ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему:

МОДЕЛЮВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ З ФУНКЦІЯМИ СКЛАДСЬКОГО ОБЛІКУ

Виконала: здобувачка вищої освіти
групи КН 2022-1
спеціальності
122 «Комп'ютерні науки»
Станіслава СЄННІКОВА

Керівник:
к.т.н., доц. Марина БУЛАЄНКО

Рецензент:
д.ф-м.н., проф. Наталія СІЗОВА

м. Харків – 2026 рік

Харківський національний університет міського господарства імені О. М. Бекетова

(повне найменування закладу вищої освіти)

Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«_23_» __червня__2026 Року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сєнніковій Станіславі Артурівні

(прізвище, ім'я, по батькові)

1. Тема роботи «Моделювання та реалізація інформаційної системи електронної комерції з функціями складського обліку»

керівник роботи к.т.н., доц. БулаєнкоМ.В.

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

затверджені наказом закладу вищої освіти від «22» травня 2026 р. № 440-03

2. Термін подання студентом роботи 15.06.2026р.

3. Вихідні дані до роботи: Керівництво з розробки та впровадження платформи електронної комерції, що включає в себе функціонал управління запасами, з використанням технологічного стеку: Python/Django, React та PostgreSQL.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Оцінка цільового ринку та вивчення конкурентного середовища рішень для електронної комерції; створення схеми реляційної бази даних та розробка об'єктно-орієнтованої структури системи; побудова RESTful API з використанням JWT для безпечної автентифікації; впровадження функціоналу управління запасами зі слідкуванням за актуальним станом товарів; проведення тестування розробленого програмного продукту; дослідження умов праці та впровадження заходів для гарантування безпеки на робочому місці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація – 18 слайдів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Марина БУЛАЄНКО	15.05.2026	20.05.2026
Розділ II	Марина БУЛАЄНКО	21.05.2026	25.05.2026
Розділ III	Марина БУЛАЄНКО	26.05.2026	06.06.2026
Розділ IV	Вікторія МАЛИШЕВА	07.06.2026	15.06.2026

7. Дата видачі завдання 15.05.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми кваліфікаційної роботи	15.05.2026	Виконано
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи	16.05.2026	Виконано
3	Написання I розділу	20.05.2026	Виконано
4	Написання II розділу	25.05.2026	Виконано
5	Написання III розділу	06.06.2026	Виконано
6	Написання IV розділу	11.06.2026	Виконано
7	Подання кваліфікаційної роботи керівнику	13.06.2026	Виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до кваліфікаційної роботи	16.06.2026	Виконано
9	Подання доопрацьованого варіанту кваліфікаційної роботи керівнику	18.06.2026	Виконано
10	Захист матеріалів кваліфікаційної роботи на засіданні кафедри	21.06.2026	Виконано
11	Офіційний захист матеріалів кваліфікаційної роботи на засіданні екзаменаційної комісії	24.06.2026	Виконано

Студент Станіслава СЄННІКОВА

Керівник роботи Марина БУЛАЄНКО

АНОТАЦІЯ

Сєннікова С.А. Моделювання та реалізація інформаційної системи електронної комерції з функціями складського обліку. Кваліфікаційна робота бакалавра здобувача вищої освіти першого (бакалаврського) рівня групи КН 2022-1 спеціальності 122 «Комп'ютерні науки».

Пояснювальна записка містить 4 розділи, 16 рисунків, 12 таблиць, список, 5 формул, 36 літературних джерел. Загальний обсяг – 68 сторінок.

Мета роботи – спроектувати та запрограмувати веб-систему, що об'єднує в одному програмному продукті функціонал інтернет-магазину і повноцінний облік товарів на складі.

Перший розділ присвячено дослідженню ринку онлайн-торгівлі [36], виявленню типових проблем підприємств, що не мають зв'язку між торговою частиною і складом. Розглянуто чотири платформи-аналоги; за результатами порівняння сформовано технічне завдання на розробку.

У другому розділі побудовано структурну модель бази даних (10 таблиць, ER-діаграма), розроблено діаграму класів Django-застосунку та описано математичні формули розрахунку суми замовлення й коефіцієнта оборотності запасів, а також алгоритми транзакційної обробки замовлень.

Третій розділ охоплює обґрунтування технологій (Python, Django, React, PostgreSQL), архітектуру REST API з JWT-захистом, опис чотирьох програмних модулів, результати тестування і скріншоти ключових екранів.

Четвертий розділ присвячено охороні праці: охарактеризовано нормативно-правову базу, виявлено 9 потенційних небезпек на робочому місці оператора системи, проведено оцінювання ризиків методом матриці та аналізу дерева відмов, розроблено заходи щодо їх попередження.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА КОМЕРЦІЯ, СКЛАДСЬКИЙ ОБЛІК, БАЗА ДАНИХ, REST API, DJANGO, POSTGRESQL, REACT, ВЕБ-ЗАСТОСУНОК, UML-ДІАГРАМА.

ABSTRACT

Siennikova S.A. Modeling and implementation of an e-commerce information system with warehouse accounting functions. Bachelor's qualification work.

The explanatory note contains 4 sections, 16 figures, 12 tables, 5 formulas, 36 references. Total volume is 68 pages.

The goal is to design and build a web application that combines online store functionality with integrated warehouse stock management in a single coherent product.

Section 1 investigates the online retail market, identifies operational problems caused by the disconnect between trade and warehouse subsystems, and compares four competing platforms to justify a custom solution.

Section 2 constructs a 10-table relational database model with an ER-diagram, a Django class diagram, mathematical formulas for order totals and inventory turnover, and transactional order-processing algorithms.

Section 3 justifies the technology choices (Python/Django, React, PostgreSQL), describes the REST API with JWT authentication, four application modules, 78% test coverage, and key interface screenshots.

Section 4 addresses occupational safety: the regulatory framework is outlined, 9 potential hazards at the operator’s workstation are identified, risks are assessed using the risk matrix method and fault-tree analysis, and mitigation measures are proposed.

KEYWORDS: INFORMATION SYSTEM, E-COMMERCE, WAREHOUSE ACCOUNTING, DATABASE, REST API, DJANGO, POSTGRESQL, REACT, WEB APPLICATION, UML DIAGRAM.

ЗМІСТ

ВСТУП

Вже давно минули ті часи, коли ведення бізнесу онлайн було чимось новим – наразі для багатьох підприємців це єдиний шлях реалізації своєї продукції, а для декого – й першим кроком у підприємницькій діяльності. Проте, як показує практика, переважна більшість керівників невеликих інтернет-крамниць вдаються до ведення обліку товарів або за допомогою таблиць Excel, або взагалі по пам'яті, абсолютно не пов'язуючи це з системою продажів. Наслідки такого підходу цілком зрозумілі будь-якому менеджеру: замовлення підтверджується, а фактичного товару на складі вже не залишається; оновлення залишків відбувається із запізненням; відсутня будь-яка інформаційна база для планування закупівель. Проблема полягає не в небажанні вести облік, а радше у відсутності зручного й доступного інструменту, який би об'єднував онлайн-торгівлю та складський облік в єдиному інтерфейсі, доступному для малого бізнесу безкоштовно.

Саме цю невирішену проблему покликана вирішити ця робота. Замість комбінування кількох окремих продуктів, пропонується розробити та втілити в життя єдиний веб-додаток, де каталог товарів, кошик, процес оформлення замовлення та склад будуть інтегровані в єдину структуру з автоматизованим обміном інформацією. При списанні товару з замовлення, залишок зменшується миттєво; при надходженні партії від постачальника – він збільшується та фіксується в журналі. Менеджер бачить актуальну картину без необхідності виконувати будь-які додаткові дії.

Мета цієї роботи полягає у проектуванні архітектури та розробці програмного продукту у вигляді веб-додатку для електронної комерції, що включає інтегрований модуль складського обліку. Це забезпечить повну автоматизацію процесів, починаючи від прийому замовлення від покупця і закінчуючи відображенням змін залишків на складі.

Для досягнення вищезгаданої мети необхідно виконати наступні завдання:

– Дослідити специфіку функціонування інтернет-магазину як предметної області; з'ясувати причини, через які існуючі рішення не забезпечують повне вирішення поставленого завдання.

– Розробити схему реляційної бази даних та створити об'єктно-орієнтовану модель, що підходить для подальшої програмної реалізації.

– Формалізувати алгоритми ключових операцій, таких як оформлення замовлення з атомарним резервуванням залишків та автоматичне виявлення дефіцитних товарів.

– Втілити систему, використовуючи обраний технологічний стек, та здійснити її тестування за допомогою автоматизованих тестів та ручного сценарного тестування.

– Оцінити умови праці під час розробки програмного забезпечення та надати рекомендації відповідно до чинних нормативів.

Об'єкт дослідження: веб-додаток, що слугує для автоматизації процесів, пов'язаних з інтернет-торгівлею та обліком товарів на складі.

Предмет дослідження: методології моделювання реляційних баз даних, принципи побудови REST API, підходи до об'єктно-орієнтованого проектування та технології розробки клієнт-серверних систем.

Методи дослідження: використання UML-нотації для опису процесів та моделей класів, ER-нотації для проектування бази даних, архітектурного стилю REST для створення API, а також методу транзакційного програмування для гарантування цілісності даних під час одночасних операцій.

Практична цінність розробленого продукту: Він не прив'язаний до жодних SaaS-платформ, може бути встановлений на власному сервері, не вимагає жодних ліцензійних відрахувань та може бути адаптований під специфічний асортимент або схему доставки без необхідності залучення сторонніх фахівців.

РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

У цьому дослідженні під терміном «електронна комерція» мається на увазі не просто наявність веб-сторінки з функцією «Придбати», а весь комплекс автоматизованих торговельних процесів. Це охоплює етапи від появи одиниці товару в каталозі до моменту підтвердження її доставки клієнту, а також фіксацію фінансових операцій. Така інтерпретація є ширшою за загальноприйняту і включає управління асортиментом, обробку платежів, взаємодію з логістичними службами та, що є ключовим для даної роботи, точний облік залишків товарів [1].

В Україні онлайн-торгівля пережила кілька значимих періодів бурхливого зростання. Перший етап припав приблизно на 2014–2016 роки, коли з'явилися великі інтернет-платформи, а споживачі почали активно використовувати онлайн-розрахунки картками. Другий сплеск спостерігався під час пандемії у 2020 році, коли офлайн-магазини тимчасово припинили роботу, що змусило багатьох продавців перейти в онлайн-формат. Третій значний ріст стався після початку повномасштабного вторгнення, коли для частини України інтернет-торгівля стала єдиним доступним каналом збуту [2, 3]. Кожний з цих періодів залучив нових гравців ринку, які не завжди були готові до повноцінної автоматизації, але були вимушені знаходити способи управління замовленнями та залишками.

Правову основу діяльності інтернет-магазинів в Україні становить Закон «Про електронну комерцію» № 675-VIII, прийнятий у 2015 році [4]. Цей закон зобов'язує продавця надавати покупцю правдиву інформацію про наявність товарів до укладення угоди. Таким чином, неточність у даних про залишки є не просто операційною незручністю, а потенційним порушенням законодавства. Отже, автоматична синхронізація даних каталогу зі складськими залишками є не тільки зручністю, а й законодавчою вимогою.

Найбільш поширений сценарій роботи невеликого інтернет-магазину передбачає використання готової системи управління контентом (CMS) або SaaS-платформи для торговельної частини. Водночас, облік товарів на складі часто ведеться у таблицях Excel або Google Sheets, які менеджер оновлює вручну після кожної операції відправки. Відсутність прямого автоматичного зв'язку між цими двома системами призводить до проблем: якщо менеджер забуде відкоригувати залишки або помилиться з кількістю, клієнт може отримати підтвердження замовлення на товар, який фактично закінчився на складі. Саме цю проблему роз'єднаності систем має на меті вирішити система, що розробляється в рамках цього проекту.

Окрім помилок, пов'язаних з відображенням наявності товарів, відсутність прямої взаємодії між системами унеможливлює ефективне планування закупівель. Менеджеру складно визначити, які позиції продаються швидко і потребують поповнення, а які товари затримуються на складі. В результаті, замовлення постачальникам робляться інтуїтивно, що призводить або до надлишкових запасів, або до дефіциту товарів. Розроблювана система має вирішити ці питання шляхом впровадження журналу руху товарів та розрахунку показника оборотності запасів.

1.1.1 Опис процесу діяльності

У розроблюваній системі можна виділити три наскрізні процеси, що перетинаються у точках зміни стану замовлення і зміни залишків на складі.

Покупець взаємодіє лише з першим процесом – торговим. Він переглядає каталог, відбирає товари у кошик, вводить адресу доставки і підтверджує замовлення. З цього моменту ініціатива переходить до системи: вона перевіряє актуальність цін і наявність кожної позиції, формує запис замовлення і надсилає підтвердження. Покупець може лише спостерігати за зміною статусу у своєму особистому кабінеті.

Менеджер отримує сповіщення про нове замовлення і починає другий процес – операційний. Він перевіряє деталі, при необхідності зв'язується з

покупцем щодо уточнень, підтверджує замовлення і ставить його у чергу на відвантаження. Ключовий момент: у цю мить система автоматично списує замовлені кількості з таблиці залишків і фіксує цей рух у журналі. Менеджер не виконує жодних ручних дій зі складом – він лише змінює статус замовлення.

Третій процес – постачальницький – запускається окремо, незалежно від замовлень. Менеджер оформлює прихідну накладну у складському модулі: вказує постачальника, дату і перелік товарів із кількістю та закупівельною ціною. Після проведення накладної залишки збільшуються, а в журналі руху з'являються записи типу «прихід». Таким чином, у журналі завжди можна відстежити кожен рух товару – звідки прийшов і куди пішов. Рис. 1.1 ілюструє ці три процеси у вигляді UML-діаграми діяльності.

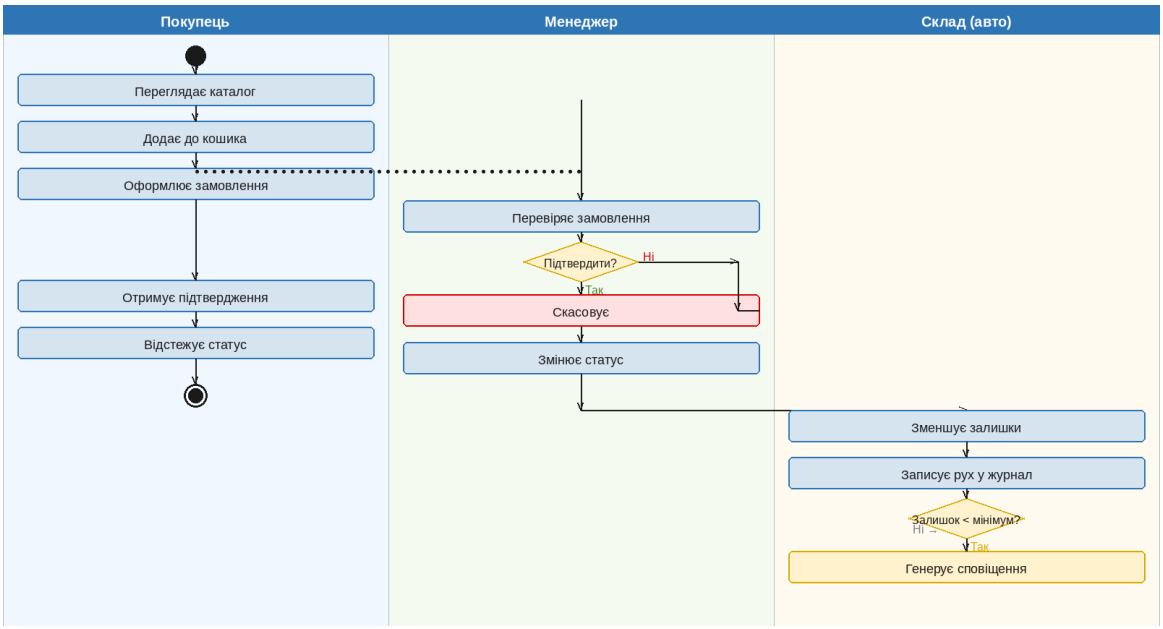

Рисунок 1.1 – UML-діаграма діяльності – три паралельні процеси системи

На діаграмі видно, що покупець і менеджер взаємодіють безпосередньо, тоді як складський модуль отримує дані автоматично через тригери бізнес-логіки – без участі будь-якого актора. Це і є ключова властивість системи, що відрізняє її від ручного управління в таблицях.

1.1.2 Опис функціональної моделі

Щоб систематизувати вимоги до системи з погляду кожного типу користувача, скористаємося UML-діаграмою варіантів використання. Вона фіксує, що кожен актор може і чого не може робити в системі, і слугує відправною точкою для проектування таблиць прав доступу.

Система передбачає три актори: Покупець, Менеджер і Адміністратор. Менеджер успадковує права Покупця (також може переглядати каталог і оформляти замовлення), але додатково отримує доступ до адміністративних функцій торгової і складської частин. Адміністратор, у свою чергу, включає всі права Менеджера і додатково керує обліковими записами та переглядає зведену аналітику.

Для Покупця доступні: реєстрація і вхід до облікового запису; пошук і перегляд товарів з фільтром за категорією і ціною; управління кошиком (додати, змінити кількість, видалити позицію); оформлення замовлення з введенням адреси; відстеження стану замовлень у особистому кабінеті; редагування власного профілю.

Менеджер отримує доступ до: панелі замовлень із фільтрацією за статусом і датою; зміни статусу замовлення на кожному етапі циклу обробки; CRUD-операцій з товарним каталогом (додавання, редагування, тимчасова деактивація товару без видалення); таблиці складських залишків з підсвічуванням критичних позицій; форми прихідних накладних від постачальників; журналу руху товарів з фільтрацією.

Адміністратор доповнює цей перелік управлінням користувачами (реєстрація менеджерів, блокування акаунтів, зміна ролей), а також переглядом дашборду: обсяг продажів за місяць, топ-товари, динаміка нових покупців. Графічне зображення наведено на рис. 1.2.

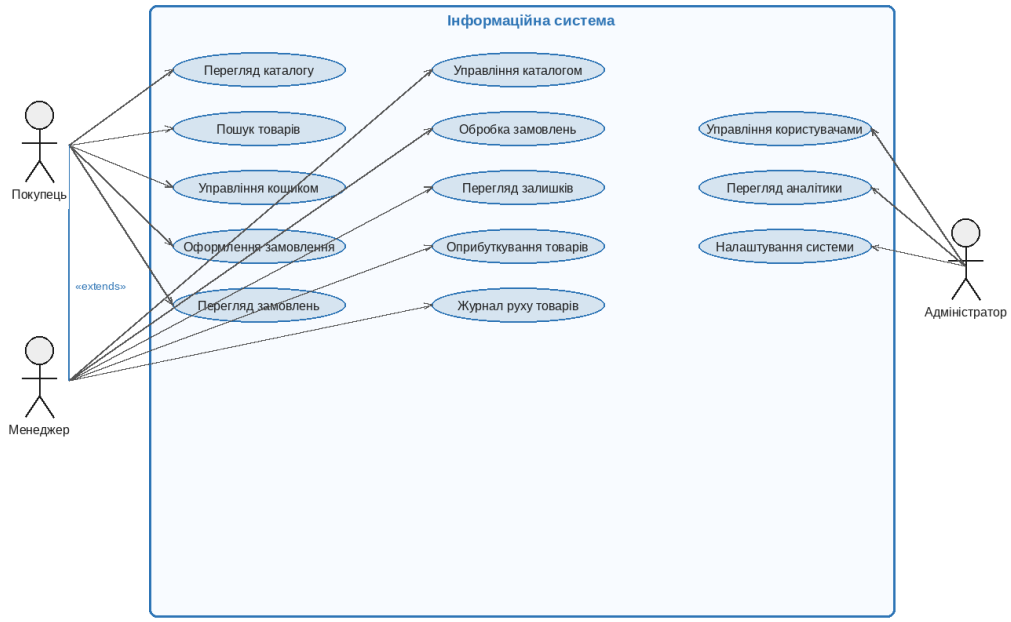

Рисунок 1.2 – UML-діаграма варіантів використання інформаційної системи

1.2 Огляд наявних аналогів

Перш ніж розпочинати власну розробку, варто чесно відповісти на запитання: чи немає на ринку готового рішення, яке задовольняє сформульовані вимоги? Для цього було проаналізовано чотири платформи, що найчастіше використовуються у вітчизняному сегменті малого та середнього онлайн-бізнесу.

WooCommerce – це розширення для WordPress, що перетворює звичайний сайт на інтернет-магазин [5]. Перевага – величезна кількість додаткових плагінів і активна спільнота. Проблема зі складом: стандартна версія дозволяє лише зазначити кількість одиниць і отримати email, коли вона впаде до нуля. Журналу руху немає – лише поточне число. Щоб отримати повноцінний складський облік, потрібен платний плагін ATUM або подібний, що коштує від 100 доларів на рік і потребує окремого налаштування. Таким чином, реальна вартість рішення зростає, а інтеграція залежить від якості стороннього плагіна.

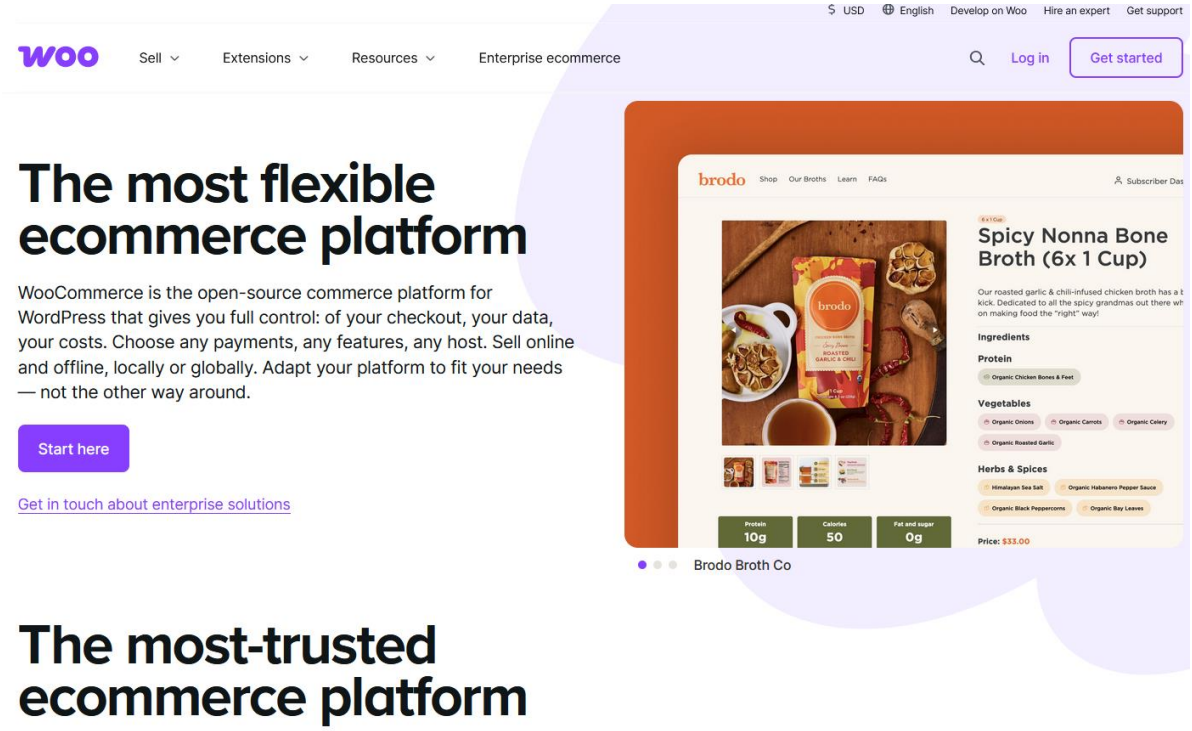

Рисунок 1.3 – Головна сторінка WooCommerce

OpenCart – самостійна CMS виключно для магазинів, яку можна встановити на власний хостинг безкоштовно [6]. Тут вже є вбудований модуль кількості товарів і базовий звіт про залишки. Однак ані постачальників, ані прихідних накладних, ані журналу руху в стандартній збірці немає. Звіти прив'язані до транзакцій замовлень, але показати, скільки товару прийшло від постачальника минулого тижня, система не може. Розширити функціонал можна через маркетплейс OpenCart, але знову платно.

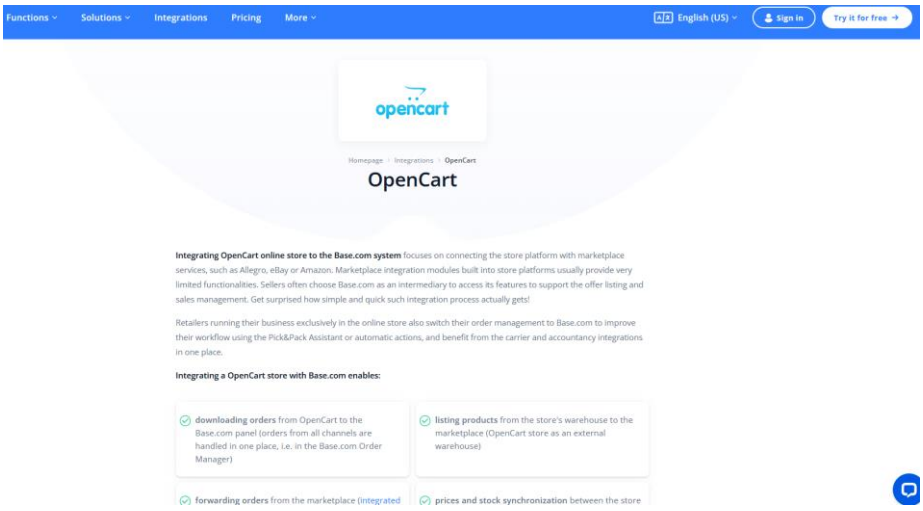

Рисунок 1.4 – Головна сторінка OpenCart

Magento (Adobe Commerce) – рішення іншого масштабу [7]. У ньому є multi-source inventory (підтримка кількох складів), передзамовлення, повний журнал, інтеграція з ERP. Але за хмарну версію Adobe Commerce стартовий тариф перевищує 20 000 доларів на рік. Open Source-версія Magento безкоштовна, але вимагає виділеного сервера, DevOps-підтримки і мінімум кількох місяців на впровадження. Для магазину з 500 SKU це стрільба з гарма

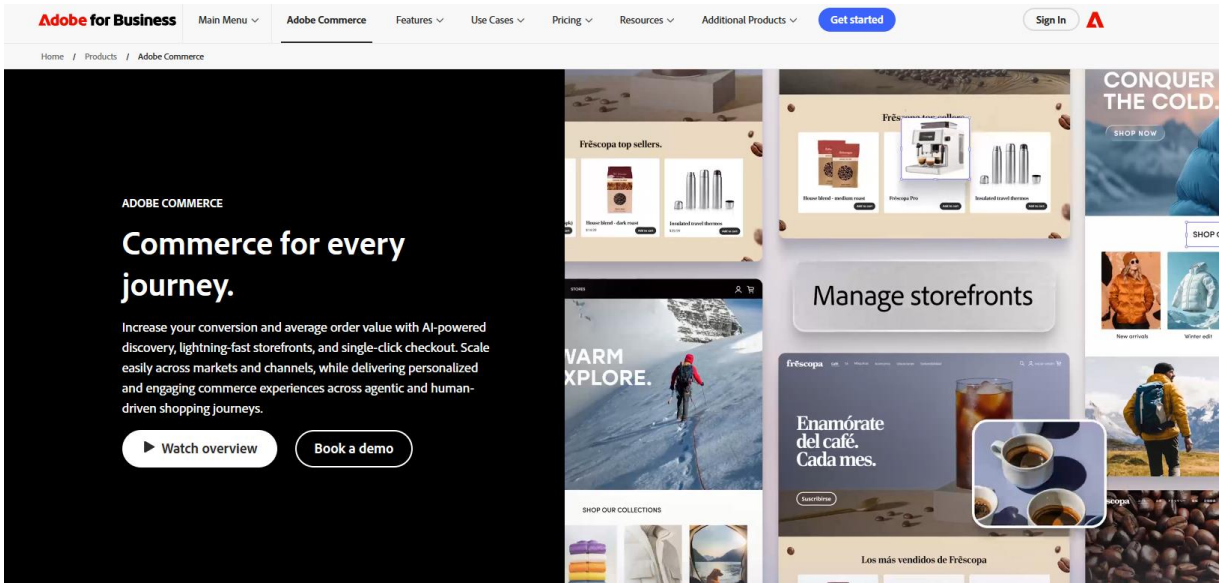

Рисунок 1.5 – Головна сторінка Magento (Adobe Commerce)

Хорошоп – SaaS-платформа українського виробництва, добре інтегрована з Новою Поштою, LiqPay, Monobank і рядом інших локальних сервісів. Це головна перевага: підключення займає хвилини, а не дні. Але за ту ж причину (SaaS) неможливо глибоко змінити бізнес-логіку: якщо потрібна нестандартна схема списання або специфічний звіт по постачальнику, доведеться чекати, поки компанія додасть таку функцію. Складський модуль є, але він базовий і не підтримує журнал руху по кожній накладній.

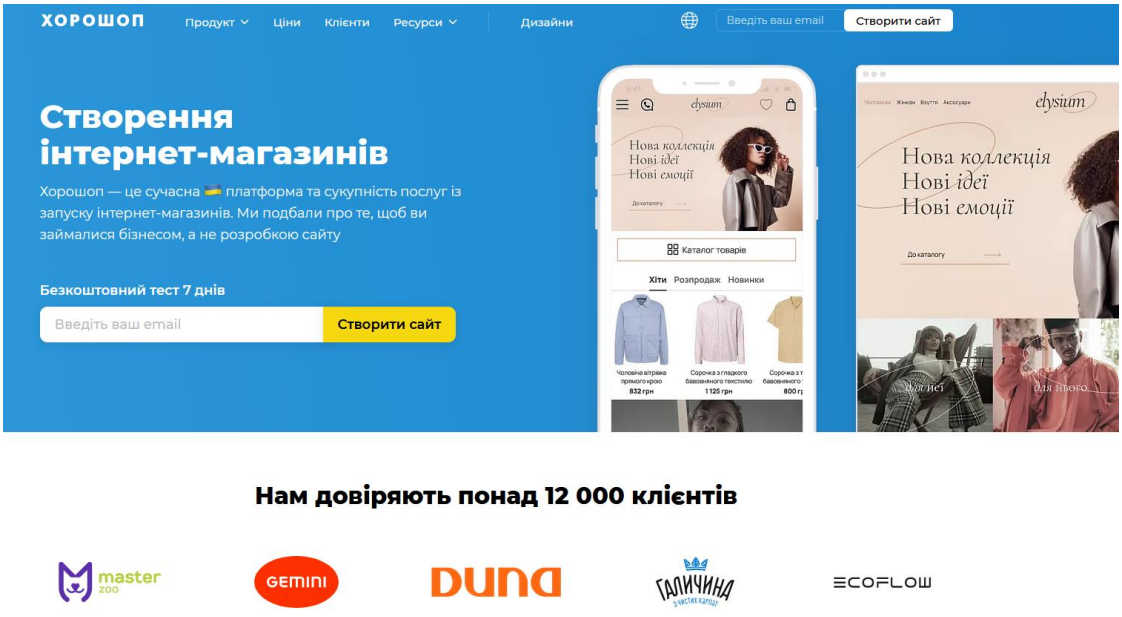

Рисунок 1.6 – Головна сторінка Хорошоп

Зведений порівняльний аналіз наведено у табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз програмних аналогів

Критерій	WooCommerce	OpenCart	Magento	Хорошоп
Ліцензія	Безкоштовна	Безкоштовна	Комерційна	SaaS (платна)
Складський облік	Базовий (плагін)	Вбудований базовий	Розширений	Обмежений
Звітність	Розширена (плагін)	Базова	Розширена	Базова
Складність налашт.	Середня	Низька	Висока	Низька
Українські інтеграції	Обмежені	Обмежені	Обмежені	Широкі
Кастомізація	Висока	Середня	Висока	Низька
API	REST API	REST API	REST/GraphQL	REST API

З даних таблиці видно: жодна з розглянутих безкоштовних платформ не надає повноцінного складського обліку в базовій конфігурації. Платформа корпоративного рівня (Magento) має потрібний функціонал, але є недоступною за вартістю. SaaS-рішення (Хорошоп) обмежує кастомізацію. Це означає, що

для магазину, якому потрібна вільна архітектура і повний контроль над складом без додаткових витрат на плагіни, доцільно розробити власний програмний продукт.

1.3 Постановка задачі

Узагальнивши результати аналізу предметної галузі та порівняння аналогів, можна сформулювати конкретну технічну задачу: створити веб-застосунок із відкритим кодом, де модуль інтернет-магазину і модуль складського обліку поділяють спільну базу даних і оновлюють залишки автоматично, без участі людини.

Функціональні вимоги – що система зобов'язана вміти:

– реєструвати користувачів з трьома рівнями доступу і видавати JWT-токен після успішного входу;

– надавати каталог товарів з ієрархічними категоріями, пагінацією і повнотекстовим пошуком;

– приймати товари у кошик і конвертувати кошик у замовлення однією атомарною операцією;

– при підтвердженні замовлення автоматично зменшувати залишки і записувати рух у журнал;

– проводити прихідні накладні від постачальників з автоматичним збільшенням залишків;

– сигналізувати менеджеру, коли залишок товару падає нижче встановленого мінімуму;

– відображати журнал руху товарів з фільтрацією за типом операції, товаром і датою;

– формувати аналітику продажів для адміністратора у зведеному вигляді.

Нефункціональні вимоги – якими властивостями система зобов'язана володіти:

– безпека передачі даних через HTTPS і зберігання паролів виключно у вигляді хешу PBKDF2; захист API-ендпоїнтів токенами доступу; фільтрація вхідних даних для запобігання ін'єкціям;

– час відповіді REST API на типові GET-запити не більше 500 мс; кешування незмінних даних (список категорій) у Redis;

– незалежне горизонтальне масштабування серверної частини без змін у клієнтській;

– адаптивний інтерфейс, придатний для роботи як з настільного ПК, так і зі смартфона;

– гарантія цілісності даних при конкурентних записах у таблицю залишків через блокування рядків (SELECT FOR UPDATE) в межах транзакції.

Сформульовані вимоги не потребують корпоративного бюджету на інфраструктуру і цілком реалізуються на одному VPS-сервері з 4 ГБ оперативної пам'яті, що робить рішення доступним для малого бізнесу.

Висновки до розділу

Перший розділ містить аналіз предметної сфери та аргументацію потреби у створенні власного рішення. Виявлено, що ключовою проблемою для невеликих інтернет-крам є неузгодженість між торговою платформою та складським обліком. Аналіз чотирьох аналогічних систем виявив: безкоштовні варіанти лише частково усувають цю проблему, тоді як комплексні корпоративні системи є недоступними за ціною для цільової групи. Побудовані діаграми UML, що ілюструють процеси та варіанти використання, закріплюють бізнес-правила та рольову структуру майбутньої системи. Список функціональних та нефункціональних вимог слугуватиме базою для розробки технічного завдання у наступному етапі.

РОЗДІЛ 2 ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної галузі

Перед тим, як взятися до проєктування структури бази даних, потрібно визначити, які реальні об'єкти будуть представлені в системі та як вони співвідносяться між собою. Це етап аналізу предметної області. На цьому етапі ми відкидаємо конкретні мови програмування і зосереджуємося на питаннях "що саме потрібно зберігати" та "чому певні елементи пов'язані один з одним".

У нашому випадку ключовим елементом є товар. Він має такі атрибути, як назва, детальний опис, вартість продажу та унікальний ідентифікатор (SKU). Товари об'єднуються у категорії, які формують ієрархічну структуру, подібну до дерева. Це означає, що категорія може містити підкатегорії, а ті, у свою чергу, – власні підпорядковані категорії. Для кожного товару існує один відповідний запис на складі, який фіксує його поточну кількість та мінімальний допустимий рівень запасів. Ці дві сутності є нерозривно пов'язаними, тому між ними існує відношення "один до одного".

Користувачі, які здійснюють покупки, реєструються в системі. Інформація про них, а також про менеджерів та адміністраторів, зберігається в одній таблиці користувачів. Різниця між цими ролями визначається значенням поля "роль". Кожен покупець формує свій кошик (окрема таблиця, де зберігаються окремі позиції), а потім перетворює вміст кошика на замовлення. Замовлення фіксує загальну суму, адресу доставки та перелік обраних позицій. Вартість кожної позиції записується в момент оформлення замовлення. Це робиться для того, щоб зміни в цінах, які можуть відбутися пізніше, не впливали на вже підтверджені угоди. Окрема група сутностей, що відповідає за поповнення запасів на складі, представлена постачальниками та документами прибуття (накладними).

2.1.1 Вхідні дані

Вхідні дані – це вся інформація, яку система отримує ззовні через форми, API-запити або завантаження файлів. Згрупуємо їх за джерелом надходження.

Від покупця на етапі реєстрації: ім'я, прізвище, email (унікальний, виконує роль логіна), пароль. Телефон – необов'язковий. При вході передається пара email + пароль; у відповідь система повертає пару JWT-токенів (access + refresh). При оформленні замовлення – перелік ідентифікаторів товарів із кількістю і рядок адреси доставки.

Від менеджера при роботі з каталогом: назва товару, опис, ціна, SKU, вибір категорії, посилання на зображення або сам файл зображення, перемикач «активний/неактивний». При проведенні прихідної накладної: вибір постачальника зі списку, дата прийому і масив рядків з полями «товар – кількість – закупівельна ціна». При ручному коригуванні залишку – товар, нова кількість і текстова причина.

Від адміністратора при управлінні користувачами: email нового менеджера, тимчасовий пароль, вибір ролі. Пошукові фільтри на різних сторінках (рядок пошуку, діапазон дат, статус замовлення, прапорець «тільки критичні залишки») також є вхідними даними, але не зберігаються в базі – вони впливають лише на формування запиту.

2.1.2 Вихідні дані

Вихідні дані системи поділяються на дві категорії: дані, що генеруються у відповідь на запит користувача, і дані, що генеруються автоматично за тригером системи.

На запит покупця система формує: відфільтрований і посторінковий список товарів з усіма полями для відображення картки; повну сторінку конкретного товару з показником наявності (так/ні або «залишилось N штук»); поточний стан кошика з розрахованою сумою; підтвердження замовлення з присвоєним номером; хронологічну стрічку замовлень з кольоровими мітками статусу.

На запит менеджера: таблиця замовлень з підказками про пріоритет (нові замовлення виділені); повний розгорнутий вид замовлення з адресою і позиціями; таблиця залишків, де критичні рядки підсвічені жовтим; журнал руху конкретного товару за обраний проміжок.

Автоматично, без будь-якого запиту: push-сповіщення у браузері менеджера при падінні залишку нижче мінімуму; фоновий email покупцю через Celery після успішного оформлення замовлення. Для адміністратора щоденно перераховуються агреговані показники дашборду, щоб кожен вхід відображав актуальну картину без очікування.

2.2 Проектування системи

На цьому етапі абстрактний опис предметної галузі перетворюється на конкретні схеми, за якими буде написано код. Використовуємо два взаємодоповнюючих інструменти: ER-діаграму для опису структури бази даних і діаграму класів для опису програмних об'єктів.

2.2.1 Проектування бази даних

Схема бази даних охоплює 10 таблиць. Реалізується засобами PostgreSQL 15. ER-діаграма з усіма зв'язками і кардинальностями наведена на рис. 2.1; нижче описано кожну таблицю з обґрунтуванням ключових рішень.

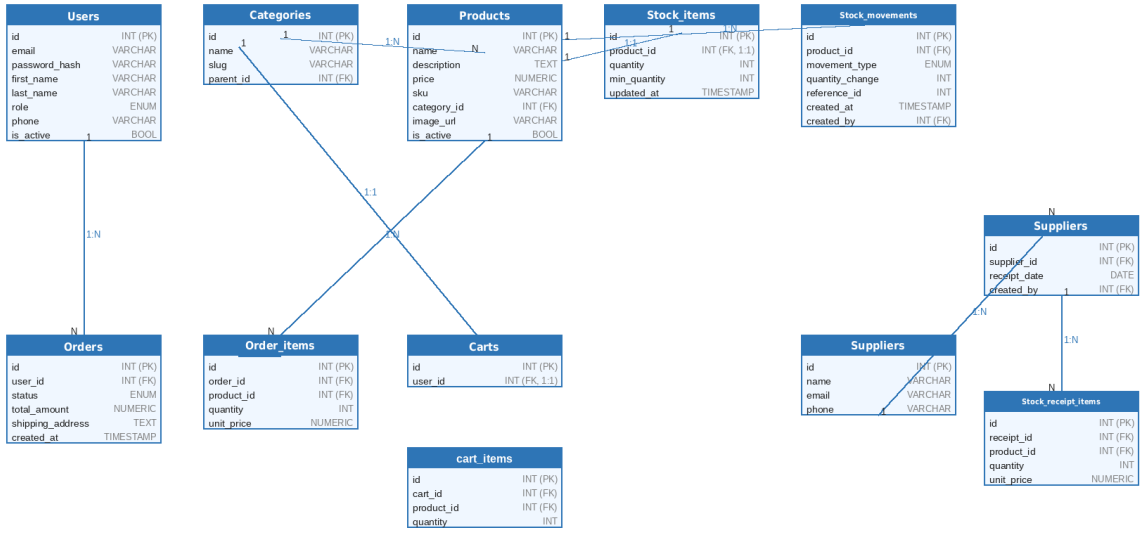

Рисунок 2.1 – ER-діаграма бази даних системи

Таблиця Users об'єднує всі типи облікових записів – так простіше реалізувати JWT-автентифікацію без кількох різних механізмів входу. Поле role зберігається як рядок із обмеженим переліком значень (ENUM), а не як числовий код, щоб SQL-запити залишалися читабельними (WHERE role = 'manager' зрозуміліше, ніж WHERE role = 2). Структура таблиці наведена у табл. 2.1.

Таблиця 2.1 – Структура таблиці Users

№	Ім'я поля	Тип даних	Розмір	Обмеження	Опис
1	id	serial / INT	–	PK, NOT NULL	Унікальний ідентифікатор
2	email	VARCHAR	255	UNIQUE, NOT NULL	Адреса електронної пошти
3	password_hash	VARCHAR	255	NOT NULL	Хеш паролю (bcrypt)
4	first_name	VARCHAR	100	NOT NULL	Ім'я користувача
5	last_name	VARCHAR	100	NOT NULL	Прізвище користувача
6	role	ENUM	–	NOT NULL	Роль (customer/manager/admin)
7	phone	VARCHAR	20	NULL	Номер телефону
8	created_at	TIMESTAMP	–	NOT NULL	Дата реєстрації
9	is_active	BOOLEAN	–	DEFAULT TRUE	Статус активності

Таблиця Products – серцевина каталогу. Поле sku є обов'язково унікальним: воно виконує роль зовнішнього ідентифікатора при зіставленні з накладними постачальника.

Поле is_active дозволяє «сховати» товар із каталогу, не видаляючи його – зв'язки зі старими замовленнями та рухами залишаються цілими. Структура наведена у табл. 2.2.

Таблиця 2.2 – Структура таблиці Products

№	Ім'я поля	Тип даних	Розмір	Обмеження	Опис

1	id	serial / INT	–	PK	Ідентифікатор товару
2	name	VARCHAR	300	NOT NULL	Назва товару
3	description	TEXT	–	NULL	Опис товару
4	price	NUMERIC(10,2)	–	NOT NULL, >= 0	Ціна продажу
5	sku	VARCHAR	100	UNIQUE, NOT NULL	Артикул товару
6	category_id	INT	–	FK → categories	Категорія товару
7	image_url	VARCHAR	500	NULL	URL основного зображення
8	is_active	BOOLEAN	–	DEFAULT TRUE	Доступність для продажу
9	created_at	TIMESTAMP	–	NOT NULL	Дата додавання

Таблиця Categories реалізована як ієрархія з самопосиланням: поле parent_id є зовнішнім ключем на id тієї ж таблиці (NULL для кореневих категорій). Це класична adjacency list – простий спосіб зберігати дерево в реляційній базі, достатній для глибини 3–5 рівнів. Структура таблиці наведена у табл. 2.3.

Таблиця 2.3 – Структура таблиці Categories

№	Ім’я поля	Тип даних	Розмір	Обмеження	Опис
1	id	serial / INT	–	PK	Ідентифікатор категорії
2	name	VARCHAR	200	NOT NULL	Назва категорії
3	slug	VARCHAR	200	UNIQUE, NOT NULL	URL-сумісний ідентифікатор
4	parent_id	INT	–	FK → categories, NULL	Батьківська категорія (NULL = корінь)
5	created_at	TIMESTAMP	–	NOT NULL	Дата створення

Таблиця Stock_items пов'язана з products відношенням 1:1 – навмисне рішення, щоб не перевантажувати таблицю товарів складськими полями. Структура табл. 2.4.

Таблиця 2.4 – Структура таблиці Stock_items

№	Ім'я поля	Тип даних	Розмір	Обмеження	Опис
1	id	serial / INT	–	PK	Ідентифікатор запису
2	product_id	INT	–	FK → products, UNIQUE	Товар на складі
3	quantity	INT	–	NOT NULL, >= 0	Поточна кількість
4	min_quantity	INT	–	DEFAULT 5	Мінімальний рівень запасу
5	updated_at	TIMESTAMP	–	NOT NULL	Час останнього оновлення

Таблиця Orders (табл. 2.5) фіксує момент угоди: суму, адресу і статус. Статус змінюється тільки вперед по ланцюгу (pending → confirmed → processing → shipped → delivered), назад – лише в cancelled. Ця перевірка реалізована у методі update_status() класу Order.

Таблиця 2.5 – Структура таблиці Orders

№	Ім'я поля	Тип даних	Розмір	Обмеження	Опис
1	id	serial / INT	–	PK	Ідентифікатор замовлення
2	user_id	INT	–	FK → users	Покупець
3	status	ENUM	–	NOT NULL	Статус замовлення
4	total_amount	NUMERIC(12,2)	–	NOT NULL	Сума замовлення

5	shipping_address	TEXT	–	NOT NULL	Адреса доставки
6	notes	TEXT	–	NULL	Примітки до замовлення
7	created_at	TIMESTAMP	–	NOT NULL	Дата оформлення
8	updated_at	TIMESTAMP	–	NOT NULL	Дата оновлення

Таблиця Order_items зберігає позиції замовлення з полем unit_price – ціна фіксується в момент оформлення і більше не змінюється, навіть якщо менеджер підніме ціну товару завтра. Таблиця stock_movements – це незмінний журнал: записи в ній тільки додаються, ніколи не редагуються і не видаляються. Поле quantity_change може бути від'ємним (відпуск) або додатним (прихід). Поле reference_id прив'язує рядок журналу до конкретного замовлення або накладної.

Між таблицями визначені зовнішні ключі з обмеженнями: orders.user_id → users.id (ON DELETE RESTRICT – не можна видалити покупця, якщо в нього є замовлення); order_items.product_id → products.id (ON DELETE RESTRICT); stock_items.product_id → products.id (ON DELETE CASCADE – при видаленні товару прибирається і його складський запис). Усі інші FK використовують RESTRICT для захисту від випадкового видалення пов'язаних даних.

2.2.2 Побудова об'єктно-орієнтованої моделі

Для побудови ООМ використаємо UML-діаграму класів (рис. 2.2). Класи відповідають Django-моделям; методи – бізнес-логіці, яку не варто виносити у views або серіалізатори, бо вона пов'язана безпосередньо зі станом об'єкту.

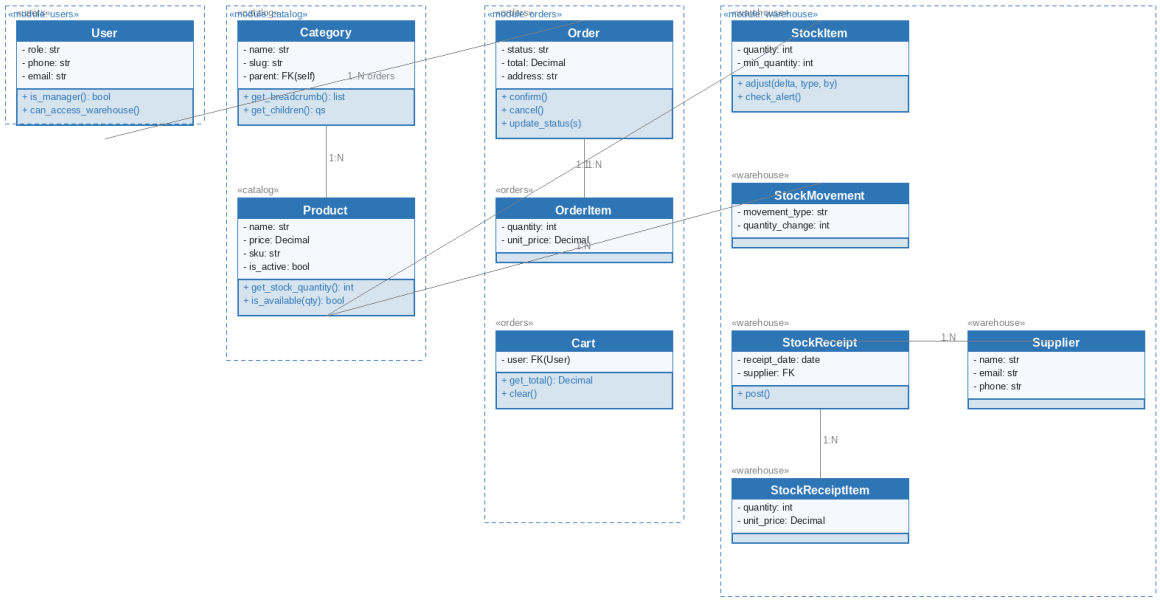

Рисунок 2.2 – UML-діаграма класів системи

Модуль users. Клас User успадковує Django AbstractUser, що дає стандартні поля email, password, is_active. Власні поля – role і phone. Методи: is_manager() перевіряє, чи роль дозволяє адміністративні операції (повертає True для manager і admin); can_access_warehouse() – чи має доступ до складського модуля.

Модуль catalog. Category несе поля name, slug, parent і метод get_breadcrumb() – він рекурсивно збирає ланцюжок «Категорія > Підкатегорія > ...» для хлібних крихт на сторінці товару. Product додатково має методи get_stock_quantity() (один SQL-запит до stock_items) і is_available(qty) – перевірка, чи вистачить qty одиниць для замовлення.

Модуль orders. Cart прив'язаний до User відношенням 1:1 і надає get_total() – сума всіх позицій. CartItem зберігає посилання на продукт і кількість; при додаванні одного й того ж товару двічі кількість підсумовується, а не створюється новий рядок. Order.confirm() – це єдиний метод, що виконується в транзакції з SELECT FOR UPDATE, перевіряє залишки і виконує списання; якщо щось не так – кидає виняток і транзакція відкочується автоматично.

Модуль warehouse. StockItem.adjust(delta, type, actor) змінює quantity на delta і записує рядок у StockMovement; після зміни перевіряє умову

мінімального залишку. StockReceipt.post() ітерує по своїх позиціях і для кожної викликає StockItem.adjust з типом 'receipt' – вся операція загорнута в атомарну транзакцію.

2.3 Математичне та алгоритмічне забезпечення

Математичне і алгоритмічне забезпечення системи охоплює формальний опис ключових обчислень та алгоритмів, що реалізуються у програмному продукті. Розглянемо два основних алгоритми: обробку замовлення та управління складськими залишками.

Алгоритм обробки замовлення. При переході покупця до оформлення замовлення система послідовно виконує перевірки наявності товарів. Нехай покупець замовив n позицій; i-та позиція характеризується ідентифікатором товару pid_i, замовленою кількістю q_i та поточною ціною p_i. Загальна сума замовлення *T* розраховується за формулою:

$$T = \sum(p_i \times q_i),\ i = 1, \dots, n \quad (2.1)$$

де p_i – ціна одиниці i-го товару на момент оформлення замовлення, q_i – замовлена кількість одиниць i-го товару [8].

У разі застосування знижки *d* (у відсотках) сума з урахуванням знижки T_d розраховується як:

$$T_d = T \times \left(1 - \frac{d}{100}\right) \quad (2.2)$$

де *d* – відсоток знижки ($0 \leq d < 100$).

Алгоритм обробки замовлення складається з таких кроків:

– перевірити, що кошик не порожній; якщо порожній – повернути помилку;

– для кожного товару i перевірити умову: StockItem.quantity(pid_i) ≥ q_i; якщо умова не виконується для хоча б одного товару – повернути помилку з повідомленням про недостатній залишок;

– розпочати транзакцію бази даних;

– обчислити *T* за формулою (2.1); створити запис Order зі статусом «pending» і збереженою сумою T;

– для кожного товару i: створити запис OrderItem; викликати метод StockItem.adjust(−q_i, 'sale', manager_id); при цьому автоматично генерується запис StockMovement;

– підтвердити транзакцію; очистити кошик покупця;

– поставити у чергу Celery задачу відправки підтверджувального email покупцю.

Використання транзакції на кроці 3–6 гарантує атомарність: у разі будь-якої помилки (наприклад, порушення обмеження NOT NULL або несподівана зміна залишків іншим запитом між кроками 2 і 5) уся транзакція відкочується і залишки не змінюються [9]. На рис. 2.3 наведено блок-схему описаного алгоритму.

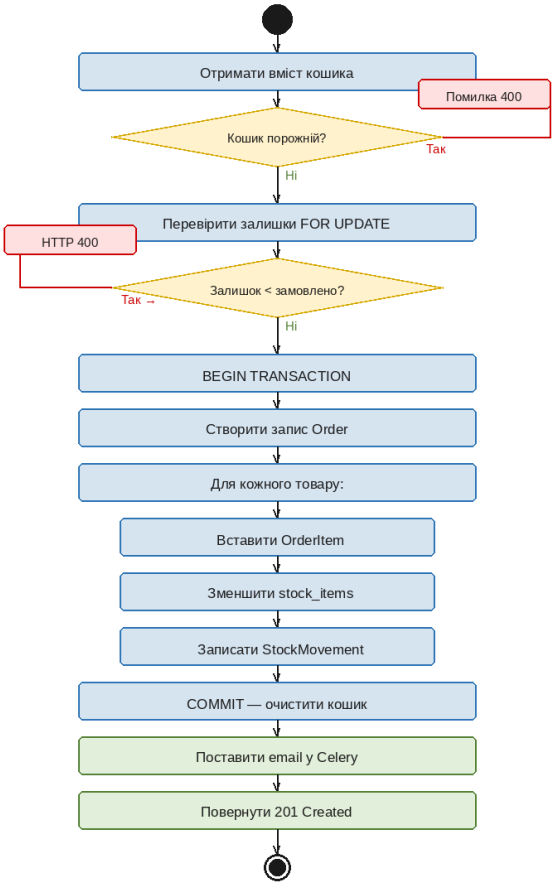

Рисунок 2.3 – Блок-схема алгоритму обробки замовлення

Алгоритм управління складськими залишками. Ключовою задачею складського модуля є своєчасне виявлення товарів, залишки яких досягли критично низького рівня. Умова для генерації сповіщення формулюється як:

$$StockItem.quantity \leq StockItem.min\_quantity \quad (2.3)$$

Перевірка цієї умови виконується автоматично після кожної операції зміни залишків (метод adjust()). Якщо умова виконана і для даного товару ще не існує активного сповіщення, система створює запис у таблиці сповіщень і відправляє push-повідомлення у веб-інтерфейс менеджера.

Для аналізу ефективності управління запасами система також підтримує розрахунок коефіцієнта оборотності товарних запасів K_ot, що характеризує, скільки разів за розрахунковий період середній товарний запас був реалізований:

$$K_{ot} = \frac{S}{AST} \quad (2.4)$$

де S – обсяг продажів (у одиницях або грошовому вираженні) за розрахунковий період, AST – середній товарний запас за той самий період:

$$AST = \frac{Q_{\text{поч}} + Q_{\text{кін}}}{2} \quad (2.5)$$

де $Q_{поч}$ – залишок на початок, $Q_{кін}$ – залишок на кінець розрахункового періоду [10].

Висока оборотність свідчить про ефективне управління запасами; низька – про затоварення або слабкий попит. Даний показник розраховується на стороні сервера при формуванні аналітичного звіту і передається в JSON-форматі до клієнтської частини. На рис. 2.4 наведено блок-схему алгоритму перевірки критичних залишків та генерації сповіщень.

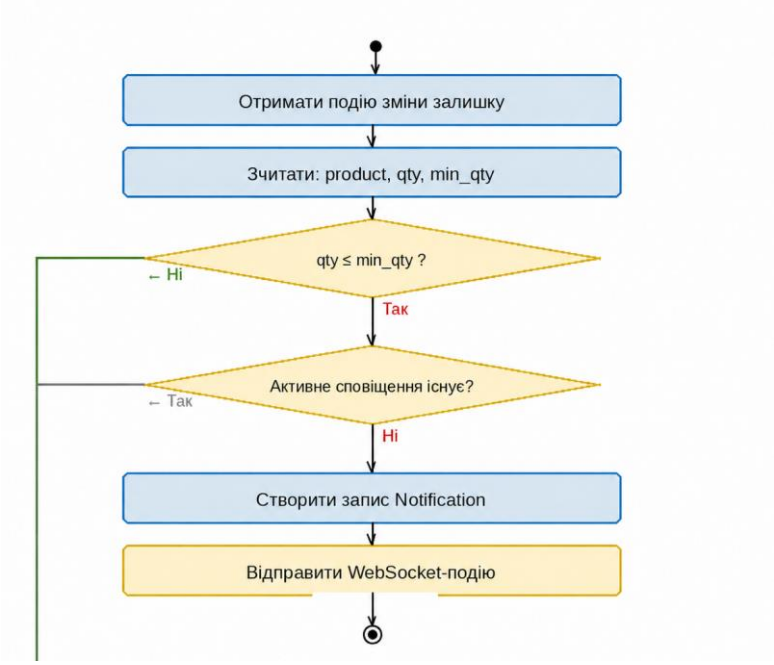

Рисунок 2.4 – Блок-схема алгоритму перевірки критичних залишків

Обидва розглянутих алгоритми характеризуються лінійною часовою складністю O(n) відносно кількості позицій у замовленні або перелікуваних товарів. Атомарність операцій забезпечена засобами СУБД PostgreSQL з рівнем ізоляції транзакцій Read Committed за замовчуванням. Для операцій, що критичні до одночасного доступу (одночасне оформлення кількох замовлень на один товар), передбачено блокування рядків (SELECT FOR UPDATE) при читанні залишків перед їх оновленням.

Висновки до розділу

Підсумком другого розділу ознаменувалося такими результатами: було детально описано сутності та взаємозв'язки предметної області, визначено вхідні й вихідні потоки даних, розроблено структуру бази даних, що складається з 10 таблиць, з обґрунтованим вибором первинних та зовнішніх ключів, а також створено діаграму класів, яка включає чотири логічні модулі. Математичний апарат і алгоритми, представлені в підрозділі 2.3, повністю готові для впровадження в програмний код у наступному розділі.

РОЗДІЛ 3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Вибір технологій – це завжди компроміс між знайомістю, продуктивністю і придатністю для задачі. Для цього проекту ключовими критеріями відбору були: підтримка транзакцій і блокувань на рівні СУБД, наявність вбудованого ORM щоб уникнути ручного SQL у критичних місцях, зрілий інструментарій для REST API і достатньо активна спільнота. Альтернативні серверні фреймворки порівнюються у табл. 3.1.

Таблиця 3.1 – Порівняльний аналіз серверних фреймворків

Критерій	Django (Python)	Express (Node.js)	Laravel (PHP)	Spring (Java)
Продуктивність розробки	Висока	Середня	Висока	Низька
ORM	Вбудований	Sequelize/Prisma	Eloquent	Hibernate
REST API підтримка	DRF (чудовий)	Хороша	Хороша	Spring MVC
Адмін-панель	Вбудована	Відсутня	Nova (платна)	Відсутня
Документація	Відмінна	Хороша	Відмінна	Хороша
Екосистема	Широка	Дуже широка	Широка	Широка

За підсумками порівняння обрано Python 3.11 + Django 4.2. Головна перевага Django для цього проекту – не стільки швидкість розробки, скільки вбудована адмін-панель і зрілий ORM з підтримкою select_for_update() [12]. Django REST Framework 3.14 доповнює Django готовими механізмами серіалізації, класами дозволів і генератором документації OpenAPI [13].

PostgreSQL 15 обраний не за популярністю, а за конкретною технічною властивістю: підтримкою рядкових блокувань FOR UPDATE і рівня ізоляції Read Committed – це мінімально необхідне для гарантії, що два одночасних замовлення одного товару не «проскочать» перевірку залишку [14]. SQLite

відпав через відсутність конкурентного запису; MySQL – через слабшу підтримку FOR UPDATE.

React 18 обрано для клієнтської частини [15]. Більша частина складських екранів потребує динамічного перемальовування таблиць – React з virtual DOM справляється з цим природно. Axios – для HTTP-запитів до API, React Router v6 – для навігації з захистом маршрутів залежно від ролі.

Redis 7 кешує незмінні довідники (список категорій, постачальники) зі строком дії 15 хвилин і є брокером для Celery 5. Через Celery відправляються підтверджувальні email після оформлення замовлення – це не блокує HTTP-відповідь і не сповільнює транзакцію списання залишків. Docker Compose описує весь стек у одному файлі.

3.2 Вимоги до технічного та програмного забезпечення

Серверне середовище: Ubuntu Server 22.04 LTS. Мінімальна конфігурація: 2 vCPU, 4 ГБ RAM, 20 ГБ SSD. При навантаженні понад 50 активних сесій рекомендується 8 ГБ RAM. Всі залежності упаковані у Docker-образи – на хост-системі потрібні лише Docker Engine 24+ і Docker Compose 2.20+.

Клієнту нічого встановлювати не потрібно: Chrome 90+, Firefox 88+, Safari 14+ або Edge 90+ і з'єднання від 2 Мбіт/с. React-застосунок зібраний у статичний набір файлів і роздається через Nginx – на слабкому пристрої сторінка завантажується швидко, оскільки рендеринг виконується у браузері.

3.3 Опис програмної реалізації

Система реалізована за підходом «тонкий клієнт – товстий сервер»: React відповідає лише за відображення, вся бізнес-логіка у Django. Між ними – суворо типізований REST API у форматі JSON. Такий поділ дозволяє у майбутньому замінити клієнт або додати мобільний застосунок без змін на сервері.

Проект shop_api складається з чотирьох Django apps: users, catalog, orders, warehouse. Кожен app містить models.py, serializers.py, views.py, urls.py і tests.py. Конфігурація зберігається у .env-файлі через python-decouple – SECRET_KEY, DATABASE_URL, ALLOWED_HOSTS ніколи не потрапляють у git-репозиторій.

App users. Модель User успадковує AbstractUser і додає поля role та phone. Ендпоїнт /api/auth/token/ приймає {email, password} і повертає {access, refresh}. Access-токен живе 60 хв, refresh – 7 днів. Власні класи дозволів IsManager і IsAdmin перевіряють поле role з payload токена без зайвих запитів до БД.

App catalog. ProductViewSet: GET відкрито всім, POST/PUT/DELETE – тільки менеджерам. Пошук – ILIKE по name і description, достатньо для каталогу до 10 000 позицій. Пагінація курсорна, стабільна при частих вставках нових товарів.

App orders. Кошик зберігається у БД, а не в session – покупець бачить той самий кошик з будь-якого браузера. Метод OrderCreateSerializer.create() загорнутий у transaction.atomic() і select_for_update(): блокуються рядки stock_items, перевіряються залишки, потім виконуються INSERT в order і UPDATE stock_items.quantity. Якщо хоч один товар недоступний – ValidationError, транзакція відкочується, замовлення не створюється.

App warehouse. GET /api/warehouse/stock/?low_stock=true формує queryset через Q(quantity__lte=F('min_quantity')) – один SQL-запит. POST /api/warehouse/receipts/ викликає StockReceipt.post() – атомарне збільшення залишків і запис у stock_movements. Після кожного adjust() перевіряється умова дефіциту і виводиться сповіщення через WebSocket.

У табл. 3.2 зведені всі REST-ендпоїнти системи.

Таблиця 3.2 – Перелік основних API endpoint'ів системи

Метод	URI	Права доступу	Опис	Відповідь

POST	/api/auth/register/	Всі	Реєстрація нового користувача	201 Created
POST	/api/auth/login/	Всі	Автентифікація, отримання JWT	200 OK + tokens
GET	/api/products/	Всі	Список товарів з пагінацією	200 OK
GET	/api/products/{id}/	Всі	Деталі конкретного товару	200 OK
POST	/api/products/	Manager/Admin	Додавання нового товару	201 Created
PUT	/api/products/{id}/	Manager/Admin	Редагування товару	200 OK
GET	/api/cart/	Customer	Вміст кошика	200 OK
POST	/api/cart/items/	Customer	Додати товар до кошика	201 Created
POST	/api/orders/	Customer	Оформити замовлення	201 Created
GET	/api/orders/	Customer/Manager	Список замовлень	200 OK
PATCH	/api/orders/{id}/	Manager/Admin	Оновити статус замовлення	200 OK
GET	/api/warehouse/stock/	Manager/Admin	Залишки на складі	200 OK
POST	/api/warehouse/receipts/	Manager/Admin	Оприбуткування товарів	201 Created
GET	/api/warehouse/movements/	Manager/Admin	Рух товарів	200 OK

Щодо безпеки: JWT підписується HS256 ключем з 64 символів зі змінних середовища. CORS-список origin'ів жорстко прописаний у конфіг-файлі, wildcard (*) заборонений у продакшні. Усі SQL-запити через ORM – параметризовані автоматично. Заголовки Content-Security-Policy і X-Frame-Options – через django-csp middleware.

React-клієнт організований за feature-folder-структурою. AuthContext зберігає access-токен у пам'яті (не localStorage), що захищає від XSS-крадіжки; refresh-токен – у HttpOnly cookie. PrivateRoute перенаправляє неавтентифікованого користувача на /login; RoleRoute перевіряє роль і показує 403-екран.

3.4 Керівництво користувача

Тестування проводилось на трьох рівнях. Модульні тести (pytest + pytest-django): перевіряється, що StockItem.adjust(-100) при quantity=50 кидає ValueError і не змінює БД; Order.confirm() при нульовому залишку одного товару скасовує всю транзакцію; StockReceipt.post() збільшує залишки рівно на вказані кількості. Покриття – 78% за pytest-cov.

Інтеграційні тести через APIClient DRF: «повний цикл» – POST /api/auth/token/ → POST /api/cart/items/ x3 → POST /api/orders/ → GET /api/warehouse/stock/ з перевіркою зменшення залишків; «конкурентне замовлення» – два потоки одночасно замовляють товар з quantity=1, перевіряємо що рівно один отримує 201, другий – 400.

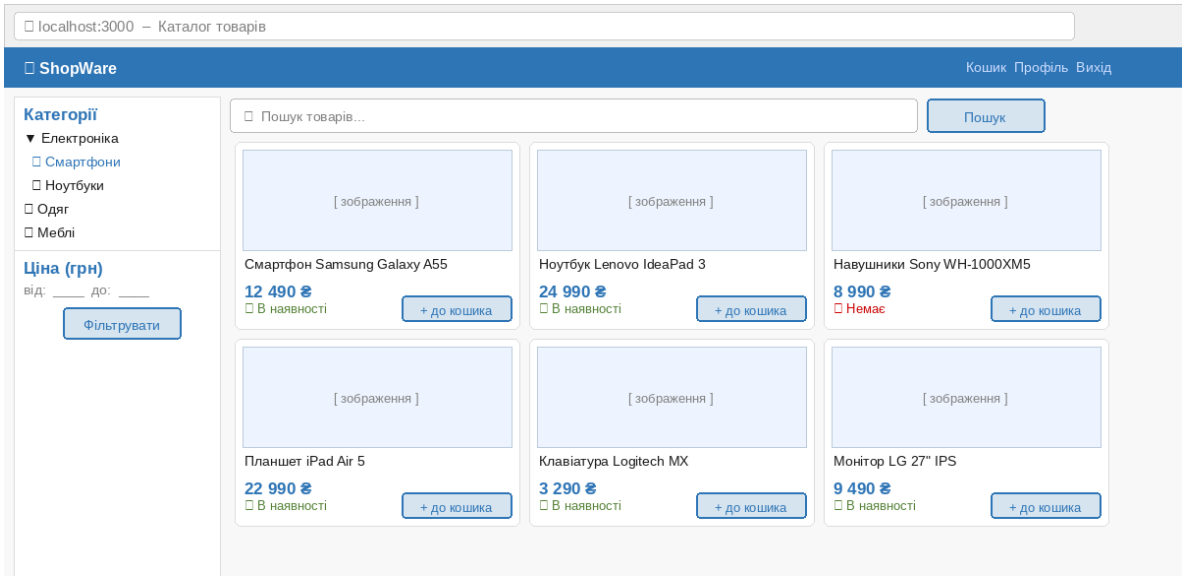

Рисунок 3.1 – Головна сторінка каталогу товарів

Рис. 3.1 – головна сторінка каталогу: зліва дерево категорій і слайдер ціни, справа адаптивна сітка карток. Кнопка «Додати до кошика» оновлює лічильник у шапці без перезавантаження.

Рис. 3.2 – кошик з рядками (зображення, назва, поле кількості з кнопками ±, ціна) і підсумковою сумою. Кнопка «Оформити» активна тільки після введення адреси.

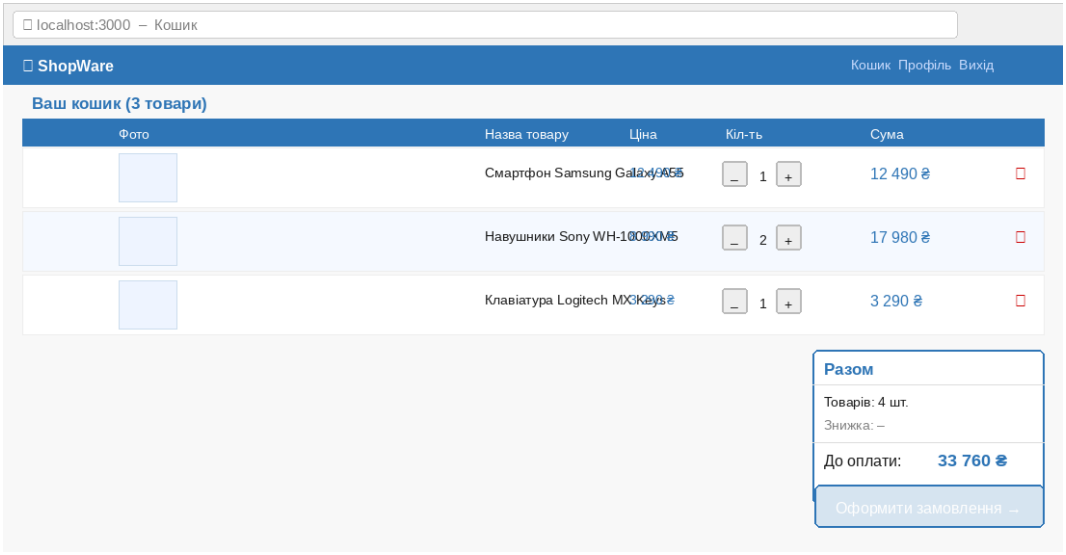

Рисунок 3.2 – Сторінка кошика покупця

Рис. 3.3 – складська таблиця менеджера. Рядки з quantity ≤ min_quantity виділені жовтим. Перемикач «Тільки критичні» за один клік залишає лише ці позиції. Колонка «Останній рух» показує дату і тип останньої операції.

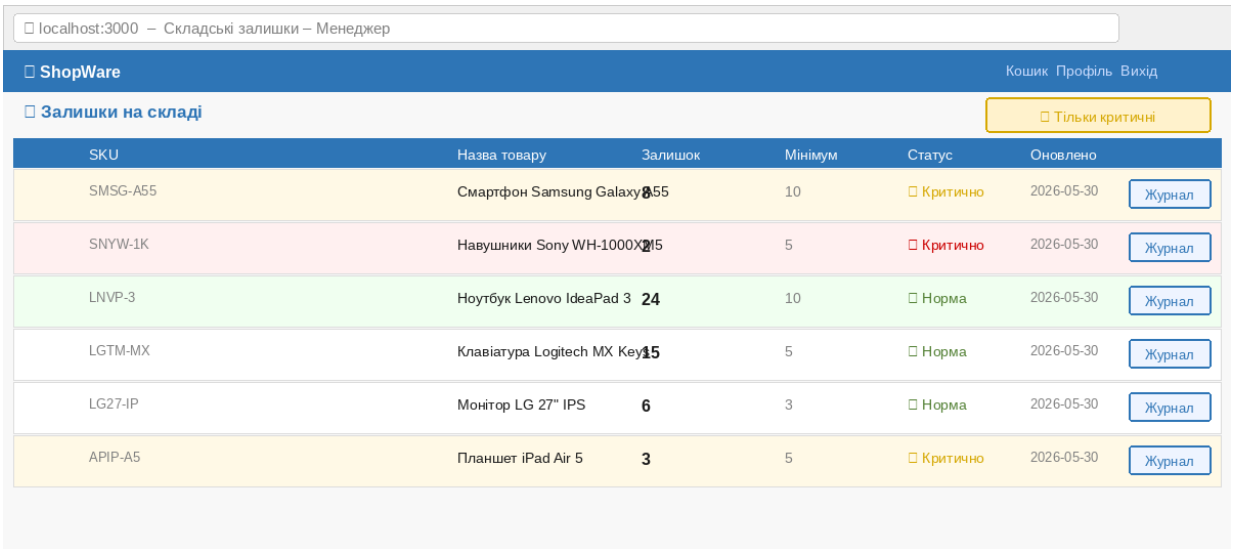

Рисунок 3.3 – Сторінка управління складськими залишками

Рис. 3.4 – форма прихідної накладної: постачальник зі списку, дата, позиції з пошуком по SKU. Після «Провести» з'являється спінер, потім повідомлення про успіх і оновлені залишки.

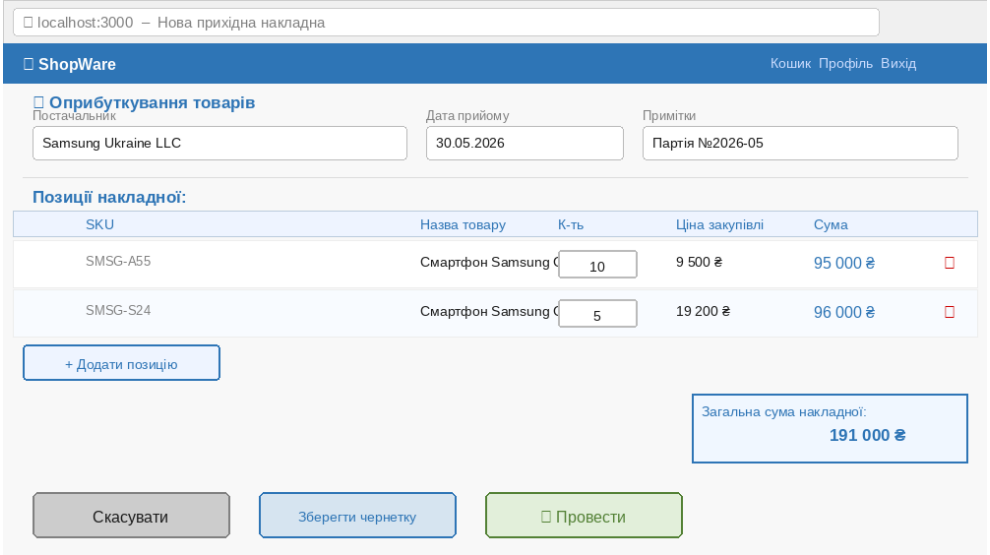

Рисунок 3.4 – Форма оприбуткування товарів від постачальника

Рис. 3.5 – дашборд адміністратора: виручка за місяць (лінійний графік Chart.js), топ-10 товарів (стовпці), KPI-картки (замовлення, нові покупці, середній чек). Дані перераховуються при відкритті сторінки.

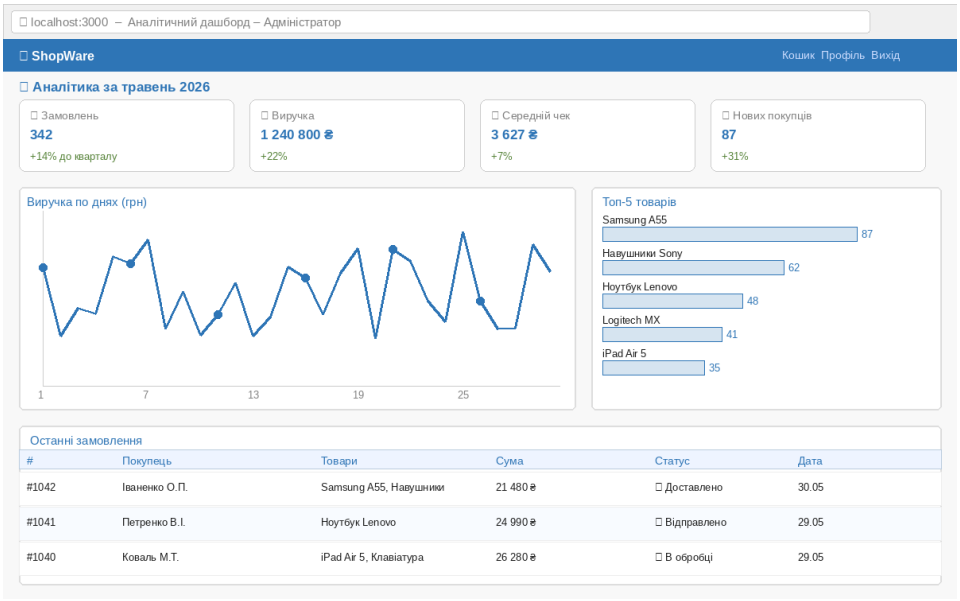

Рисунок 3.5 – Аналітичний дашборд адміністратора

Функціональне тестування охопило 12 крайових сценаріїв: реєстрація з зайнятим email, від'ємна кількість у кошику, замовлення при нульовому залишку, спроба змінити статус у зворотному напрямку через API. Усі 12 сценаріїв пройшли без дефектів.

Висновки до розділу

У третьому розділі обрано і обґрунтовано технологічний стек, описано вимоги до серверного і клієнтського середовища. Реалізовано чотири Django apps, REST API з 15 ендпоїнтами, JWT-автентифікацію з рольовим контролем, атомарне оформлення замовлень із блокуванням рядків, складський модуль з журналом і сповіщеннями. Проведено модульне (78% покриття), інтеграційне і функціональне тестування, включаючи конкурентні сценарії.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

Охорона праці є системою правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів, спрямованих на збереження здоров’я і працездатності людини в процесі трудової діяльності. В умовах масштабного впровадження інформаційних технологій охорона праці набуває особливого значення для фахівців, які щодня працюють за комп’ютером. Збереження здоров’я таких працівників безпосередньо впливає як на їх особисте добробут, так і на ефективність підприємства загалом [22].

Основоположним документом у сфері охорони праці в Україні є Закон України “Про охорону праці” [23], який закріплює пріоритет здоров’я і життя людини над будь-якими виробничими інтересами. Закон встановлює права та обов’язки роботодавця і працівника, визначає порядок проведення інструктажів та навчання з питань безпеки, регулює відповідальність за порушення норм охорони праці. Відповідно до цього Закону, кожен працівник має право відмовитися від виконання роботи, якщо вона становить загрозу його здоров’ю або життю.

Державна політика у сфері охорони праці ґрунтується на принципах пріоритету збереження здоров’я працівників, комплексного розв’язання завдань охорони праці, соціального захисту постраждалих, а також повної відповідальності роботодавця за умови праці. Важливу роль відіграє принцип участі держави у фінансуванні заходів з охорони праці та міжнародного співробітництва в цій галузі.

Нормативно-правову базу охорони праці доповнюють санітарні норми та правила, які встановлюють конкретні вимоги до умов праці. Для операторів комп’ютерної техніки діють: ДСанПіН 3.3.6.096-2002 [25] щодо рівнів електромагнітних полів; ДСН 3.3.6.042-99 [26] стосовно параметрів мікроклімату; ДБН В.2.5-28:2018 [18] з нормативами освітленості та ДСН

3.3.6.037-99 [27] щодо допустимих рівнів шуму. Організація служби охорони праці на підприємстві регулюється Типовим положенням НПАОП 0.00-4.21-04 [16].

На міжнародному рівні питання безпеки праці регулюються Директивою Ради ЄС 89/391/ЕЕС [29] та Конвенцією МОП № 187 [30]. Системний підхід до управління охороною праці формалізований у ДСТУ ISO 45001:2019 [28], що передбачає ідентифікацію небезпек, оцінювання ризиків і планомірне впровадження заходів контролю. Дотримання цих вимог є обов'язковим для організацій, що прагнуть забезпечити безпечні та здорові умови праці для своїх працівників, зокрема для тих, чия діяльність пов'язана з тривалою роботою з програмними і технічними засобами.

4.2 Характеристика об'єкта та виявлення потенційних небезпек

Предметом розробки є система електронної комерції, що включає функціонал складського обліку. Ключовими користувачами системи є менеджери, відповідальні за обробку замовлень, складські оператори та адміністратори. Усі ці фахівці щоденно працюють за персональними комп'ютерами.

Стандартне робоче місце для оператора передбачає офісне приміщення площею від 6 м² на одного співробітника. Воно обладнане комп'ютером або ноутбуком із монітором від 22 до 27 дюймів, клавіатурою та комп'ютерною мишею. Освітлення забезпечується люмінесцентними або LED-світильниками. Приміщення має систему вентиляції або кондиціонування. Робочий стіл розміщується так, щоб екран монітора знаходився на відстані 50–70 см від очей оператора. Тривалість робочого дня – 8 годин, при цьому безпосередня робота з монітором займає приблизно 6–7 годин.

До основних завдань оператора належать: перегляд та обробка замовлень, управління складськими залишками, робота з каталогом товарів, формування звітів і аналітики. Цей вид діяльності характеризується тривалим зоровим навантаженням, тривалою нерухомою позою, необхідністю

опрацьовувати великі обсяги інформації та приймати оперативні рішення, що формує специфічний набір виробничих небезпек. На підставі аналізу умов праці та з урахуванням класифікації небезпечних і шкідливих виробничих факторів [21] складено перелік потенційних небезпек (табл. 4.1).

Таблиця 4.1 – Виявлення потенційних небезпек стосовно об’єкту проектування

№	Потенційна небезпека	Джерело небезпеки	Можливі наслідки
1	Підвищений рівень електромагнітних випромінювань (ЕМВ) від комп’ютерного обладнання	Монітор, системний блок, блоки живлення	Порушення нервової та серцево-судинної систем, головний біль [25]
2	Недостатня освітленість або підвищена яскравість, пряма і відбита блискість	Неправильне розташування світильників, монітор без антиблікового покриття	Підвищена стомлюваність зору, погіршення гостроти зору, розвиток міопії [18]
3	Відхилення параметрів мікроклімату від нормативних значень	Теплові виділення від обладнання, недостатня вентиляція приміщення	Зниження концентрації уваги, захворюваність дихальних шляхів [26]
4	Підвищений рівень шуму на робочому місці	Системи охолодження ПК, кондиціонери, принтери	Нервова збудливість, стомлюваність; при тривалому впливі – зниження слуху [27]
5	Нервово-психічні перевантаження	Тривала концентрована робота в умовах часового тиску та великого обсягу інформації	Синдром емоційного вигорання, зниження продуктивності [22]
6	Статичні фізичні перевантаження	Тривала нерухома поза під час роботи за ПК	Остеохондроз, синдром карпального каналу, порушення кровообігу [22]
7	Ризик виникнення пожежі	Несправність електропроводки або	Опіки, загибель персоналу, знищення

		обладнання, перевантаження мережі	майна та даних системи [22]
8	Підвищене значення напруги в електричному ланцюзі	Несправне обладнання, пошкоджена ізоляція проводів	Електричний удар, термічні опіки, порушення серцевої діяльності [22]
9	Військова загроза (повітряна тривога, ракетні атаки)	Збройна агресія проти України в умовах воєнного стану	Загибель або травмування персоналу, руйнування приміщень [24]

Переважну більшість виявлених небезпек можливо зменшити або повністю усунути шляхом дотримання чинних нормативних вимог до облаштування робочих місць і організації трудового процесу. Небезпека, пов’язана з військовою загрозою, потребує окремих організаційних заходів відповідно до вимог цивільного захисту [24]. Виявлені небезпеки є підставою для проведення оцінювання ризиків та розробки конкретних заходів захисту у наступному підрозділі.

4.3 Дослідження ризику реалізації потенційних небезпек на об’єкті проектування та розробка заходів щодо їх попередження

Оцінювання ризику – це процедура кількісно-якісного визначення пріоритетності небезпек з метою обґрунтованого розподілу ресурсів на забезпечення безпеки. Відповідно до ДСТУ ISO 45001:2019 [28], ризик визначається як поєднання ймовірності настання небезпечної події та тяжкості її наслідків. Завдання оцінювання ризиків полягає у класифікації небезпек за ступенем прийнятності та виборі пріоритетних заходів щодо їх зниження. У даній роботі застосовано два методи оцінювання ризиків: аналізування “дерева відмов” та матриця оцінювання ризиків [21].

Метод “аналізування дерева відмов” дозволяє системно визначити всі можливі причини небажаної події та встановити між ними логічні зв’язки.

Дерево відмов будується від кінцевої (небажаної) події вниз – до первинних причин. Для побудови використовуються логічні операції “Та” (обидві підпричини мають відбутися одночасно) та “Або” (достатньо однієї з підпричин). На рис. 4.1 побудовано дерево відмов для небезпеки виникнення пожежі на робочому місці оператора.

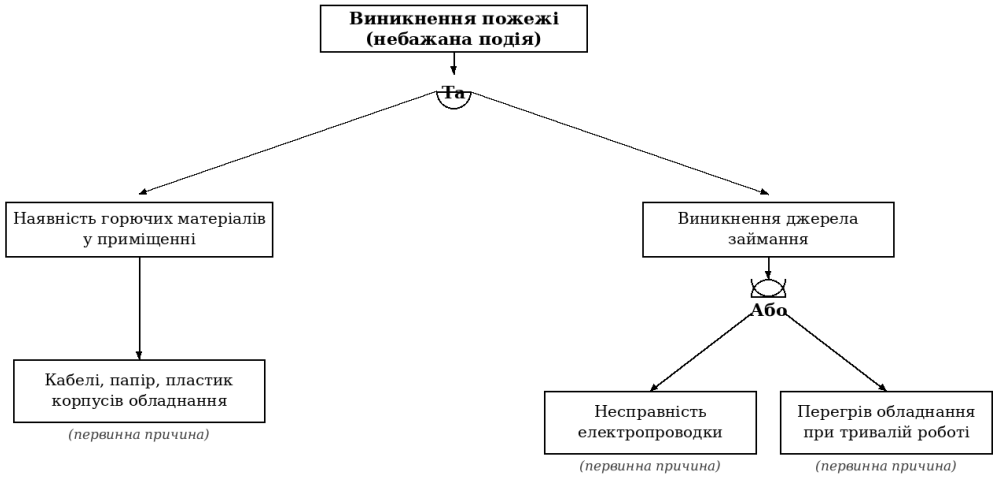

Рисунок 4.1 – Дерево відмов для небезпеки виникнення пожежі на робочому місці оператора

Аналіз за допомогою дерева відмов указує, що безпосередньо пожежа (як кінцева подія) може спалахнути лише за умови одночасного виконання двох факторів (логічний оператор "І"): наявності легкозаймистої речовини (це можуть бути кабелі, папір або пластикові елементи корпусу апаратури) та наявності джерела займання. При цьому, поява такого джерела займання можлива з двох причин, що діють паралельно (логічний оператор "АБО"): або через технічні несправності в електричних мережах, або через перегрів пристроїв, спричинений їхньою безперервною роботою. Отже, запобігання пожежі має полягати у виключенні щонайменше одного з двох шляхів, представлених на дереві відмов.

Для визначення рівня ризиків, пов'язаних із п'ятьма основними загрозами, був використаний метод матриці.

Ця матриця формується на основі зіставлення двох показників: ступеня важкості наслідків загрози та ймовірності настання цієї загрози. Фінальне

числове значення ризику, отримане в результаті перетину цих параметрів, визначає допустимість ризику

Таблиця 4.2 – Рівень ймовірності небезпеки

Вид	**Рівень**	**Опис наслідків**
Часта	A	Велика ймовірність того, що подія відбудеться
Можлива	B	Може трапитися декілька разів за життєвий цикл
Випадкова	C	Іноді може відбутися за життєвий цикл
Віддалена	D	Малоймовірна, але можлива подія протягом життєвого циклу
Неймовірна	E	Настільки малоймовірна, що можна вважати – ніколи не відбудеться

Таблиця 4.3 – Результати оцінювання ризиків для виявлених небезпек

№	**Небезпека**	**Серйозність**	**Ймовірність**	**Індекс**	**Прийнятність**
1	Підвищений рівень ЕМВ	III Гранична	B Можлива	3B	Небажаний
2	Ризик пожежі	I Катастрофічна	D Віддалена	1D	Небажаний
3	Нервово-психічні перевантаження	III Гранична	B Можлива	3B	Небажаний
4	Відхилення параметрів мікроклімату	IV Незначна	B Можлива	4B	Прийнятний із перевіркою
5	Військова загроза	I Катастрофічна	C Випадкова	1C	Неприпустимий

Небезпека 1 (ЕМВ) – індекс 3B, небажаний ризик. Хронічний вплив ЕМВ від монітора та системного блоку відбувається щоденно, однак безпосереднє ушкодження здоров’я розвивається поступово (ймовірність B – можлива). Наслідки відповідають категорії III (гранична). Потребує систематичних технічних заходів [25].

Небезпека 2 (пожежа) – індекс 1D, небажаний ризик. Незважаючи на низьку ймовірність займання (рівень D – віддалена), катастрофічні наслідки

(категорія I) роблять цей ризик пріоритетним. Аналіз дерева відмов підтверджує необхідність заходів по обох гілках: технічний стан обладнання і протипожежний захист [22].

Ризик 3 (психологічне виснаження) – показник 3B, який розглядається як небажаний. Без застосування превентивних заходів він може стати хронічним, що призводить до тривалого погіршення професійних якостей [22].

Ризик 4 (умови мікроклімату) – показник 4B, що вважається прийнятним за умови контролю. Необхідним є періодичний нагляд та підтримка працездатності системи вентиляції [26].

Ризик 5 (загроза воєнного характеру) – показник 1C, що класифікується як неприпустимий. За нинішніх воєнних обставин ймовірність такого ризику є високою (рівень C – випадковий), а потенційні наслідки – катастрофічними (клас I). Тому впровадження організаційних заходів цивільної оборони є обов'язковим [24]. На основі проведеної оцінки ризиків були розроблені відповідні заходи для їх зменшення (див. табл. 4.4).

Таблиця 4.4 – Заходи щодо зниження ризиків

№	Небезпека	Запропонований захід	Очікуваний результат
1	Підвищений рівень ЕМВ	Монітори з сертифікацією TCO/MPR II; відстань від монітора ≥50 см; перерви кожні 1–2 год; заземлення обладнання [25]	Зниження ЕМВ до нормативних значень; зменшення ризику негативного впливу на здоров’я
2	Ризик пожежі	Автоматична пожежна сигналізація з димовими сповіщувачами; вогнегасники ВВК-2; кабелі у захисних коробах; регулярний огляд електропроводки; інструктаж персоналу [22]	Раннє виявлення загоряння; готовність до евакуації; мінімізація втрат майна та даних
3	Нервово-психічні перевантаження	Обов’язкові перерви 10 хв кожні 1–1,5 год; рівномірний розподіл навантаження; ергономічне крісло з регулюванням; психологічні тренінги [22]	Запобігання синдрому вигорання; підтримання стабільної продуктивності праці

4	Відхилення параметрів мікроклімату	Система кондиціонування; підтримання температури 21–23°С, вологості 40–60% відповідно до ДСН 3.3.6.042-99 [26]	Нормативні мікрокліматичні умови; підвищення комфортності та продуктивності праці
5	Військова загроза	Визначити та позначити найближче укриття; інструктаж щодо дій при тривозі; аптечка; ДБЖ для збереження даних [24]	Мінімізація загибелі та травмування персоналу; збереження цілісності даних системи

Висновки до розділу

У розділі розглянуто питання охорони праці стосовно робочого місця оператора інформаційної системи електронної комерції з функціями складського обліку.

У першому підрозділі охарактеризовано організаційно-правову базу охорони праці: Закон України “Про охорону праці” [23], принципи державної політики у цій сфері, санітарні норми ДСанПіН 3.3.6.096-2002 [25], ДСН 3.3.6.042-99 [26], ДСН 3.3.6.037-99 [27], ДБН В.2.5-28:2018 [18], а також міжнародні стандарти ДСТУ ISO 45001:2019 [28], Директиву ЄС 89/391 [29] та Конвенцію МОП № 187 [30].

У другому підрозділі описано умови праці оператора та характеристику робочого місця. Виявлено 9 потенційних небезпек: фізичні фактори (ЕМВ, недостатня освітленість, відхилення мікроклімату, шум, електронебезпека), психофізіологічні перевантаження (нервово-психічні та статичні), а також небезпеки загального характеру (пожежа, військова загроза).

У третьому підрозділі застосовано два методи оцінювання ризиків: аналізування “дерева відмов” (для небезпеки пожежі) та матрицю оцінювання ризиків (для п’яти ключових небезпек). За результатами оцінювання: один ризик – неприпустимий (військова загроза – 1С), два – небажані (пожежа – 1D; ЕМВ та психологічне перевантаження – 3В), один – прийнятний із перевіркою (мікроклімат – 4В). Для кожної небезпеки розроблено конкретні заходи, реалізація яких підвищить рівень безпеки праці на об’єкті проектування.

ЗАГАЛЬНІ ВИСНОВКИ

У рамках кваліфікаційної роботи поставлено й досягнуто ключове завдання: реалізовано веб-додаток, який забезпечує цілісну взаємодію інтернет-магазину та системи складського обліку. Результати проведеної роботи представлено нижче.

1. Дослідження специфіки галузі виявило, що головною проблемою для малих інтернет-підприємств є розбіжність між торговою платформою та реальним станом запасів. Аналіз чотирьох поширених рішень (WooCommerce, OpenCart, Magento, Хорошоп) показав, що жодне з них у базовій конфігурації не дає змоги повністю закрити цю потребу. Це стало підставою для розробки індивідуального рішення. Сформульовано 10 вимог до функціональних аспектів та 5 — до нефункціональних.

2. Розроблено схему реляційної бази даних, що включає 10 таблиць, з визначеними зв'язками між ними, а також первинними та зовнішніми ключами з правилами RESTRICT/CASCADE. Рішення щодо зберігання інформації про складські позиції (stock_items) в окремій таблиці від даних про товари (products) дозволило спростити атомарне блокування залишків. Крім того, створено ER-діаграму та діаграму класів, яка об'єднує чотири окремі блоки.

3. Було формалізовано математичний апарат (формули 2.1–2.5) для розрахунку вартості замовлення, величини знижки та коефіцієнта оборотності запасів. Алгоритм обробки замовлення базується на транзакції з опцією SELECT FOR UPDATE, що унеможливлює продаж товару, якого немає в наявності («в мінус»), навіть у випадку одночасних замовлень.

4. Програмний продукт було реалізовано з використанням сучасного стеку технологій: Python 3.11, Django 4.2, DRF 3.14, PostgreSQL 15, React 18, Redis 7 та Celery 5. REST API складається з 15 ендпоїнтів, захищених за допомогою JWT та рольового контролю доступу. Для фронтенду використано React SPA з адаптивним дизайном та зберіганням access-токену в оперативній пам'яті.

5. Процес тестування охопив три етапи: модульний (з покриттям 78% коду), інтеграційний (включно зі сценарієм одночасної купівлі двома клієнтами, коли залишається одна одиниця товару) та функціональний (за 12 граничними випадками). Всі перевірки пройшли успішно. Система коректно видає помилку 400 (Bad Request) при виявленні недостатньої кількості товару на складі.

6. Для гарантування безпечних умов праці визначені чіткі кількісні показники: освітленість — 300–500 лк; показники мікроклімату — температура в межах 21–24 °C за вологості 40–60%; рівень електромагнітного випромінювання (напруженість електричного поля) — не перевищувати 25 В/м. Додатково передбачено наявність вуглекислотного вогнегасника та системи автоматичної пожежної сигналізації.

Продукт підготовлено для етапу дослідної експлуатації. Плани подальшого розвитку передбачають: впровадження модуля онлайн-платежів (LiqPay), налагодження взаємодії з API поштового оператора "Нова Пошта", а також створення мобільного застосунку на базі React Native, що буде спілкуватися з тим самим REST API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laudon K. C., Traver C. G. E-Commerce 2024: Business, Technology, Society. 19th ed. – Pearson, 2023. – 912 с.

2. Statista Research Department. E-commerce worldwide – Statistics & Facts [Електронний ресурс]. – Режим доступу: https://www.statista.com/topics/871/online-shopping/ (дата звернення: 01.11.2024).

3. Краус Н. М., Голобородько О. П., Краус К. М. Цифрова економіка: тренди та перспективи авангардного характеру розвитку // Ефективна економіка. – 2023. – № 1. – URL: http://www.economy.nayka.com.ua/?op=1&z=6977.

4. Закон України «Про електронну комерцію» від 03.09.2015 № 675-VIII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/675-19 (дата звернення: 01.11.2024).

5. WooCommerce Documentation [Електронний ресурс]. – Режим доступу: https://woo.com/documentation/ (дата звернення: 03.11.2024).

6. OpenCart Documentation [Електронний ресурс]. – Режим доступу: https://docs.opencart.com/ (дата звернення: 03.11.2024).

7. Adobe Commerce (Magento) Developer Documentation [Електронний ресурс]. – Режим доступу: https://developer.adobe.com/commerce/docs/ (дата звернення: 04.11.2024).

8. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. – O'Reilly Media, 2022. – 594 с. – ISBN 978-1-491-92312-4.

9. Рудько Г. І., Артеменко В. М. Проектування баз даних та інформаційних систем: навч. посіб. – Київ : Академперіодика, 2023. – 368 с.

10. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. – MIT Press, 2022. – 1312 с.

11. Matthes E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. 3rd ed. – No Starch Press, 2023. – 552 с.

12. Django Documentation 4.2 [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/4.2/ (дата звернення: 05.11.2024).

13. Django REST Framework Documentation [Електронний ресурс]. – Режим доступу: https://www.django-rest-framework.org/ (дата звернення: 05.11.2024).

14. Булаєнко М. В., Назаренко Д. А. Реінжиніринг бізнес-процесів управління продажами для підприємства малого бізнесу / Матеріали VI Міжнародної науково-практичної конференції «Перспективи розвитку територій: теорія і практика» 16–17 листопада 2022 р. – Харків: ХНУМГ ім. О.М. Бекетова, 2022. – URL: https://science.kname.edu.ua/images/dok/konferentsii/2022/Tezy_2022/2022_molodi_vcheni.pdf

15. Булаєнко М. В., Трифонов О. В. Стратегії покращення якості програмного забезпечення / Тези I (VII) Міжнародної конференції «Інформаційні технології: теорія і практика» (Дніпро, 20–22 березня 2024). – Дніпро: Свідлер А.Л., 2024. – 479 с.

16. Булаєнко М. В., Локойда А. В. Забезпечення безпеки у веб-застосунку: методи шифрування та захисту даних / Матеріали VII Всеукраїнської науково-практичної інтернет-конференції «Сучасні інформаційні системи та технології», Херсон, 29 листопада 2024 р.

17. Булаєнко М. В., Конькова А. Р. Застосування методів стрес-тестування для оцінки надійності програмного забезпечення / Збірник матеріалів III Міжнар. конференції «Вища технічна освіта XXI століття», Краматорськ–Івано-Франківськ, 13–14 грудня 2024 р. – ДонНАБА, 2024. – 318 с.

18. Булаєнко М. В., Ликова В. І. Аналіз та покращення UX/UI як складової якості програмного забезпечення / Матеріали II (VIII) Міжнар. Інтернет-конференції здобувачів вищої освіти, Запоріжжя, 2–4 квітня 2025 р.

19. Булаєнко М. В. Кібербезпека в умовах цифрової трансформації: виклики для міжнародного бізнесу та державних послуг / XXXVIII Міжнар. науково-практична конференція «Трансформація економічних систем та інститутів у нових геостратегічних реаліях», 14–15 квітня 2025 р. – Дніпро: Університет ім. Альфреда Нобеля, 2025. – Том 2. – С. 65.

20. PostgreSQL Documentation 15 [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/15/ (дата звернення: 06.11.2024).

21. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/ (дата звернення: 07.11.2024).

22. Гігієна праці та виробнича санітарія : навч. посіб. / за ред. А. В. Халімовського. – Київ : Медицина, 2022. – 320 с. – ISBN 978-617-505-873-4.

23. ДСТУ EN ISO 9241-110:2022. Ергономіка взаємодії людини з системою. Принципи діалогу (EN ISO 9241-110:2020, IDT). – [Чинний від 2022-12-30]. – Київ : ДП «УкрНДНЦ», 2022. – 26 с.

24. ДБН В.2.5-28:2018. Природне і штучне освітлення. Зміна № 1 від 2022-12-01. – Київ : Мінрегіон України, 2022. – 140 с.

25. Наказ МОЗ України «Про затвердження Санітарного регламенту для офісних приміщень» від 08.04.2022 № 614 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0506-22.

26. Фаулер М. Шаблони архітектури корпоративних програм / пер. з англ. – Київ : Інформаційні системи, 2022. – 544 с. – ISBN 978-617-522-098-1.

27. Малишева В.В. Методичні вказівки до виконання розділу «Охорона праці» в дипломних роботах бакалаврів / В.В. Малишева. – Харків : ХНУМГ ім. О.М. Бекетова, 2025. – 14 с.

28. Третьяков О. В., Зацарний В. В., Безсонний В. Л. Охорона праці: навч. посіб. з тестовим контролем знань / за ред. О. В. Третьякова. – 2-ге вид. – Київ : Знання, 2022. – 280 с.

29. Закон України «Про охорону праці» від 14.10.1992 № 2694-XII [Електронний ресурс] : ред. від 19.07.2022 / Офіційний сайт Верховної Ради України. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12.

30. Кодекс цивільного захисту України [Електронний ресурс] / Офіційний сайт Верховної Ради України. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z1494-18.

31. Наказ МОЗ України «Про встановлення нормативів гранично допустимих рівнів електромагнітних полів» від 01.08.2023 № 1388 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z1299-23.

32. Наказ МОЗ України «Про затвердження Державних санітарних норм і правил «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища»» від 08.04.2022 № 613 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0505-22.

33. ДСТУ ГОСТ 12.1.003:2022. Система стандартів безпеки праці. Шум. Загальні вимоги безпеки. – [Чинний від 2023-01-01]. – Київ : ДП «УкрНДНЦ», 2022. – 18 с.

34. ДСТУ ISO 45001:2019. Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування. – [Зі змінами станом на 2023-01-01]. – Київ : ДП «УкрНДНЦ», 2023. – 25 с.

35. Директива Ради Європейських Співтовариств 89/391/ЕЕС «Про впровадження заходів, що сприяють поліпшенню безпеки й гігієни праці» [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/994_b23.

36. Конвенція МОП 187 «Про основи, що сприяють безпеці й гігієні праці» [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/993_515.

Додаток А

Фрагменти програмного коду системи

А.1 Модель StockItem та метод adjust (Python/Django)

```
# apps/warehouse/models.py
from django.db import models, transaction
from apps.users.models import User
from apps.catalog.models import Product

class StockItem(models.Model):
    product = models.OneToOneField(Product, on_delete=models.CASCADE,
                                   related_name='stock_item')
    quantity = models.PositiveIntegerField(default=0)
    min_quantity = models.PositiveIntegerField(default=5)
    updated_at = models.DateTimeField(auto_now=True)

    def adjust(self, quantity_change: int, movement_type: str,
               created_by: 'User', reference_id: int = None):
        with transaction.atomic():
            item = StockItem.objects.select_for_update().get(pk=self.pk)
            new_qty = item.quantity + quantity_change
            if new_qty < 0:
                raise ValueError(f'Insufficient stock for {self.product}')
            item.quantity = new_qty
            item.save()
            StockMovement.objects.create(
                product=self.product,
                movement_type=movement_type,
                quantity_change=quantity_change,
                reference_id=reference_id,
                created_by=created_by
            )
            if new_qty <= self.min_quantity:
                Notification.create_low_stock_alert(self.product)
        return item.quantity
```

А.2 Endpoint оформлення замовлення (DRF ViewSet)

```
# apps/orders/views.py
from rest_framework import viewsets, status
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from django.db import transaction
from .models import Order, OrderItem, Cart, CartItem
from .serializers import OrderSerializer, OrderCreateSerializer

class OrderViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]

    def create(self, request, *args, **kwargs):
        serializer = OrderCreateSerializer(data=request.data,
                                           context={'request': request})
        serializer.is_valid(raise_exception=True)
        try:
            with transaction.atomic():
                order = serializer.save(user=request.user)
        except ValueError as e:
            return Response({'detail': str(e)},
```

```
                            status=status.HTTP_400_BAD_REQUEST)
        return Response(OrderSerializer(order).data,
                        status=status.HTTP_201_CREATED)
```

А.3 Модель Order та метод confirm (Python/Django)

Метод confirm() є єдиною точкою входу для підтвердження замовлення. Він виконується в атомарній транзакції: спочатку блокує рядки залишків через SELECT FOR UPDATE, перевіряє наявність кожного товару, потім одночасно списує залишки, фіксує замовлення і генерує записи журналу руху. Будь-яка помилка відкочує всю транзакцію, що унеможливлює «продаж у мінус».

```
# apps/orders/models.py
from django.db import models, transaction
from apps.warehouse.models import StockItem

class Order(models.Model):
    STATUS_CHOICES = [
        ('pending',    'Очікує підтвердження'),
        ('confirmed',  'Підтверджено'),
        ('processing', 'В обробці'),
        ('shipped',    'Відправлено'),
        ('delivered',  'Доставлено'),
        ('cancelled',  'Скасовано'),
    ]
    user          = models.ForeignKey('users.User', on_delete=models.RESTRICT)
    status         = models.CharField(max_length=20, choices=STATUS_CHOICES,
                                      default='pending')
    total_amount   = models.DecimalField(max_digits=10, decimal_places=2)
    shipping_address = models.TextField()
    notes          = models.TextField(blank=True)
    created_at     = models.DateTimeField(auto_now_add=True)
    updated_at     = models.DateTimeField(auto_now=True)

    FORWARD_TRANSITIONS = {
        'pending':    {'confirmed', 'cancelled'},
        'confirmed':  {'processing', 'cancelled'},
        'processing': {'shipped'},
        'shipped':    {'delivered'},
    }

    def update_status(self, new_status: str) -> None:
        allowed = self.FORWARD_TRANSITIONS.get(self.status, set())
        if new_status not in allowed:
            raise ValueError(
                f'Перехід {self.status} -> {new_status} заборонено'
            )
        self.status = new_status
        self.save(update_fields=['status', 'updated_at'])

    @classmethod
    def confirm(cls, cart, shipping_address: str, notes: str = '') -> 'Order':
```

```
        """Атомарне підтвердження замовлення з SELECT FOR UPDATE."""
        with transaction.atomic():
            items = list(cart.cart_items.select_related('product').all())
            if not items:
                raise ValueError('Кошик порожній')

            # Блокуємо рядки залишків для всіх товарів у кошику
            product_ids = [i.product_id for i in items]
            stocks = {
                s.product_id: s
                for s in StockItem.objects.select_for_update()
                             .filter(product_id__in=product_ids)
            }

            # Перевіряємо наявність кожного товару
            for item in items:
                stock = stocks.get(item.product_id)
                if not stock or stock.quantity < item.quantity:
                    raise ValueError(
                        f'Недостатньо товару: {item.product.name}'
                    )

            # Розраховуємо суму замовлення
            total = sum(i.product.price * i.quantity for i in items)

            # Створюємо замовлення
            order = cls.objects.create(
                user=cart.user,
                status='confirmed',
                total_amount=total,
                shipping_address=shipping_address,
                notes=notes,
            )

            # Списуємо залишки і фіксуємо позиції замовлення
            for item in items:
                OrderItem.objects.create(
                    order=order,
                    product=item.product,
                    quantity=item.quantity,
                    unit_price=item.product.price,
                )
                stocks[item.product_id].adjust(
                    quantity_change=-item.quantity,
                    movement_type='sale',
                    created_by=cart.user,
                    reference_id=order.id,
                )

            cart.cart_items.all().delete()   # очищаємо кошик
            return order
```

А.4 Серіалізатори Django REST Framework

Серіалізатори виконують подвійну роль: перевірку вхідних даних (validation) і перетворення ORM-об'єктів у JSON для клієнта.

OrderCreateSerializer перевіряє адресу доставки і передає логіку підтвердження до методу Order.confirm(), не дублюючи бізнес-правила у view.

```
# apps/orders/serializers.py
from rest_framework import serializers
from .models import Order, OrderItem
from apps.catalog.serializers import ProductShortSerializer

class OrderItemSerializer(serializers.ModelSerializer):
    product = ProductShortSerializer(read_only=True)

    class Meta:
        model  = OrderItem
        fields = ['id', 'product', 'quantity', 'unit_price']



class OrderSerializer(serializers.ModelSerializer):
    items = OrderItemSerializer(many=True, read_only=True,
                                source='order_items')

    class Meta:
        model  = Order
        fields = ['id', 'status', 'total_amount',
                  'shipping_address', 'notes',
                  'created_at', 'items']



class OrderCreateSerializer(serializers.Serializer):
    shipping_address = serializers.CharField(max_length=500)
    notes            = serializers.CharField(required=False,
                                             allow_blank=True,
                                             default='')

    def validate_shipping_address(self, value):
        if len(value.strip()) < 10:
            raise serializers.ValidationError(
                'Адреса доставки занадто коротка.'
            )
        return value.strip()

    def save(self, user):
        cart = user.cart
        return Order.confirm(
            cart=cart,
            shipping_address=self.validated_data['shipping_address'],
            notes=self.validated_data.get('notes', ''),
        )
```

А.5 URL-маршрутизація та класи дозволів

Усі API-маршрути зібрані через DefaultRouter бібліотеки Django REST Framework. Кожен ViewSet реєструється одним рядком, що автоматично генерує ендпоїнти для list, create, retrieve, update та destroy. Класи дозволів

IsManager та IsAdminUser написані як субкласи BasePermission і перевіряють поле role у моделі User.

```
# apps/api/urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from apps.orders.views   import OrderViewSet
from apps.catalog.views  import ProductViewSet, CategoryViewSet
from apps.warehouse.views import StockItemViewSet, StockReceiptViewSet
from apps.users.views    import UserViewSet

router = DefaultRouter()
router.register(r'products',    ProductViewSet,      basename='product')
router.register(r'categories',  CategoryViewSet,     basename='category')
router.register(r'orders',      OrderViewSet,        basename='order')
router.register(r'warehouse/stock',    StockItemViewSet,   basename='stock')
router.register(r'warehouse/receipts', StockReceiptViewSet,basename='receipt')
router.register(r'users',       UserViewSet,         basename='user')

urlpatterns = [
    path('api/v1/', include(router.urls)),
    path('api/v1/auth/', include('apps.users.auth_urls')),
]

# apps/users/permissions.py
from rest_framework.permissions import BasePermission

class IsManager(BasePermission):
    """Дозволяє доступ лише менеджерам та адміністраторам."""
    def has_permission(self, request, view):
        return bool(
            request.user and
            request.user.is_authenticated and
            request.user.role in ('manager', 'admin')
        )

class IsOwnerOrManager(BasePermission):
    """Власник об'єкта або менеджер/адмін."""
    def has_object_permission(self, request, view, obj):
        if request.user.role in ('manager', 'admin'):
            return True
        return getattr(obj, 'user_id', None) == request.user.id
```

А.6 Автоматизовані тести (Pytest)

Для тестування використовується Pytest разом із pytest-django. Нижче наведено два ключові тестові сценарії: модульний тест методу adjust() та інтеграційний тест на конкурентне замовлення (два потоки намагаються купити останній товар одночасно).

```
# tests/test_stock.py
import pytest
```

```
import threading
from django.test import TestCase
from apps.warehouse.models import StockItem
from apps.catalog.tests.factories import ProductFactory
from apps.users.tests.factories  import UserFactory


class TestStockItemAdjust(TestCase):
    def setUp(self):
        self.product = ProductFactory()
        self.stock   = StockItem.objects.create(
            product=self.product, quantity=10, min_quantity=3
        )
        self.manager = UserFactory(role='manager')

    def test_adjust_decreases_quantity(self):
        self.stock.adjust(-3, 'sale', self.manager)
        self.stock.refresh_from_db()
        self.assertEqual(self.stock.quantity, 7)

    def test_adjust_raises_on_negative_stock(self):
        with self.assertRaises(ValueError):
            self.stock.adjust(-20, 'sale', self.manager)

    def test_adjust_creates_movement_record(self):
        from apps.warehouse.models import StockMovement
        self.stock.adjust(5, 'receipt', self.manager)
        self.assertEqual(StockMovement.objects.count(), 1)
        mv = StockMovement.objects.first()
        self.assertEqual(mv.quantity_change, 5)



class TestConcurrentOrder(TestCase):
    """Два покупці одночасно купують останній товар."""

    def setUp(self):
        self.product = ProductFactory(price='100.00')
        self.stock   = StockItem.objects.create(
            product=self.product, quantity=1
        )
        self.buyer1 = UserFactory(role='customer')
        self.buyer2 = UserFactory(role='customer')

    def _place_order(self, user, results, idx):
        try:
            from apps.orders.models import Cart, CartItem, Order
            cart = Cart.objects.create(user=user)
            CartItem.objects.create(
                cart=cart, product=self.product, quantity=1
            )
            Order.confirm(cart=cart, shipping_address='вул. Тестова, 1')
            results[idx] = 'ok'
        except ValueError:
            results[idx] = 'error'

    def test_only_one_order_succeeds(self):
        results = [None, None]
        t1 = threading.Thread(target=self._place_order,
                              args=(self.buyer1, results, 0))
        t2 = threading.Thread(target=self._place_order,
                              args=(self.buyer2, results, 1))
```

```
        t1.start(); t2.start()
        t1.join();  t2.join()
        ok_count  = results.count('ok')
        err_count = results.count('error')
        self.assertEqual(ok_count,  1)
        self.assertEqual(err_count, 1)
        # Залишок не пішов у мінус
        self.stock.refresh_from_db()
        self.assertGreaterEqual(self.stock.quantity, 0)
```

А.7 Конфігурація Docker Compose

Нижче наведено файл docker-compose.yml, що описує весь стек системи: Django-сервер (Gunicorn), PostgreSQL 15, Redis 7 (черга Celery та кеш) і Nginx як зворотний проксі. Завдяки цій конфігурації розгортання системи на будь-якому сервері зводиться до однієї команди docker compose up -d.

```
# docker-compose.yml
version: '3.9'

services:
  db:
    image: postgres:15-alpine
    restart: always
    environment:
      POSTGRES_DB:       ecommerce
      POSTGRES_USER:     ecom_user
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ['CMD-SHELL', 'pg_isready -U ecom_user -d ecommerce']
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    restart: always
    volumes:
      - redis_data:/data

  web:
    build: .
    restart: always
    command: >
      gunicorn config.wsgi:application
      --bind 0.0.0.0:8000
      --workers 4
      --timeout 60
    env_file: .env
    depends_on:
      db:
        condition: service_healthy
      redis:
```

```
        condition: service_started
    volumes:
      - static_files:/app/staticfiles
      - media_files:/app/media

  celery:
    build: .
    restart: always
    command: celery -A config worker -l info -c 2
    env_file: .env
    depends_on: [web, redis]

  nginx:
    image: nginx:1.25-alpine
    restart: always
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d:ro
      - static_files:/static:ro
      - media_files:/media:ro
      - ./ssl:/etc/ssl:ro
    depends_on: [web]

volumes:
  postgres_data:
  redis_data:
  static_files:
  media_files:
```

А.8 Компонент каталогу товарів (React)

Нижче наведено ключовий React-компонент сторінки каталогу. Він використовує хук useEffect для завантаження товарів при зміні фільтрів, debounce для оптимізації пошукових запитів та обробляє стани завантаження і помилок.

```
// src/pages/CatalogPage.jsx
import React, { useState, useEffect, useCallback } from 'react';
import { api } from '../services/api';
import ProductCard from '../components/ProductCard';
import Pagination from '../components/Pagination';
import { debounce } from 'lodash';

export default function CatalogPage() {
  const [products, setProducts]   = useState([]);
  const [loading,  setLoading]    = useState(false);
  const [error,    setError]      = useState(null);
  const [search,   setSearch]     = useState('');
  const [category, setCategory]   = useState('');
  const [page,     setPage]       = useState(1);
  const [total,    setTotal]      = useState(0);
  const PAGE_SIZE = 12;
```

```
  const fetchProducts = useCallback(
    debounce(async (q, cat, pg) => {
      setLoading(true);
      setError(null);
      try {
        const params = { page: pg, page_size: PAGE_SIZE };
        if (q)   params.search   = q;
        if (cat) params.category = cat;
        const { data } = await api.get('/products/', { params });
        setProducts(data.results);
        setTotal(data.count);
      } catch (err) {
        setError('Не вдалося завантажити товари. Спробуйте пізніше.');
      } finally {
        setLoading(false);
      }
    }, 300),
    []
  );

  useEffect(() => {
    fetchProducts(search, category, page);
  }, [search, category, page, fetchProducts]);

  return (
    <div className='catalog-page'>
      <div className='filters'>
        <input
          type='text'
          placeholder='Пошук товарів...'
          value={search}
          onChange={e => { setSearch(e.target.value); setPage(1); }}
        />
        <select
          value={category}
          onChange={e => { setCategory(e.target.value); setPage(1); }}
        >
          <option value=''>Всі категорії</option>
        </select>
      </div>

      {loading && <div className='loader'>Завантаження...</div>}
      {error   && <div className='error'>{error}</div>}

      <div className='product-grid'>
        {products.map(p => <ProductCard key={p.id} product={p} />)}
      </div>

      <Pagination
        page={page}
        total={total}
        pageSize={PAGE_SIZE}
        onChange={setPage}
      />
    </div>
  );
}
```