

Pinia-стор кошика (stores/cart.js). Кошик ініціалізується з localStorage та зберігає стан між перезавантаженнями незалежно від авторизації:

```
const items = ref(JSON.parse(localStorage.getItem("cart") ||
"[]"));

function addItem(product, quantity = 1) {
  const existing = items.value.find((i) => i.productId ===
product.id);
  if (existing) {
    existing.quantity = Math.min(existing.quantity + quantity,
product.stock);
  } else {
    items.value.push({ productId: product.id, name: product.name,
      price: product.price, stock: product.stock,
      quantity: Math.min(quantity, product.stock) });}
  localStorage.setItem("cart", JSON.stringify(items.value)); //
persist}
```

Функція addItem() обмежує кількість товару значенням product.stock, запобігаючи перевищенню доступного залишку на клієнтському рівні. Стор

списку бажань (stores/wishlist.js) синхронізується із сервером через POST і DELETE, одразу оновлюючи локальний стан без повторного запиту.

Головна сторінка та навігація

Головна сторінка відображає рекомендовані товари, категорії та рекламні блоки (рис. 3.1). Компонент навігаційної панелі реалізує оптимізований пошук із трьома механізмами продуктивності: затримкою введення (debounce) у 400 мс, кешуванням результатів у пам’яті (до 50 записів) та скасуванням попередніх незавершених запитів через AbortController. Це запобігає зайвим мережевим зверненням і перезапису актуальних результатів старішими відповідями. Також підтримується навігація клавіатурою (стрілки, Enter, Escape).

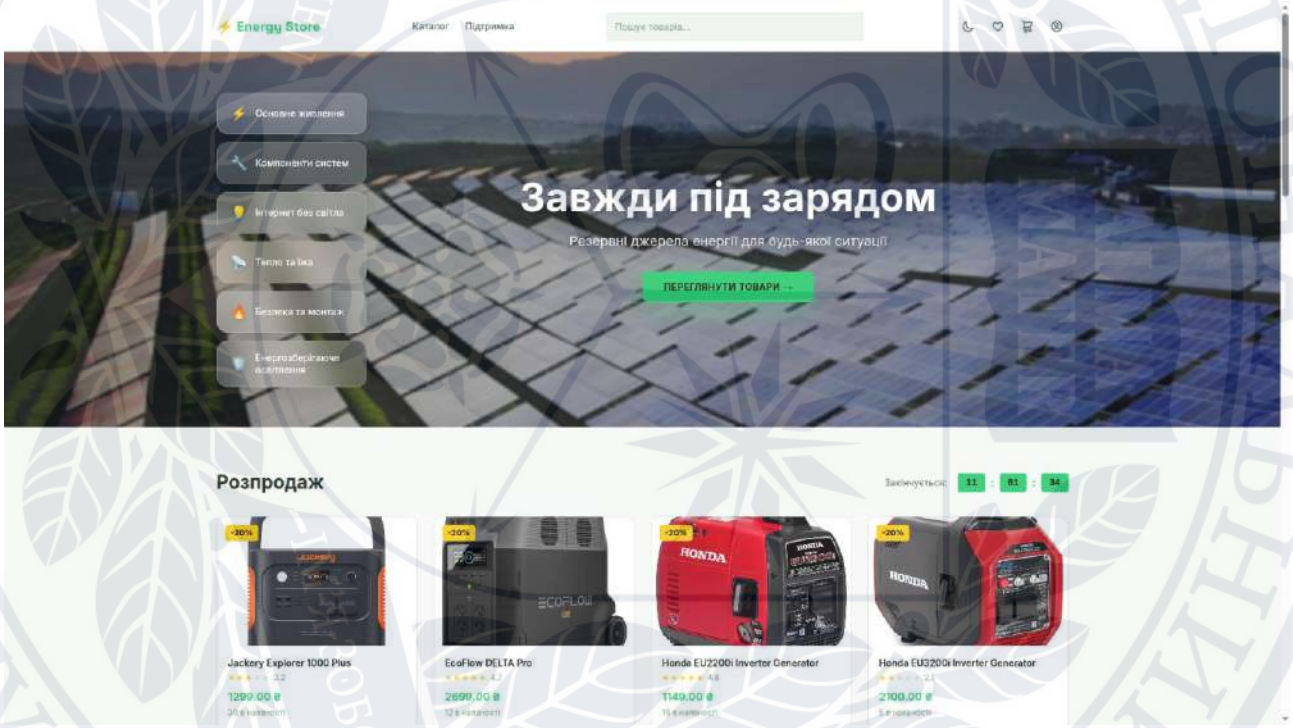


Рисунок 3.1 – Головна сторінка інтернет-магазину

Каталог товарів

Сторінка каталогу забезпечує фільтрацію та сортування товарів (рис. 3.2). Фільтри динамічно генеруються на основі JSONB-конфігурації категорії, яку налаштовує адміністратор, а клієнтський код автоматично створює відповідні елементи форми без змін у коді. Після зміни параметрів пошуку або сортування сторінка скидається до першої, а список товарів оновлюється. Під час зміни

категорії додатково виконуються запити для отримання діапазону цін, виробників і товарів.

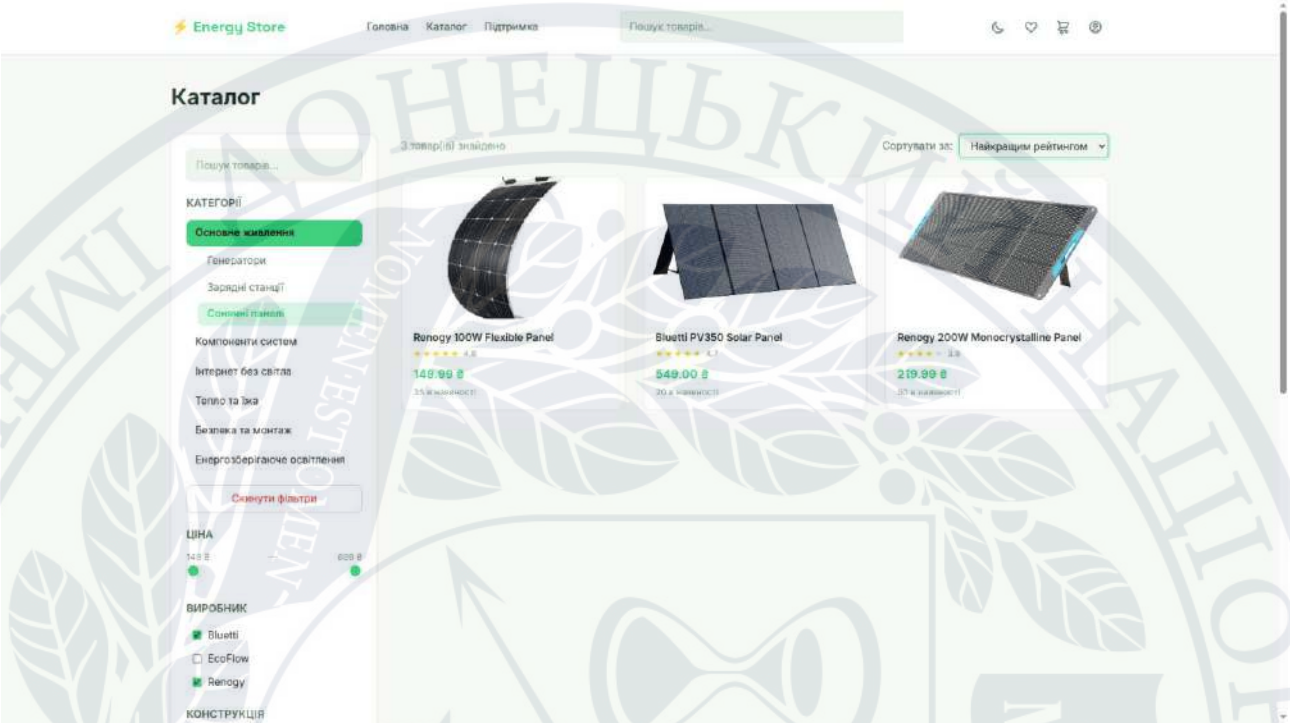


Рисунок 3.2 – Каталог товарів із застосованою панеллю фільтрів та сортуванням

Сторінка товару

На сторінці деталей товару доступний залишок розраховується з урахуванням кількості одиниць, вже доданих до кошика. Наприклад, якщо на складі є 3 одиниці, а 2 вже у кошику – відображається доступна кількість 1. У разі нульового залишку картка товару переходить у неактивний стан, кнопка додавання до кошика зникає. Після успішного додавання товару протягом 2 секунд відображається текстове підтвердження дії (рис. 3.3).

Рисунок 3.3 – Картка товару, доданого в кошик

Кошик та оформлення замовлення

Форма оформлення замовлення підтримує авторизованих і гостьових користувачів (рис. 3.4). Для авторизованих дані підтягуються з профілю, для гостей відображаються додаткові поля введення. Під час надсилання кнопка підтвердження блокується, щоб уникнути повторних запитів. Після успішного оформлення кошик очищається та відображається екран із номером замовлення.

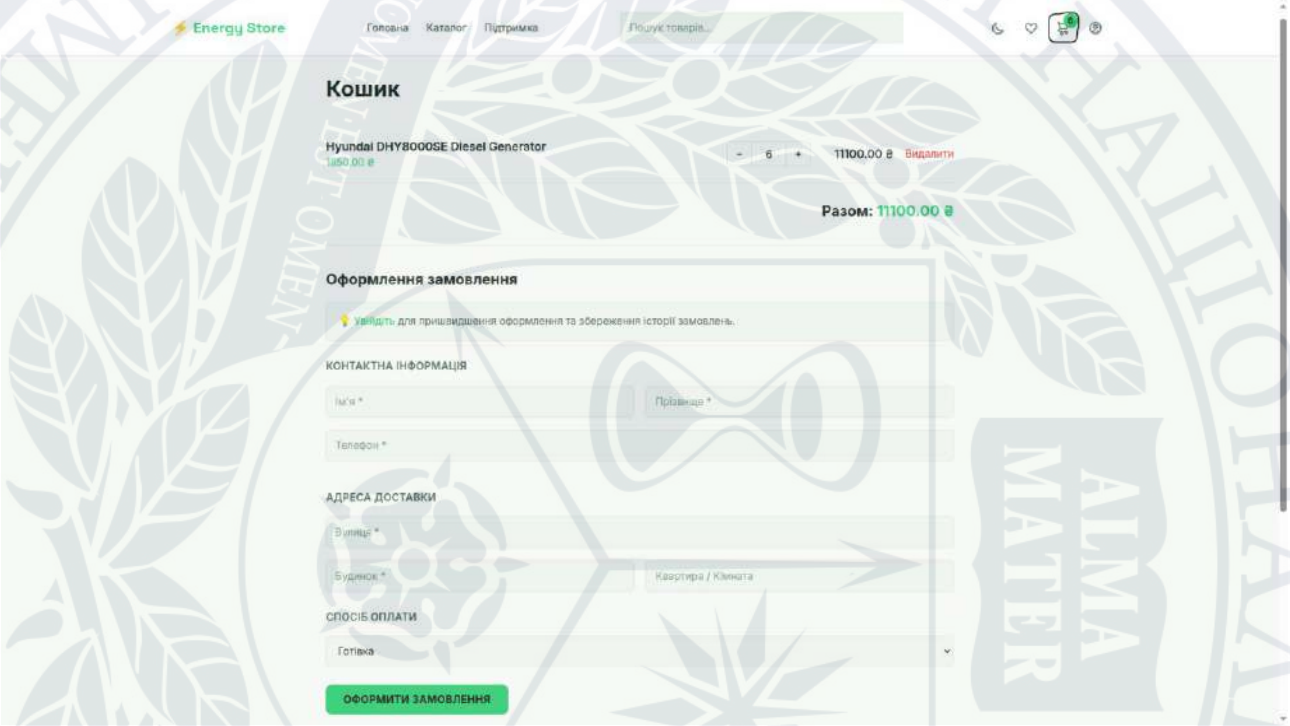


Рисунок 3.4 – Кошик з формою оформлення замовлення

Адміністративний модуль

Панель управління товарами та замовленнями забезпечує повний цикл адміністрування: перегляд, створення, редагування та видалення. Редагування товарів виконується у модальному вікні з динамічним редактором атрибутів, що дозволяє додавати пари «ключ – значення», які зберігаються у форматі JSONB. У панелі замовлень статус змінюється безпосередньо в таблиці через випадаючий список, після чого оновлення відображається одразу без повторного запиту до сервера. Статуси позначаються кольоровими мітками для швидкої візуальної ідентифікації (рис. 3.5).

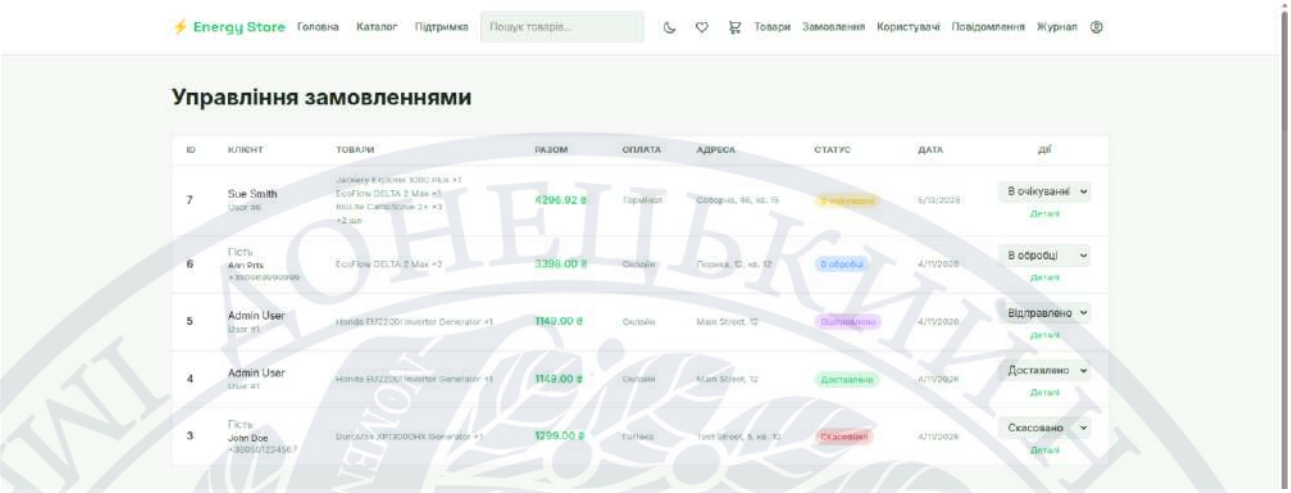


Рисунок 3.5 – Адміністративний модуль управління замовленнями

Адміністративний модуль додатково містить інструменти для моніторингу та комунікації: журнали аудиту та повідомлень. Журнал аудиту представлений у формі структурованої таблиці, що забезпечує повний контроль за діями персоналу шляхом фіксації дати, ідентифікатора адміністратора, типу операції та деталізованих параметрів змін об’єктів. Розділ повідомлень підтримки реалізований як реєстр вхідних запитів, де у зручному табличному форматі відображаються дані відправника, часові мітки та зміст звернень, що дозволяє адміністратору оперативно опрацьовувати клієнтські запити.

3.5. Тестування та розгортання системи

Тестування REST API. Функціональне тестування серверної частини виконано у середовищі Postman шляхом перевірки позитивних і негативних сценаріїв для кожного ресурсу API. Перевірялися коректність HTTP-статусів відповідей, структура тіла відповіді у форматі JSON, поведінка захисного проміжного програмного забезпечення – зокрема повернення статусів 401 та 403 у разі несанкціонованого доступу – а також дотримання бізнес-логічних обмежень, наприклад відхилення запиту з кодом 409 або 400 для спроб замовити кількість товару, що перевищує наявний залишок.

Тестування клієнтської частини. Ручне тестування інтерфейсу користувача виконано у браузері Chrome за такими ключовими сценаріями:

повний цикл покупця від реєстрації та входу до перегляду каталогу, додавання товарів до кошика, оформлення замовлення та перегляду його в особистому кабінеті; гостьове оформлення замовлення без авторизації з відображенням екрану підтвердження; додавання товарів до списку бажань із перевіркою коректної синхронізації після входу в обліковий запис; адміністративний цикл, що охоплює створення товару, зміну статусу замовлення та перевірку відповідного запису в журналі аудиту. Усі перелічені сценарії завершилися без помилок. Адаптивність інтерфейсу перевірено у режимі емуляції браузерних інструментів розробника для ширин 375 пікселів (мобільний пристрій), 768 пікселів (планшет) та 1280 пікселів (настільний комп'ютер).

Розгортання системи. Розгортання реалізовано з використанням Docker Compose для бази даних та скриптів для серверної й клієнтської частин. PostgreSQL 16 запускається у контейнері з іменованим томом для збереження даних та автоматичним перезапуском. Конфігураційні параметри підключення мають зчитуватися зі змінних середовища.

Локальне розгортання включає запуск контейнера БД, налаштування змінних середовища, виконання міграцій і заповнення бази, після чого запускаються сервер і клієнт. API доступний на порту 3000, клієнт – на 5173. Vite-проксі перенаправляє запити до API, усуваючи необхідність окремого налаштування CORS.

Тестування підтвердило коректність бізнес-логіки та відповідність вимогам. Використання Docker забезпечує відтворюваність середовища та спрощує розгортання на будь-якій машині з встановленим Docker.

У третьому розділі реалізовано повноцінну інформаційну систему. Сервер побудовано на Node.js із Fastify та PostgreSQL, із реалізацією автентифікації, авторизації та основних бізнес-процесів системи. Клієнтська частина розроблена як SPA на Vue 3 з централізованим керуванням станом і взаємодією з API. Завершальним етапом стали тестування та розгортання системи, які підтвердили її коректну роботу, стабільність і готовність до використання.

ВИСНОВКИ

У результаті виконання бакалаврської роботи розроблено та реалізовано інформаційну систему інтернет-магазину товарів енергонезалежності, що відповідає сучасним вимогам до вебзастосунків, забезпечує високу продуктивність, масштабованість та зручність використання.

У процесі виконання роботи систематизовано особливості предметної області ринку товарів енергонезалежності, який характеризується високою динамічністю розвитку, зростаючим попитом, технічною складністю продукції та значною залежністю від стану енергетичної інфраструктури. Виявлено основні обмеження існуючих онлайн-рішень, зокрема недостатню гнучкість пошуку та фільтрації товарів, перевантаженість інтерфейсів, обмежену масштабованість окремих платформ та вузьку спеціалізацію частини сервісів. Це дало змогу обґрунтувати актуальність створення більш гнучкої та функціональної системи.

У ході дослідження визначено теоретичні засади побудови сучасних вебінформаційних систем, зокрема розглянуто принципи клієнт-серверної архітектури, REST-підхід, SPA-модель, основи проєктування реляційних баз даних, а також ключові вимоги до безпеки, продуктивності та адаптивності інтерфейсу. Узагальнення цих положень дозволило сформувати цілісне концептуальне підґрунтя для подальшого проєктування системи.

На основі визначених теоретичних положень було сформульовано та формалізовано функціональні й нефункціональні вимоги до системи, а також визначено основні ролі користувачів і сценарії їхньої взаємодії з платформою. Це дало змогу структуровано окреслити майбутню архітектуру та виділити ключові функціональні модулі, серед яких управління товарами, кошиком, замовленнями та адміністративна частина.

Було обґрунтовано вибір технологічного стеку, до якого входять Node.js, Fastify, PostgreSQL, Vue 3 та інші. Обрані технології забезпечують ефективну взаємодію між компонентами системи, високу продуктивність і створюють передумови для подальшого масштабування та розвитку проєкту.

У межах проєктування розроблено трирівневу клієнт-серверну архітектуру, що забезпечує чіткий поділ відповідальності між клієнтською, серверною частинами та рівнем зберігання даних, підвищуючи модульність і спрощуючи супровід системи. Також спроєктовано реляційну базу даних PostgreSQL з продуманими зв'язками між сутностями та використанням ENUM-типів, що забезпечує цілісність і узгодженість даних.

Реалізовано серверну частину з REST API, JWT-автентифікацією, рольовою моделлю доступу та бізнес-логікою для обробки замовлень, фільтрації товарів і адміністративних функцій, що гарантує безпечну та стабільну взаємодію з даними. Паралельно створено клієнтську SPA-частину з адаптивним інтерфейсом, яка забезпечує швидку взаємодію із сервером і покращує користувацький досвід.

Завершальним етапом стало комплексне тестування та розгортання системи з використанням Docker, що підтвердило коректність роботи основних модулів, стабільність системи та її готовність до використання в реальних умовах.

Наукова новизна роботи полягає у формуванні цілісної архітектурно-функціональної моделі системи інтернет-магазину товарів енергонезалежності, в якій інтегровано сучасні принципи побудови клієнт-серверних застосунків, механізми безпечної автентифікації та структурованого управління даними з урахуванням специфіки та складності предметної області. Отримані результати підтверджують повне виконання поставлених у вступі завдань і демонструють, що розроблена інформаційна система відповідає сучасним вимогам та може бути основою для подальшого розвитку і масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Міністерство енергетики України. Стан енергетичної інфраструктури. NV Бізнес, 2025. URL: https://biz.nv.ua/ukr/economics/v-ukrajini-za-tri-roki-viyni-rosiyani-poshkodili-63-000-ob-yektiv-energetiki-50504863.html (дата звернення: 30.05.2026).

2. DOU. Стан електроенергетики України. 2024. URL: https://dou.ua/lenta/articles/electricity-industry-of-ukraine (дата звернення: 30.05.2026).

3. Pro-Consulting. Дослідження ринку товарів енергонезалежності в Україні 2023. URL: https://pro-consulting.ua (дата звернення: 30.04.2026).

4. TheInWeb Media. Яка різниця між B2B та B2C? URL: https://theinweb.media/yaka-riznicya-mizh-b2b-ta-b2c (дата звернення: 01.05.2026).

5. Kotler P., Keller K. L. Marketing Management. 16th ed. Pearson Education, 2021. 832 p.

6. Shoponlina. Online shops in Ukraine. URL: https://shoponlina.com/en/online-shops-in-ukraine (дата звернення: 03.05.2026).

7. Rozetka. Офіційний сайт інтернет-магазину. URL: https://rozetka.com.ua (дата звернення: 03.05.2026).

8. Епіцентр К. Офіційний сайт мережі магазинів. URL: https://epicentrk.ua (дата звернення: 03.05.2026).

9. YASNO. Офіційний сайт енергетичної компанії. URL: https://yasno.ua (дата звернення: 03.05.2026).

10. EcoFlow Ukraine. Офіційний сайт представництва в Україні. URL: https://ecoflowukraine.com (дата звернення: 03.05.2026).

11. Laudon K. C., Laudon J. P. Management Information Systems: Managing the Digital Firm. 16th ed. Pearson, 2019. 656 p.

12. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. 3rd ed. Prentice Hall, 2017. 596 p.

13. Osmani A. Learning JavaScript Design Patterns. O'Reilly Media, 2023. 298 p.
14. RESTful API. What is REST? URL: https://restfulapi.net (дата звернення: 07.05.2026).
15. DigitalOcean. Database Normalization. URL: https://www.digitalocean.com/community/tutorials/database-normalization (дата звернення: 07.05.2026).
16. CData. Object-Relational Mapping. URL: https://www.cdata.com/blog/object-relational-mapping (дата звернення: 07.05.2026).
17. OWASP. Top 10 Web Application Security Risks 2025. URL: https://owasp.org/Top10/2025 (дата звернення: 07.05.2026).
18. MDN Web Docs. Responsive Design. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design (дата звернення: 07.05.2026).
19. Ecommerce Germany. European Ecommerce Overview: Ukraine. URL: https://ecommercegermany.com/blog/european-ecommerce-overview-ukraine (дата звернення: 08.05.2026).
20. Vue.js. Official Documentation. Introduction. URL: https://vuejs.org/guide/introduction.html (дата звернення: 09.05.2026).
21. Angular. Official Documentation. Overview. URL: https://angular.dev/overview (дата звернення: 09.05.2026).
22. React. Official Documentation. Learn React. URL: https://react.dev/learn (дата звернення: 09.05.2026).
23. Vite. Official Documentation. Getting Started. URL: https://vite.dev/guide (дата звернення: 09.05.2026).
24. Pinia. Official Documentation. Core Concepts. URL: https://pinia.vuejs.org/core-concepts (дата звернення: 09.05.2026).
25. Tailwind CSS. Official Documentation. Installation with Vite. URL: https://tailwindcss.com/docs/installation/using-vite (дата звернення: 09.05.2026).
26. Node.js. Official Documentation. Synopsis. URL: https://nodejs.org/docs/latest/api/synopsis.html (дата звернення: 09.05.2026).

27. Fastify. Official Documentation. URL: https://fastify.dev (дата звернення: 09.05.2026).

28. Express.js. Official Website. URL: https://expressjs.com (дата звернення: 09.05.2026).

29. Hapi.js. Official Website. URL: https://hapi.dev (дата звернення: 09.05.2026).

30. NestJS. Official Website. URL: https://nestjs.com (дата звернення: 09.05.2026).

31. PostgreSQL. Official Website. URL: https://www.postgresql.org (дата звернення: 10.05.2026).

32. MongoDB. Official Website. URL: https://www.mongodb.com (дата звернення: 10.05.2026).

33. MySQL. Official Website. URL: https://www.mysql.com (дата звернення: 10.05.2026).

34. Drizzle ORM. Official Documentation. Overview. URL: https://orm.drizzle.team/docs/overview (дата звернення: 10.05.2026).

35. Docker. Official Documentation. URL: https://docs.docker.com (дата звернення: 11.05.2026).

36. Docker Compose. Official Documentation. URL: https://docs.docker.com/compose (дата звернення: 11.05.2026).

37. bcrypt. npm Package. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 11.05.2026).

38. Neon. PostgreSQL ENUM Tutorial. URL: https://neon.com/postgresql/tutorial/enum (дата звернення: 10.05.2026).

39. pgModeler. Relationships Documentation. URL: https://pgmodeler.io/support/docs/310-relationships (дата звернення: 10.05.2026).

40. JWT.io. Introduction to JSON Web Tokens. URL: https://www.jwt.io/introduction#what-is-json-web-token (дата звернення: 07.05.2026).

41. IBM. Role-Based Access Control (RBAC). URL: https://www.ibm.com/think/topics/rbac (дата звернення: 07.05.2026).

42. TheTextTool. Camel Case vs Snake Case. URL: https://thetexttool.com/compare/camel-vs-snake-case (дата звернення: 10.05.2026).
43. Postman. Official Website. URL: https://www.postman.com (дата звернення: 12.05.2026).

ДОДАТКИ

ДОДАТОК А

Таблиця А.1 Матриця прецедентів використання системи за ролями

Прецедент	**Гість**	**Клієнт**	**Адміністратор**
Перегляд каталогу товарів	доступно	доступно	доступно
Пошук та фільтрація	доступно	доступно	доступно
Перегляд картки товару	доступно	доступно	доступно
Додавання до списку бажань	–	доступно	доступно
Додавання до кошика	доступно	доступно	доступно
Оформлення замовлення	доступно	доступно	доступно
Перегляд історії замовлень	–	доступно	доступно
Редагування профілю	–	доступно	доступно
Керування товарами	–	–	доступно
Зміна статусів замовлень	–	–	доступно
Керування користувачами	–	–	доступно

ДОДАТОК Б

Порівняльний аналіз використаних інструментів розробки

Таблиця Б.1 Порівняльний аналіз SPA-фреймворків

Критерій	**Vue 3**	**React**	**Angular**
Крива навчання	низька	середня	висока
Розмір бандлу (мін.)	~22 KB	~42 KB	~143 KB
Composition API / Hooks	вбудований	аналог (Hooks)	обмежено
SSR підтримка	Nuxt 3	Next.js	Angular SSR
TypeScript інтеграція	повна	повна	повна
Двостороннє зв’язування даних	+	ручне	+
Офіційний state manager	Pinia	Redux/Zustand	NgRx
Швидкість рендерингу	висока	висока	середня

Таблиця Б.2 Порівняльний аналіз Node.js-фреймворків

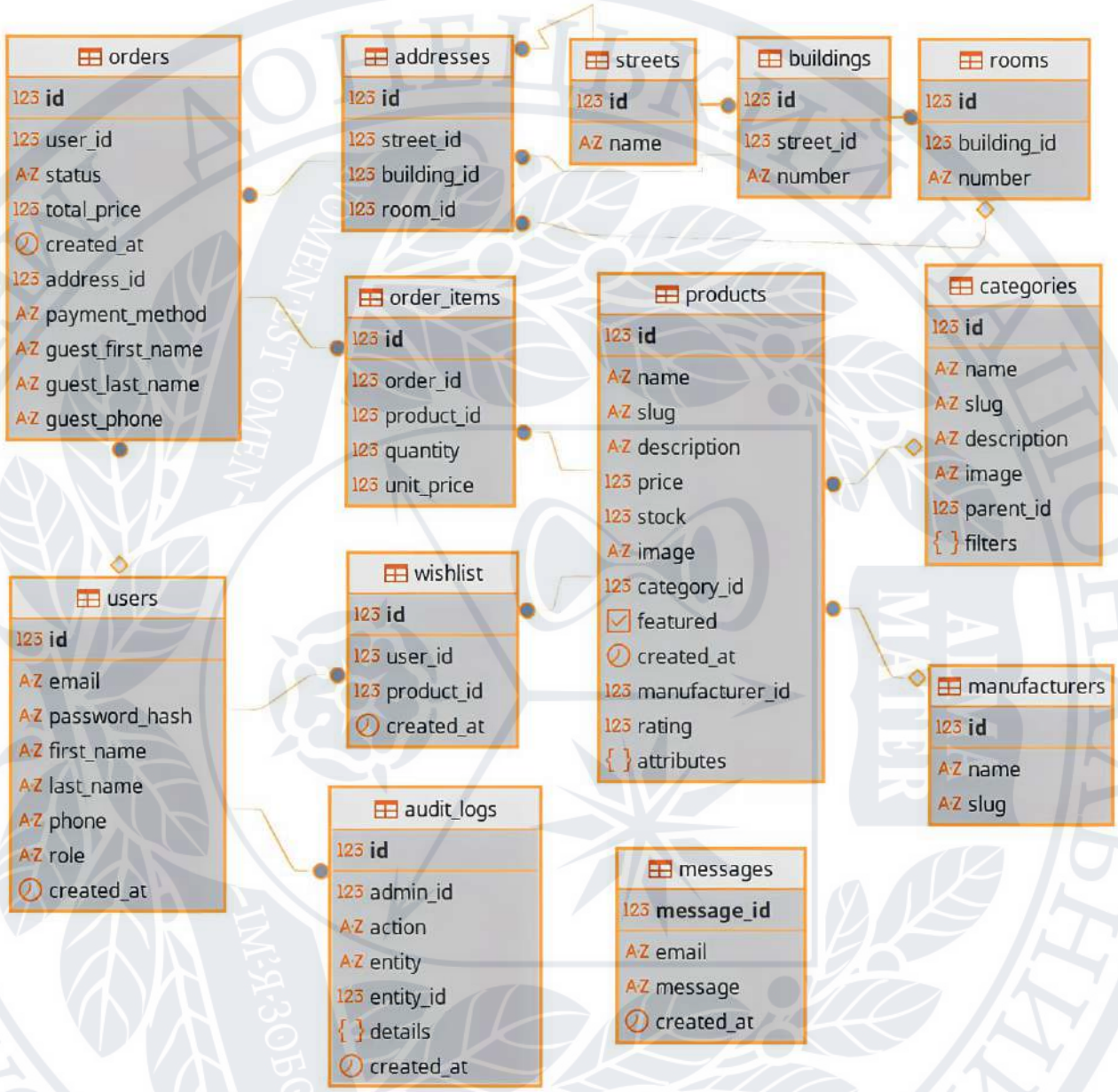
Критерій	**Fastify**	**Express**	**Hapi**	**NestJS**
Запитів/сек (bench.)	~76 000	~15 000	~38 000	~25 000
JSON Schema валідація	вбудована	стороння	вбудована	через pipes
TypeScript підтримка	повна	часткова	повна	повна
Система розширень	async DI	middleware	register	модулі
Накладні витрати	мінімальні	середні	середні	значні
OpenAPI / Swagger	автогенерування	сторонній	сторонній	вбудований
Крива навчання	низька	найнижча	середня	висока

Таблиця Б.3 Порівняльний аналіз СУБД

Критерій	**PostgreSQL**	**MongoDB**	**MySQL**
Модель даних	об’єктно-реляційна	документна	реляційна
ACID-транзакції	+ (найнадійніші)	+	+
Підтримка JSON	повна	нативна	обмежена
Зовнішні ключі	+	-	+
Складні JOIN-запити	найкраща підтримка	обмежено	базово
Full-text пошук	+	+	обмежено

ДОДАТОК В

Рисунок В.1 – Схема бази даних інформаційної системи



ДОДАТОК Г

Таблиця Г.1 Маршрути клієнтської частини та рівні їхнього захисту

Шлях	Компонент (View)	Guard	Призначення
/	Home.vue	–	Головна сторінка
/products	Products.vue	–	Каталог із фільтрами
/products/:id	ProductDetail.vue	–	Картка товару
/cart	Cart.vue	–	Кошик та оформлення
/login	Login.vue	guest	Форма входу
/register	Register.vue	guest	Форма реєстрації
/profile	Profile.vue	requiresAuth	Особистий кабінет
/orders	Orders.vue	requiresAuth	Список замовлень
/orders/:id	OrderDetail.vue	requiresAuth	Деталі замовлення
/wishlist	Wishlist.vue	requiresAuth	Список бажань
/support	Support.vue	–	Центр підтримки
/admin/products	AdminProducts.vue	requiresAdmin	Управління товарами
/admin/orders	AdminOrders.vue	requiresAdmin	Управління замовленнями
/admin/orders/:id	AdminOrderDetail.vue	requiresAdmin	Деталі замовлення (адмін)
/admin/users	AdminUsers.vue	requiresAdmin	Управління користувачами
/admin/messages	AdminMessages.vue	requiresAdmin	Повідомлення підтримки
/admin/audit	AdminAuditLog.vue	requiresAdmin	Журнал аудиту