

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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система менеджменту товарів для оптимізації
мережі магазинів. Спец-частина: розробка Frontend-частини
системи»

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

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

_____ Богдан ГРИДЬКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

_____ Богдан ГРИДЬКО

Керівник: _____ Владислав ЯСКЕВИЧ
канд.техн.наук, доц.

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

_____ Гридку Богдану Андрійовичу _____

1. Тема кваліфікаційної роботи: «Система менеджменту товарів для оптимізації мережі магазинів. Спец-частина: розробка Frontend-частини системи»
керівник кваліфікаційної роботи канд.техн.наук, доц. Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

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

3. Вихідні дані до кваліфікаційної роботи: вихідні дані до кваліфікаційної роботи: теоретичні відомості про управління товарними запасами у мережі магазинів, опис існуючих програмних засобів для обліку товарів і залишків, технічна документація до React, TypeScript, Vite, Chakra UI, React Router, Axios, TanStack Query, Jest та React Testing Library.

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

1. Провести аналіз предметної галузі, існуючих рішень, актуальності розробки, об'єкта, предмета, мети та задач роботи.

2. Спроектувати Frontend-частину системи LvStore: вимоги, архітектуру, рольову модель, сценарії користувачів та інтерфейс.

3. Реалізувати Frontend-частину LvStore, описати структуру проєкту, маршрутизацію, взаємодію з API та рольові кабінети.
4. Провести тестування системи: тест-кейси, кросбраузерність, адаптивність, функціональні сценарії, форми та role-based доступ.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Карта вебзастосунку
5. Діаграма варіантів використання.
6. Діаграма компонентів.
7. Результати тестування.
8. Екранні форми.
9. Апробація результатів дослідження

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Формування вимог до Frontend-частини системи LvStore	01.03-07.03.2026	
4	Проектування архітектури, карти вебзастосунку та рольових сценаріїв	15.03-29.03.2026	
5	Програмна реалізація застосунку	30.03-24.04.2026	
6	Тестування застосунку	25.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-11.05.2026	
8	Розробка демонстраційних матеріалів	12.05-18.05.2026	
9	Попередній захист роботи	19.05-29.05.2026	

Здобувач вищої освіти

(підпис)

Богдан ГРИДЬКО

Керівник

кваліфікаційної роботи

(підпис)

Владислав ЯСКЕВИЧ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 3 табл., 45 рис., 27 джерел.

Мета роботи - спрощення управління товарними залишками й обробки замовлень у роздрібній мережі за рахунок клієнтської частини веб-орієнтованої інформаційної системи LvStore.

Об'єкт дослідження - процес розробки веб-орієнтованого програмного забезпечення для обробки даних про товари, товарні залишки та замовлення в інформаційній системі роздрібної мережі.

Предмет дослідження - методи, засоби та технології побудови клієнтської частини веб-орієнтованої інформаційної системи управління товарними залишками та замовленнями в роздрібній мережі.

Короткий зміст роботи: В роботі розглянуто предметну галузь управління товарними залишками, проаналізовано існуючі програмні засоби Торгсофт, WooCommerce та Loyverse POS, сформовано вимоги до Frontend-частини системи LvStore. Виконано проєктування клієнтської частини вебзастосунку, описано карту вебзастосунку, діаграму варіантів використання, діаграму компонентів і підхід до побудови рольових кабінетів. Обґрунтовано вибір технологій React, TypeScript, Vite, Chakra UI, React Router, Axios, TanStack Query, Jest і React Testing Library. Реалізовано основні сторінки для адміністратора магазину, продавця та менеджера складу, організовано взаємодію з REST API, проведено автоматизоване і ручне тестування Frontend-частини.

Сферою використання застосунку є роздрібні мережі магазинів, у яких потрібно координувати облік товарів, контроль залишків, створення замовлень.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, FRONTEND, LVSTORE, REACT, TYPESCRIPT, VITE, CHAKRA UI, ROLE-BASED ACCESS, REST API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	9
ВСТУП	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ	12
1.1 Вебзастосунок для управління товарними залишками у мережі магазинів	12
1.2 Проблеми управління товарними залишками у мережі магазинів	14
1.3 Аналіз аналогів	15
1.3.1 Торгсофт.....	16
1.3.2 WooCommerce	18
1.3.3 Loyverse POS	20
1.3.4 Таблиця порівнянь аналогів.....	22
1.4 Огляд та обґрунтування ІТ-засобів для реалізації вебзастосунку	23
1.4.1 React.....	23
1.4.2 TypeScript.....	24
1.4.3 Vite	25
1.4.4 Chakra UI.....	25
1.4.5 React Router.....	26
1.4.6 Axios	26
1.4.7 TanStack Query.....	27
1.4.8 Jest і React Testing Library	27
2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ	28
2.1 Вимоги до програмного забезпечення.....	28
2.1.1 Функціональні вимоги.....	28
2.1.2 Нефункціональні вимоги.....	29
2.2 Ролі користувачів і сценарії роботи	30
2.3 Проєктування варіантів використання вебзастосунку	31

2.4	Проектування клієнтської архітектури вебзастосунку	33
2.5	Проектування компонентної структури Frontend-частини.....	34
2.6	Проектування навігаційної карти вебзастосунку	37
2.7	Проектування інтерфейсу користувача	38
3	ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ.....	43
3.1	Архітектура та структура вебзастосунку.....	43
3.1.1	Файлова структура вебзастосунку	43
3.1.2	Організація маршрутизації та role-based доступу.....	45
3.1.3	Організація взаємодії з API.....	48
3.2	Реалізація користувацького інтерфейсу вебзастосунку.....	49
3.2.1	Реалізація головної сторінки вебзастосунку	49
3.2.2	Реалізація сторінки авторизації користувача	53
3.2.3	Реалізація кабінету адміністратора магазину	57
3.2.4	Реалізація кабінету продавця	61
3.2.5	Реалізація кабінету менеджера складу.....	63
4	ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ВЕБЗАСТОСУНКУ	65
4.1	Кросбраузерне тестування	65
4.2	Тестування адаптивності під різні пристрої	67
4.3	Функціональне тестування.....	69
4.4	Нефункціональне тестування продуктивності.....	75
	ВИСНОВКИ.....	81
	ПЕРЕЛІК ПОСИЛАНЬ	84
	ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	86
	ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ	92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

DOM - Document Object Model

FCP - First Contentful Paint

HTTP - HyperText Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

LCP - Largest Contentful Paint

NOK - Not OK

OAuth - Open Authorization

OK - Correct test result

REST - Representational State Transfer

UI - User Interface

URL - Uniform Resource Locator

UX - User Experience

ВСТУП

Актуальність: роботи пов'язана з тим, що роздрібні мережі працюють із великою кількістю товарів, магазинів, складів і внутрішніх замовлень. Навіть за невеликої кількості торгових точок виникає потреба швидко бачити фактичні залишки, розуміти, які позиції наближаються до критичного рівня, і передавати замовлення на склад без дублювання інформації у таблицях або повідомленнях. Якщо ці процеси ведуться вручну, продавець, адміністратор і менеджер складу часто працюють із різними версіями даних. Це збільшує ризик помилок під час поповнення запасів і ускладнює контроль виконання замовлень.

Для вирішення цієї задачі розробляється система LvStore, у якій спеціальна частина кваліфікаційної роботи присвячена саме Frontend-частині. Клієнтська частина відповідає за відображення даних, організацію робочих кабінетів користувачів, role-based доступ, зручну навігацію та взаємодію з REST API. На відміну від загального інтернет-магазину, LvStore орієнтується на внутрішню роботу мережі магазинів: продавець створює замовлення на поповнення, адміністратор керує товарами і магазинами, а менеджер складу працює з відвантаженням і складськими залишками.

Мета роботи - спрощення управління товарними залишками й обробки замовлень у роздрібній мережі за рахунок клієнтської частини веб-орієнтованої інформаційної системи LvStore.

Об'єкт дослідження - процес розробки веб-орієнтованого програмного забезпечення для обробки даних про товари, товарні залишки та замовлення в інформаційній системі роздрібною мережі.

Предмет дослідження - методи, засоби та технології побудови клієнтської частини веб-орієнтованої інформаційної системи управління товарними залишками та замовленнями в роздрібній мережі.

Для реалізації поставленої мети визначено такі задачі:

1. Провести аналіз предметної галузі управління товарними залишками у мережі магазинів, визначити основні проблеми внутрішніх замовлень, контролю залишків і рольової взаємодії користувачів.

2. Проаналізувати існуючі програмні рішення для обліку товарів і залишків, визначити їхні переваги та недоліки, скласти порівняльну характеристику аналогів і обґрунтувати доцільність розробки Frontend-частини системи LvStore.

3. Сформулювати функціональні та нефункціональні вимоги до Frontend-частини вебзастосунку з урахуванням ролей адміністратора магазину, продавця та менеджера складу.

4. Спроекувати клієнтську архітектуру вебзастосунку, рольову модель доступу, карту вебзастосунку, діаграму варіантів використання, діаграму компонентів та інтерфейс користувача відповідно до основних сценаріїв роботи.

5. Обґрунтувати вибір технологічних засобів розробки та реалізувати Frontend-частину системи LvStore, зокрема сторінку авторизації, головну сторінку, рольові кабінети користувачів, маршрутизацію, захищені маршрути та взаємодію з REST API.

6. Провести тестування Frontend-частини вебзастосунку

Наукова новизна роботи полягає в уточненні підходу до побудови рольово-орієнтованої клієнтської частини для системи управління товарними запасами, у якій маршрути, оболонки інтерфейсу та набір сторінок визначаються роллю автентифікованого користувача.

Практична значущість роботи полягає в розробленні Frontend-частини, яку можна використовувати як основу для подальшого впровадження системи управління товарами, залишками та внутрішніми замовленнями в торговельній мережі. Запропоноване рішення може бути використане для побудови рольових кабінетів користувачів, організації доступу до сторінок залежно від ролі, відображення товарів, залишків і замовлень, а також для подальшого розвитку системи у взаємодії із серверною частиною.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Вебзастосунок для управління товарними залишками у мережі магазинів

Цифровізація торговельних процесів поступово змінює підхід до управління товарами, залишками та внутрішніми замовленнями. Для мережі магазинів уже недостатньо вести облік у таблицях, окремих файлах або паперових журналах, оскільки такі способи погано працюють при збільшенні кількості торгових точок, складів і працівників. Чим більше товарних позицій бере участь у щоденній роботі, тим складніше підтримувати актуальність даних без автоматизованої системи. Помилки в залишках, несвоєчасне поповнення запасів або відсутність єдиного джерела інформації напряду впливають на якість обслуговування покупців і стабільність роботи магазину [1-3].

Традиційний підхід до управління товарними залишками часто передбачає, що інформація про товари, склади, магазини і замовлення зберігається в різних місцях. Частина даних може знаходитись у локальних документах, частина в обліковій програмі, частина передаватись між працівниками вручну. У такій ситуації складно швидко визначити, скільки товару залишилось у конкретному магазині, чи потрібно формувати замовлення на поповнення, хто має підтвердити операцію і на якому етапі перебуває виконання. Для невеликого магазину такий підхід ще може бути прийнятним, але для мережі він створює затримки і підвищує ризик помилок.

Автоматизація управління залишками дає змогу звести основні операції в єдине інформаційне середовище. У межах такого середовища працівники можуть переглядати товари, контролювати залишки, створювати замовлення, змінювати їх статуси та працювати відповідно до своєї ролі. Це особливо важливо для системи, де адміністратор, продавець і менеджер складу виконують різні завдання. Продавець повинен бачити залишки свого магазину і створювати замовлення на

поповнення. Менеджер складу має працювати із замовленнями, складами та складськими залишками. Адміністратор відповідає за загальне керування товарами, магазинами, користувачами і контроль операцій у системі.

Потреба у вебзастосунку пояснюється тим, що така форма реалізації не прив'язує користувача до одного робочого місця. Доступ до системи може здійснюватися через браузер, а інтерфейс може бути однаковим для різних пристроїв. Це спрощує використання системи в магазині, на складі або в адміністративному кабінеті. Окремою перевагою є те, що frontend-частина може бути відокремлена від backend-частини, а обмін даними між ними відбувається через API. Завдяки цьому інтерфейс відповідає за зручність роботи користувача, а серверна частина виконує зберігання, обробку та перевірку даних [9, 24-26].

Вебзастосунок для управління товарними залишками у мережі магазинів повинен не просто відображати список товарів. Він має підтримувати реальні сценарії роботи працівників: авторизацію, розмежування доступу, перегляд залишків, створення замовлень, підтвердження або скасування операцій, керування складами і магазинами. У проєкті LvStore така система розглядається як інструмент для впорядкування внутрішніх процесів мережі магазинів. Основна увага приділяється не продажам як окремій операції, а саме контролю товарних запасів, узгодженню дій між ролями та зменшенню ручної роботи під час обробки внутрішніх замовлень.

Таким чином, розробка зручного вебзастосунку для управління товарними залишками є обґрунтованою через зростання вимог до швидкості, точності та прозорості внутрішніх процесів. Для мережі магазинів важливо мати систему, яка дозволяє не тільки зберігати дані про товари, а й організовувати роботу різних ролей у межах спільного інформаційного простору.

1.2 Проблеми управління товарними залишками у мережі магазинів

Управління товарними залишками у мережі магазинів є складнішим, ніж облік товарів в одній торговій точці. У такій системі потрібно враховувати не лише кількість товару, а й місце його зберігання, відповідального працівника, потребу в поповненні, статус внутрішнього замовлення і доступність інформації для різних ролей. Якщо ці процеси не автоматизовані, управління запасами стає залежним від ручного введення даних і постійного уточнення інформації між працівниками.

Однією з основних проблем є неактуальність залишків. У разі несвоєчасного оновлення даних продавець може бачити неправильну кількість товару, менеджер складу може сформулювати неповне відвантаження, а адміністратор не матиме достовірної картини щодо стану запасів у мережі. Такі помилки виникають тоді, коли інформація про рух товарів не оновлюється одразу після виконання операції або коли дані зберігаються в кількох незалежних джерелах.

Другою проблемою є складність контролю внутрішніх замовлень. Для мережі магазинів важливо, щоб запит на поповнення проходив зрозумілий шлях: створення, перегляд, підтвердження, відвантаження, скасування або редагування. Якщо цей процес не має чіткої цифрової форми, частина замовлень може губитися, дублюватися або виконуватися із затримкою. У результаті одні магазини можуть мати надлишок товару, а інші не отримують потрібні позиції вчасно.

Третьою проблемою є відсутність чіткого розмежування прав доступу. У системі управління залишками не всі користувачі повинні мати однакові можливості. Продавцю не потрібен доступ до керування користувачами або всіма складами. Менеджер складу має працювати зі складськими залишками, замовленнями і відвантаженнями, але не обов'язково повинен змінювати загальні налаштування магазинів. Адміністратор, навпаки, потребує ширшого доступу для контролю системи. Якщо рольова модель не реалізована або реалізована слабо, зростає ризик випадкових змін, некоректного видалення даних і плутанини у відповідальності.

Окремо слід виділити проблему надлишкової ручної роботи. Працівники можуть витрачати час на пошук інформації, повторне введення однакових даних, уточнення залишків у чатах або таблицях. Такий підхід уповільнює роботу і не дає швидко отримати відповідь на прості питання: який товар є в наявності, у якому магазині він потрібен, яке замовлення вже підтверджено, а яке ще очікує обробки.

Ще одна проблема пов'язана з масштабуванням. Система, яка працює для одного магазину, не завжди підходить для мережі. Зі збільшенням кількості торгових точок зростає кількість товарних позицій, користувачів, складів і замовлень. Якщо програмне рішення не має продуманої структури, підтримка такої системи стає складною. Зміна одного процесу може впливати на інші частини, а додавання нової ролі або сторінки потребує значних переробок [8, 9, 24, 25].

Для вирішення зазначених проблем потрібен вебзастосунок, який поєднує рольову модель доступу, зручний інтерфейс, роботу з товарами, залишками і замовленнями, а також централізовану взаємодію з серверною частиною. У цьому контексті LvStore орієнтований на практичні потреби мережі магазинів, де важливо не тільки зберігати інформацію, а й зробити її доступною для кожної ролі в потрібному обсязі.

1.3 Аналіз аналогів

Аналіз аналогів дає змогу визначити, які підходи вже використовуються у сфері автоматизації торгівлі, складського обліку та управління товарними залишками. Для розробки вебзастосунку LvStore такий аналіз потрібний не лише для порівняння функцій, а й для виявлення обмежень наявних рішень. Особливу увагу слід приділити тому, наскільки системи підтримують внутрішню логістику, рольовий доступ, роботу із залишками, автоматизацію замовлень і можливість розгортання у контрольованому середовищі.

У межах роботи розглянуто три програмні рішення: Торгсофт, WooCommerce і Loyverse POS. Вони належать до різних типів систем [4-6].

Порівняння цих рішень дозволяє побачити, що кожне з них частково вирішує задачі, пов'язані з управлінням товарами. Водночас вони не повністю відповідають контексту LvStore, де основний акцент зроблено на внутрішніх замовленнях, рольовій взаємодії між адміністратором, продавцем і менеджером складу, а також на frontend-частині, яка повинна бути зручною для щоденної роботи в мережі магазинів.

1.3.1 Торгсофт

Торгсофт є програмним рішенням для автоматизації торгівлі, складського обліку, контролю товарів, продажів, фінансів і роботи з клієнтами. Система використовується для ведення обліку в магазинах, роботи зі штрихкодами, контролю залишків, формування документів і аналізу торговельної діяльності. За своїм призначенням Торгсофт ближчий до повноцінної облікової системи, ніж до окремого frontend-застосунку для рольового управління внутрішніми процесами [4].

Екранну форму системи Торгсофт наведено на рисунку 1.1.

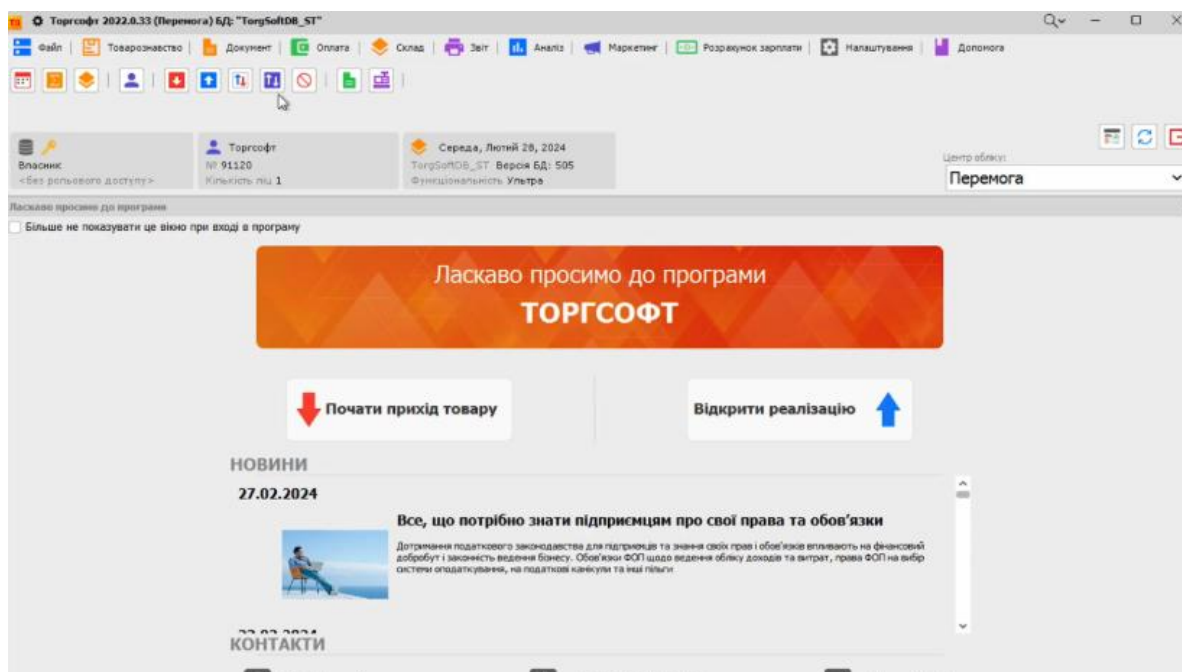


Рис. 1.1 Екранна форма системи Торгсофт

До переваг Торгсофт можна віднести широкий набір функцій для торгівлі та складського обліку. Система підтримує роботу з товарами, залишками, продажами, складом, фінансами, клієнтами та торговим обладнанням. Для підприємств, які потребують комплексного обліку, це є суттєвою перевагою, оскільки значна частина операцій зосереджена в одному програмному середовищі.

Ще однією перевагою є орієнтація на реальні процеси торгівлі. Система може бути корисною для магазинів, які вже мають усталені процеси продажів і складського обліку. Наявність інструментів для роботи із залишками дає можливість контролювати кількість товарів і зменшувати ризик втрати інформації про рух номенклатури.

Разом з цим Торгсофт має певні недоліки в контексті розроблюваного застосунку. Насамперед система є ширшою за функціональністю, ніж потрібно для задачі frontend-частини LvStore. Вона охоплює багато напрямів обліку, тому інтерфейс може бути складнішим для користувача, якому потрібно виконувати лише конкретну рольову дію: створити замовлення, переглянути залишки магазину або обробити складське відвантаження.

Також рішення не є вузько орієнтованим саме на досліджувану модель внутрішніх замовлень між магазином і складом. У ньому є засоби складського обліку, але логіка, де продавець, менеджер складу й адміністратор працюють через окремі кабінети з чітко розділеними функціями, не є основною ідеєю системи. Для LvStore така рольова структура є центральною, тому готове рішення не повністю відповідає поставленій задачі.

Таким чином, Торгсофт можна розглядати як сильний аналог у сфері торгового та складського обліку. Однак для задачі розробки спеціалізованої frontend-частини з окремими кабінетами користувачів, внутрішніми замовленнями і спрощеним web-інтерфейсом він є надмірно широким і менш гнучким для досліджуваної моделі.

1.3.2 WooCommerce

WooCommerce є платформою для створення інтернет-магазинів на базі WordPress. Основне призначення цієї системи полягає в організації онлайн-продажів, керуванні товарами, замовленнями, оплатами, доставкою та клієнтською частиною електронної комерції. Базова система може бути розширена додатковими плагінами, зокрема для управління запасами, постачальниками, прогнозування попиту або автоматизації поповнення [5].

Екранну форму WooCommerce наведено на рисунку 1.2.

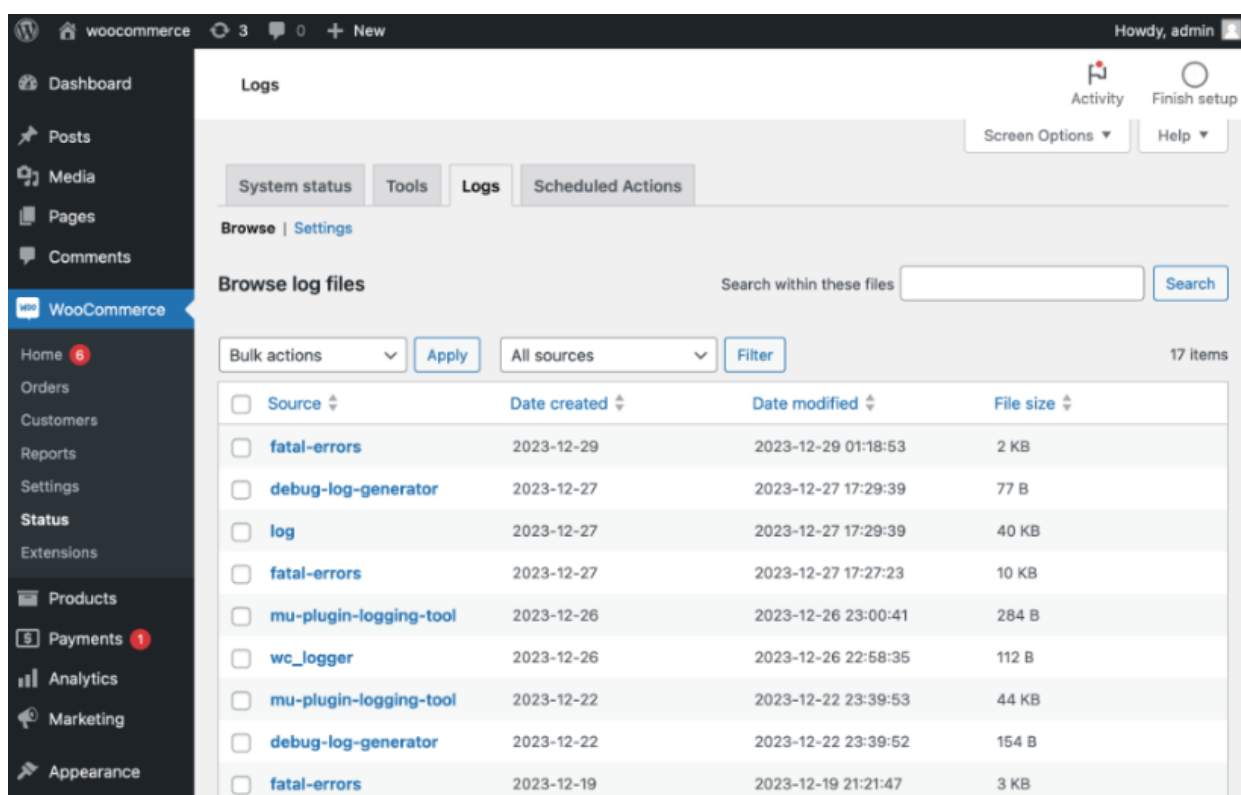


Рис. 1.2 Екранна форма системи WooCommerce

Основною перевагою WooCommerce є гнучкість. Платформа має велику кількість розширень, тому її можна адаптувати під різні сценарії електронної комерції. Для власника інтернет-магазину це означає можливість налаштувати каталог товарів, обробку замовлень, оплату, доставку, облік залишків та інші функції без розробки системи з нуля.

Ще однією перевагою є поширеність WordPress-екосистеми. Для WooCommerce існує багато готових тем, модулів і технічної документації. Це полегшує запуск магазину і дозволяє швидко отримати робочу систему для онлайн-продажів. У частині управління запасами платформа може бути доповнена спеціалізованими плагінами, які дозволяють працювати зі складськими залишками, постачальниками або повідомленнями про поповнення товару.

Однак у контексті LvStore платформа WooCommerce має суттєві обмеження. Передусім ця платформа орієнтована на електронну комерцію, а не на внутрішню логістику мережі магазинів. Її основна логіка побудована навколо продажу товарів клієнтам, а не навколо внутрішніх замовлень між продавцем, адміністратором і менеджером складу.

Також базова рольова модель WooCommerce не відповідає напряду обраній структурі LvStore. Для реалізації окремих кабінетів продавця, адміністратора магазину і менеджера складу потрібні додаткові налаштування або сторонні плагіни. Це ускладнює контроль доступу і може зробити систему залежною від якості сторонніх розширень.

Ще одним недоліком є залежність від WordPress-інфраструктури. Для окремого спеціалізованого застосунку, який має бути ізольованим і побудованим під конкретну бізнес-логіку, така залежність не завжди є зручною. У LvStore frontend-частина розробляється як окремий вебзастосунок, що взаємодіє із сервером через REST API, тоді як WooCommerce працює в межах іншої архітектурної моделі.

Отже, WooCommerce є зручним рішенням для інтернет-магазинів, але для задачі управління внутрішніми товарними залишками в мережі магазинів його можливості є частково придатними.

1.3.3 Loyverse POS

Loyverse POS є POS-системою, орієнтованою на продажі, облік товарів, контроль залишків, роботу з кількома магазинами та базову аналітику. Система дозволяє відстежувати залишки в реальному часі, отримувати повідомлення про низький рівень запасів, працювати із замовленнями постачальникам і переміщенням товарів між магазинами. Частина розширених функцій належить до модуля Advanced Inventory [6].

Екранну форму Loyverse POS наведено на рисунку 1.3.

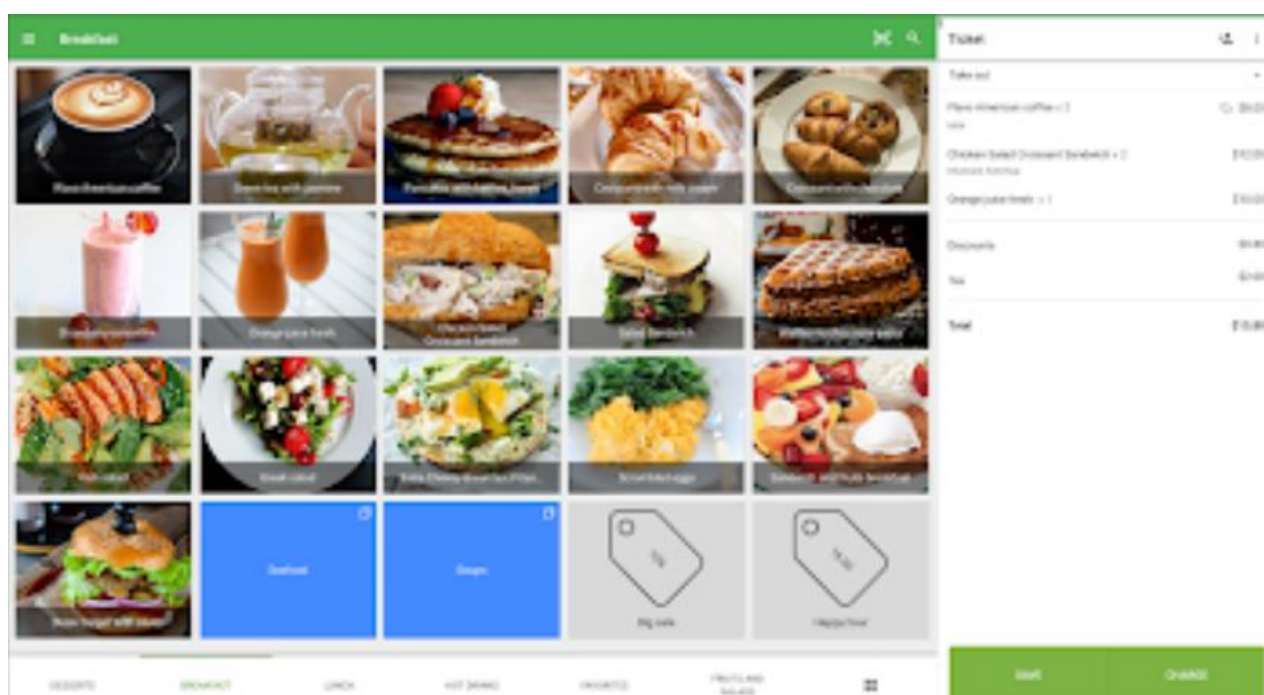


Рис. 1.3 Екранна форма системи Loyverse POS

До переваг Loyverse POS належить простота використання. Система орієнтована на щоденну роботу торгової точки, тому її інтерфейс є зрозумілим для продавців і працівників малого бізнесу. Наявність функцій контролю залишків, low stock alerts і переміщення товарів між магазинами робить її корисною для підприємств, які хочуть швидко впорядкувати базовий облік.

Ще однією перевагою є підтримка роботи з кількома торговими точками. Для мережі магазинів це важливо, оскільки товари можуть знаходитись у різних місцях, а керівник або відповідальний працівник повинен бачити загальну ситуацію із

запасами. Функції purchase orders і transfer orders частково закривають потребу в поповненні та переміщенні товарів.

Окремо слід зазначити, що Loyverse POS добре підходить для підприємств, у яких основна увага приділяється продажам і швидкому обслуговуванню покупців. Система надає продавцю зручні інструменти для роботи з товарами, чеками, залишками і базовою аналітикою. Проте така орієнтація означає, що більшість процесів побудовані навколо касової операції, а не навколо внутрішнього руху товарів між магазином і складом.

У контексті розроблюваної системи LvStore важливим є не тільки факт наявності товару, а й те, хто створює замовлення, хто його переглядає, хто підтверджує або виконує складську дію. У Loyverse POS такі процеси можуть бути частково представлені через інструменти замовлень і переміщень, але вони не формують окрему рольову модель взаємодії між продавцем, адміністратором магазину і менеджером складу.

Також потрібно врахувати, що Loyverse POS є готовим сервісним рішенням, у якому користувач працює в межах уже заданої логіки системи та готового інтерфейсу. Це зручно для швидкого запуску торгової точки, але обмежує можливості глибокої адаптації під власну клієнтську архітектуру. У випадку LvStore frontend-частина проєктується окремо, тому її структура може бути побудована відповідно до конкретних сценаріїв роботи: перегляду залишків магазину, створення внутрішнього замовлення, обробки заявки складом і контролю дій адміністратора. Крім того, у LvStore досліджується саме технічна побудова клієнтської частини: компонентна структура, role-based маршрутизація, захищені сторінки, API-шар і відображення даних відповідно до ролі користувача.

Таким чином, Loyverse POS можна вважати сильним аналогом для POS-обліку та контролю залишків. Проте для задачі розробки frontend-частини з чітким розподілом ролей, внутрішніми замовленнями і спеціалізованою логікою складської взаємодії його можливості є лише частково відповідними.

1.3.4 Таблиця порівнянь аналогів

Зведені результати аналізу характеристик розглянутих застосунків наведено у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих аналогів

Критерій	Торгсофт	WooCommerce	Loyverse POS
Спеціалізація на внутрішній логістиці	Частково	Частково	Частково
Автоматизація внутрішніх замовлень	Частково	Ні	Частково
Жорстка багаторівнева рольова модель	Так	Ні	Ні
Захист даних від випадкового видалення	Так	Частково	Так
Портативність та ізолюваність розгортання	Ні	Частково	Ні

Порівняння показує, що жодне з розглянутих рішень не повторює цільову постановку задачі LvStore. Торгсофт має сильний блок обліку, але не демонструє власну Frontend-архітектуру. WooCommerce добре підходить для електронної комерції, але не орієнтується на внутрішні замовлення між магазином і складом. Loyverse POS зручний для торгової точки, але не є основою для дослідження власної клієнтської частини.

Отже, розроблення власного вебзастосунку є обґрунтованим. LvStore поєднає керування товарами, контроль залишків, внутрішні замовлення та рольове розмежування в одному інтерфейсі.

1.4 Огляд та обґрунтування ІТ-засобів для реалізації вебзастосунок

Для розробки Frontend-частини системи LvStore необхідно було обрати такі ІТ-засоби, які дозволяють побудувати зручний, підтримуваний і рольово-орієнтований вебзастосунок. Оскільки система передбачає роботу кількох категорій користувачів, а саме адміністратора магазину, продавця та менеджера складу, клієнтська частина повинна забезпечувати не лише відображення сторінок, а й організацію захищених маршрутів, повторне використання UI-компонентів, взаємодію з REST API, обробку серверних даних і можливість тестування основних сценаріїв.

Під час вибору технологій враховувалися кілька критеріїв. По-перше, засоби розробки повинні підтримувати компонентний підхід, оскільки інтерфейс LvStore складається з багатьох повторюваних елементів: таблиць, форм, кнопок дій, діалогових вікон, бокових меню, верхніх панелей і сторінок кабінетів. По-друге, важливою була підтримка типізації, оскільки застосунок працює з різними структурами даних: товарами, користувачами, магазинами, складами, залишками та замовленнями. По-третє, технологічний стек мав забезпечити зручну маршрутизацію, контроль доступу за ролями, централізовану взаємодію з API та можливість перевірки роботи інтерфейсу за допомогою тестів.

1.4.1 React

Основою клієнтської частини було обрано бібліотеку React. Її використання є доцільним для LvStore, оскільки React дозволяє будувати інтерфейс через окремі компоненти [12]. Для системи менеджменту товарів це має практичне значення, адже багато сторінок мають схожу структуру, але працюють із різними даними. Наприклад, сторінки товарів, замовлень, складів і залишків використовують таблиці, кнопки керування, фільтри, пагінацію та модальні вікна. Компонентний підхід дає змогу не створювати кожен елемент з нуля, а виділяти спільні частини інтерфейсу і повторно використовувати їх у різних кабінетах.

React також підходить для побудови рольових кабінетів. У LvStore передбачено окремі робочі зони для адміністратора, продавця та менеджера складу. Кожна з них має власну оболонку, набір сторінок і функціональні можливості. Завдяки компонентній структурі можна відокремити загальний макет кабінету від конкретних сторінок, які відкриваються всередині нього. Це спрощує підтримку системи, оскільки зміна однієї сторінки не потребує переписування всього кабінету.

Важливою перевагою React є декларативний підхід до побудови інтерфейсу. Розробник описує, яким має бути відображення залежно від стану даних, а бібліотека оновлює потрібні частини сторінки. Для LvStore це важливо, оскільки дані можуть змінюватися після створення товару, редагування користувача, формування замовлення або оновлення залишків.

1.4.2 TypeScript

Для підвищення надійності коду було обрано TypeScript. Він розширює JavaScript механізмом статичної типізації та дозволяє описувати структуру даних, параметри функцій, властивості компонентів і відповіді API [13]. У системі LvStore це має особливе значення, оскільки однакові сутності використовуються в різних частинах застосунку. Наприклад, товар може відображатися в кабінеті адміністратора, продавця і менеджера складу, але в кожному випадку з різним набором доступних дій.

У межах LvStore TypeScript доцільний також через наявність багатьох форм. Користувачі створюють або редагують товари, замовлення, склади, користувачів і залишки. Якщо структура даних описана типами, розробник може швидше помітити помилку ще під час написання коду. Наприклад, якщо функція очікує числовий ідентифікатор товару, а компонент передає значення іншого типу, така проблема може бути виявлена до запуску застосунку в браузері.

1.4.3 Vite

Для запуску середовища розробки та формування production-збірки було обрано Vite [14]. Цей інструмент добре підходить для React і TypeScript-проектів, оскільки забезпечує швидкий локальний запуск і швидке оновлення сторінки після змін у коді. Під час розробки Frontend-частини це важливо, тому що інтерфейс постійно перевіряється в браузері: змінюється розміщення елементів, вигляд форм, поведінка таблиць, модальних вікон і станів завантаження.

Vite також є доцільним для LvStore через простоту конфігурації. У проекті головна увага приділяється реалізації сторінок, маршрутизації, рольового доступу та взаємодії з API, тому інструмент збірки не повинен ускладнювати структуру зайвими налаштуваннями. Це дозволяє зосередитися на логіці клієнтської частини, а не на складному налаштуванні середовища розробки.

1.4.4 Chakra UI

Для побудови користувацького інтерфейсу було обрано Chakra UI. Ця бібліотека надає готові React-компоненти для створення кнопок, полів введення, таблиць, контейнерів, діалогових вікон і елементів компоновання [15]. Для системи LvStore це є практично корисним, оскільки застосунок містить багато типових службових елементів. Адміністратор, продавець і менеджер складу працюють із таблицями, формами, кнопками дій і модальними вікнами, тому важливо забезпечити однаковий стиль і передбачувану поведінку інтерфейсу.

Використання Chakra UI дозволяє скоротити час на створення базових елементів і зосередитися на логіці застосунку. Наприклад, замість ручного написання стилів для кожної кнопки або форми можна використовувати готові компоненти й адаптувати їх через властивості. Це особливо важливо для ролей системи, оскільки різні кабінети мають бути візуально узгодженими, але містити різний набір функцій.

1.4.5 React Router

Для організації переходів між сторінками було використано React Router [16]. У LvStore маршрутизація є одним із ключових елементів клієнтської архітектури, оскільки застосунок має публічну частину, сторінку входу і три захищені рольові зони. Після авторизації користувач повинен потрапляти не в загальний інтерфейс, а до кабінету, який відповідає його ролі. Для цього в маршрутах передбачено окремі зони адміністратора, продавця і менеджера складу.

React Router дозволяє реалізувати вкладені маршрути, що є зручним для побудови рольових кабінетів. Наприклад, оболонка кабінету адміністратора формує загальний макет із боковим меню і верхньою панеллю, а всередині неї відкриваються сторінки товарів, замовлень, магазинів, користувачів і залишків. Аналогічний принцип застосовується для продавця і менеджера складу.

Окреме значення має role-based доступ. У LvStore обмеження реалізується не лише через приховування пунктів меню, а й на рівні маршрутів. Якщо користувач спробує вручну перейти до чужого кабінету через URL, guard-компонент повинен заблокувати такий доступ.

1.4.6 Axios

Для взаємодії з серверною частиною було обрано Axios [17]. Він використовується як HTTP-клієнт для виконання запитів до REST API. У LvStore взаємодія з API є основою роботи всіх рольових кабінетів, оскільки товари, користувачі, склади, залишки і замовлення зберігаються на сервері. Frontend-частина повинна отримувати ці дані, відображати їх у таблицях, а також надсилати запити на створення, редагування або видалення записів.

Важливо, що в проєкті HTTP-запити не формуються безпосередньо на сторінках. Для цього використовується окремий API-шар і налаштований Axios Instance. Такий підхід дозволяє централізовано задавати базову адресу серверної частини, параметри запитів і правила обробки помилок. Якщо адреса API або налаштування взаємодії змінюються, їх можна оновити в одному місці, не переписуючи всі сторінки застосунку.

1.4.7 TanStack Query

Для керування серверним станом було обрано TanStack Query [18]. На відміну від локального стану компонента, серверний стан залежить від відповіді API, може змінюватися після дій користувачів і потребує оновлення після створення, редагування або видалення записів. У системі управління товарними запасами це особливо важливо, оскільки користувачі повинні бачити актуальну інформацію про товари, залишки та замовлення.

У LvStore TanStack Query використовується для отримання даних через `useQuery` і виконання змін через `useMutation`. Такий підхід зменшує дублювання коду, оскільки сторінкам не потрібно самостійно реалізовувати всі стани завантаження, помилки і повторного отримання даних. Після успішної мутації можна інвалідувати відповідний ключ запиту, щоб оновити таблицю. Наприклад, після створення товару оновлюється список товарів, а після створення складу - список складів.

1.4.8 Jest і React Testing Library

Для перевірки роботи Frontend-частини було обрано Jest і React Testing Library [19, 20]. Ці інструменти використовуються для автоматизованого тестування JavaScript і TypeScript-коду, а також React-компонентів. Для LvStore тестування є важливим через наявність авторизації, захищених маршрутів, `role-based` доступу, форм і модальних вікон. Помилка в цих частинах може призвести до некоректного доступу або порушення робочого сценарію користувача.

Jest відповідає за запуск тестів і перевірку очікуваних результатів, а React Testing Library дозволяє тестувати компоненти через поведінку, близьку до дій користувача. Це означає, що перевіряється не внутрішня реалізація компонента, а те, що бачить і виконує користувач. Наприклад, можна перевірити появу повідомлення про помилку після некоректного заповнення форми або відображення потрібної сторінки після проходження `guard`-компонента.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Вимоги до програмного забезпечення

На основі аналізу предметної галузі та розгляду існуючих рішень сформовано вимоги до програмного забезпечення для управління товарами, залишками і внутрішніми замовленнями у мережі магазинів. Вимоги визначають, які дії повинні бути доступні користувачам, які ролі підтримує система, як має поводитися інтерфейс під час роботи з даними і які обмеження необхідно врахувати під час розробки [7, 24, 25].

Функціональні вимоги описують поведінку системи з погляду користувача. Нефункціональні вимоги описують якість роботи: швидкодію, сумісність, безпеку, підтримуваність і адаптивність. Такий поділ дає змогу послідовно перейти від аналізу предметної галузі до проєктування сторінок, ролей, маршрутів і компонентів.

2.1.1 Функціональні вимоги

Frontend-частина повинна забезпечувати автентифікацію та авторизацію користувачів. Користувач входить до системи через сторінку авторизації, після чого система отримує дані поточного користувача і визначає його роль. Відповідно до ролі користувач переходить до кабінету адміністратора, продавця або менеджера складу [7, 21].

Система повинна підтримувати розмежування прав доступу. Продавець, адміністратор магазину і менеджер складу не повинні працювати в одному загальному кабінеті. Для кожної ролі передбачено окремий набір сторінок, окрему навігацію і власні сценарії. Це зменшує ризик випадкового виконання дій, які не належать користувачу.

Frontend-частина повинна забезпечувати перегляд товарної номенклатури і залишків. Користувачі мають бачити актуальні дані про товари, категорії, штрихкоди, ціни, залишки магазинів і складські залишки.

Для продавця основним є залишок магазину, для менеджера складу - складський запас, для адміністратора - загальний контроль магазинної частини.

Окремою вимогою є робота із замовленнями. Продавець повинен створювати замовлення на поповнення, адміністратор - переглядати, підтверджувати, скасовувати або редагувати замовлення, а менеджер складу - переглядати і відвантажувати замовлення. В інтерфейсі мають відображатися статуси, щоб користувач розумів поточний стан заявки.

Після створення, редагування або видалення запису застосунк повинен оновлювати відповідні дані. Якщо адміністратор створив товар, таблиця товарів має оновитися. Якщо продавець створив замовлення, воно має з'явитися в списку. Якщо менеджер складу створив склад, список складів повинен відобразити новий запис [8, 18].

2.1.2 Нефункціональні вимоги

До нефункціональних вимог належить продуктивність. Початкове відображення основного контенту повинно залишатися швидким при стандартному 4G-з'єднанні. Сторінки з таблицями мають коректно працювати з обсягом до 500 записів без помітних затримок під час перегляду, прокрутки або оновлення даних. Для цього у застосунку використовується розділення сторінок, пагінація і кешування серверних даних [8, 18, 23].

Сумісність передбачає коректну роботу застосунку в актуальних версіях Google Chrome, Mozilla Firefox і Microsoft Edge. Основними для роботи є десктопні і планшетні екрани, оскільки таблиці товарів, залишків і замовлень потребують достатньої ширини. Мобільний режим також перевіряється для головної сторінки, авторизації і базових елементів навігації.

Безпека обміну даними передбачає, що доступ до робочих кабінетів мають тільки авторизовані користувачі. На рівні Frontend це реалізується через захищені маршрути і перевірку ролі. Взаємодія з серверною частиною повинна виконуватися через захищене з'єднання. Користувач не повинен отримати доступ до чужого кабінету при прямому введенні адреси маршруту [21, 22].

2.2 Ролі користувачів і сценарії роботи

У системі LvStore передбачено три основні ролі користувачів: адміністратор магазину, продавець і менеджер складу. Такий поділ пов'язаний з реальною організацією роботи мережі магазинів. У кожній ролі є своя зона відповідальності, тому інтерфейс не повинен бути однаковим для всіх користувачів [7-9].

Окрім основних сценаріїв роботи, система LvStore передбачає можливість гнучкого розширення функціоналу для кожної ролі відповідно до змін бізнес-процесів мережі магазинів. Це дозволяє адаптувати інтерфейс та набір доступних дій під нові вимоги, забезпечуючи масштабованість і актуальність системи у разі появи нових ролей або змін у структурі організації.

Адміністратор магазину відповідає за керування товарами, магазинами, користувачами, залишками магазинів і замовленнями. Він має найширший доступ у межах магазинної частини системи, оскільки повинен підтримувати довідники та контролювати стан заявок. Продавець працює з товарами, залишками свого магазину і замовленнями на поповнення. Менеджер складу відповідає за склади, складські залишки, відвантаження і перегляд замовлень.

Рольовий поділ також визначає зовнішній вигляд кабінетів. Кабінет адміністратора містить ширший набір пунктів меню. Кабінет продавця є простішим і не містить зайвих адміністративних сторінок. Кабінет менеджера складу орієнтований на складські операції.

Розподіл функцій між ролями користувачів наведено в таблиці 2.1.

Розподіл функцій між ролями користувачів

Роль	Основні функції
Адміністратор магазину	Керування товарами, магазинами та користувачами; перегляд залишків магазинів; створення, редагування, підтвердження та скасування замовлень.
Продавець	Перегляд товарів, залишків магазину та замовлень; створення замовлень; позначення отримання товару.
Менеджер складу	Перегляд і відвантаження замовлень; скасування замовлень; керування складами і складськими залишками; перегляд товарів та залишків.

Розподіл функцій між ролями впливає на всі наступні етапи проєктування. Він визначає, які маршрути потрібно створити, які сторінки входять до кожного кабінету, які guard-компоненти потрібні для захисту доступу і які API-модулі використовуються на конкретних сторінках. У результаті система має не один універсальний інтерфейс, а три робочі середовища, налаштовані під різні задачі.

2.3 Проєктування варіантів використання вебзастосунку

Варіанти використання описують способи взаємодії користувачів із системою. Для вебзастосунку LvStore така діаграма потрібна для того, щоб показати, які дії доступні неавторизованому користувачу та які функції отримують користувачі після входу до системи. Діаграма не деталізує програмний код, але відображає основні сценарії роботи на рівні користувача [7, 24, 25].

У процесі проєктування враховано, що вхід до системи є початковим сценарієм. Після авторизації користувач не обирає кабінет самостійно. Система отримує інформацію про користувача, визначає його роль і виконує перехід до відповідного кабінету.

Діаграму варіантів використання системи LvStore наведено на рисунку 2.1.

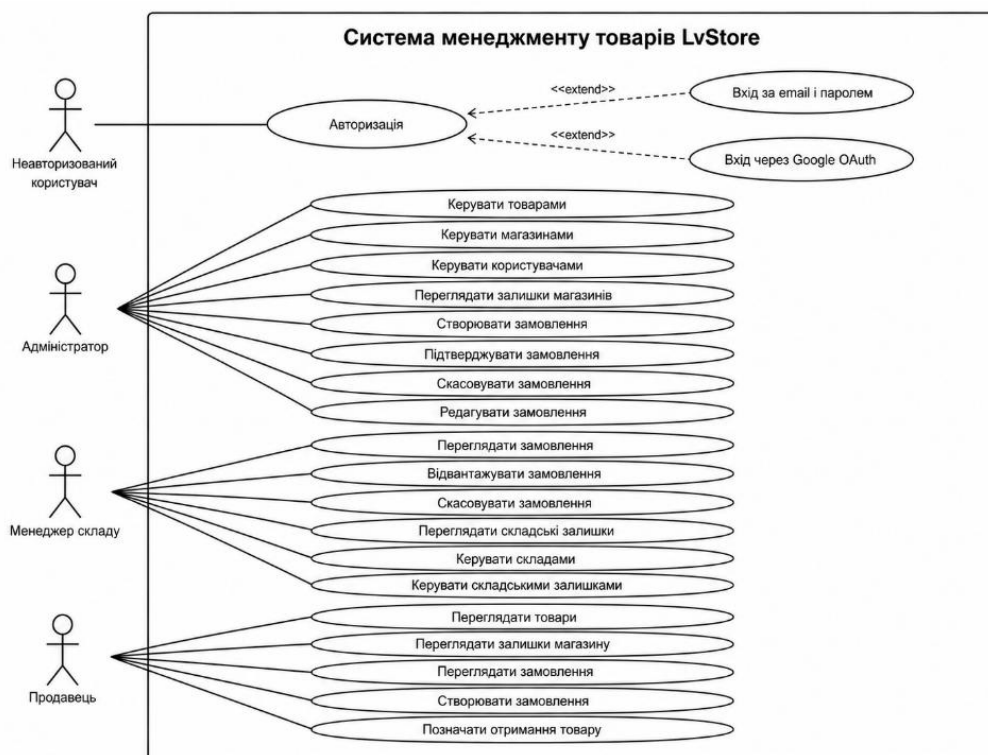


Рис. 2.1 Діаграма варіантів використання системи LvStore

На даній діаграмі основними акторами є неавторизований користувач, адміністратор, менеджер складу і продавець. Неавторизований користувач взаємодіє зі сценарієм входу до системи. Для входу передбачено два альтернативні варіанти: за email і паролем або через Google OAuth.

Адміністратор працює в кабінеті адміністратора. Йому доступне керування товарами, магазинами і користувачами, перегляд залишків магазинів, створення, підтвердження, скасування і редагування замовлень. Ці дії відповідають ролі, яка відповідає за організацію магазинної частини системи.

Менеджер складу працює в кабінеті менеджера складу. Йому доступні перегляд замовлень, відвантаження, скасування, перегляд складських залишків, перегляд товарів, керування складами і керування складськими залишками. На відміну від адміністратора, ця роль зосереджена на складській логіці.

Продавець працює в кабінеті продавця. Він переглядає товари, залишки магазину, замовлення, створює нове замовлення. Такий набір дій є достатнім для роботи торгової точки і не містить зайвого адміністративного функціоналу.

2.4 Проєктування клієнтської архітектури вебзастосунку

Клієнтська архітектура LvStore спроектована як окремий вебзастосунок, що працює в браузері та обмінюється даними із сервером через REST API. Такий підхід дозволяє розділити зону відповідальності між інтерфейсом і серверною логікою. Користувацька частина відповідає за сторінки, форми, таблиці, маршрутизацію, перевірку доступу та відображення результатів запитів. Серверна частина забезпечує зберігання даних, обробку операцій і повернення відповідей для клієнта [9, 16-18].

У структурі застосунку виділено кілька рівнів. Точка входу `main.tsx` ініціалізує основні провайдери, необхідні для роботи інтерфейсу. Маршрутизатор визначає доступні сторінки, публічні маршрути та захищені рольові зони. Guard-компоненти перевіряють, чи має користувач право перейти до відповідного кабінету. Рольові оболонки формують загальний вигляд робочої області, а сторінки реалізують конкретні функції, пов'язані з товарами, замовленнями, магазинами, складами і залишками.

Такий поділ зменшує залежність між частинами коду. Сторінка товарів не повинна містити налаштування HTTP-клієнта, а кабінет продавця не повинен включати логіку адміністратора. Компоненти навігації відповідають за переходи між сторінками, але не виконують операції створення або редагування даних. Завдяки цьому кожен блок має власне призначення, а зміни в одному місці менше впливають на інші частини системи.

Окреме місце займає API-шар. Він ізолює HTTP-запити від сторінок, тому компоненти інтерфейсу не працюють безпосередньо з адресами серверних маршрутів. Сторінка викликає функцію з відповідного модуля, наприклад для товарів, замовлень або складів, а вже цей модуль звертається до налаштованого `Axios Instance`. У результаті код сторінок залишається коротшим, а логіку обміну даними легше перевіряти і змінювати.

Для роботи із серверним станом використано TanStack Query. Цей інструмент відповідає за отримання даних, кешування, обробку станів завантаження і повторне оновлення після мутацій. У межах системи це потрібно для таблиць товарів, замовлень, магазинів, складів, залишків магазинів і складських залишків. Після створення або редагування запису відповідний запит інвалідується, і користувач бачить оновлені дані без ручного перезавантаження сторінки.

Рольові оболонки AdminShell, SellerShell і WarehouseShell мають окрему навігацію та власні вкладені сторінки. Адміністратор отримує доступ до керування товарами, магазинами, користувачами, замовленнями і залишками. Продавець працює з товарами, залишками магазину і замовленнями. Менеджер складу має інтерфейс для складських процесів, відвантаження замовлень, керування складами та складськими залишками. Така організація підтримує обрану модель доступу і відповідає логіці предметної галузі.

Архітектурне рішення побудоване з урахуванням подальшого розширення. Якщо потрібно додати нову сторінку для певної ролі, її можна розмістити у відповідній частині структури і підключити до routes.tsx. Якщо з'являється нова група серверних запитів, її доцільно винести в окремий API-модуль. Завдяки цьому застосунок можна розвивати без повної зміни наявної структури.

2.5 Проєктування компонентної структури Frontend-частини

Компонентна структура визначає основні модулі клієнтського застосунку та їхні взаємозв'язки, дозволяючи детально описати, які частини формують архітектуру інтерфейсу, які з них є внутрішніми, а які зовнішніми залежностями. Для вебзастосунку такий підхід потрібний для пояснення роботи провайдерів, маршрутизації, сторінки авторизації, guard-компонентів, рольових оболонок, сторінок ролей, API-шару і спільних UI-компонентів [9, 12, 15, 16, 18].

Діаграма компонентів для вебзастосунку LvStore відображає, що клієнтська частина запускається у середовищі Web Browser, а для роботи із зовнішніми системами використовує Backend REST API та Google OAuth. Усередині рамки Frontend-частини розміщено основні компоненти, які формують структуру застосунку.

Діаграму компонентів наведено на рисунку 2.2.

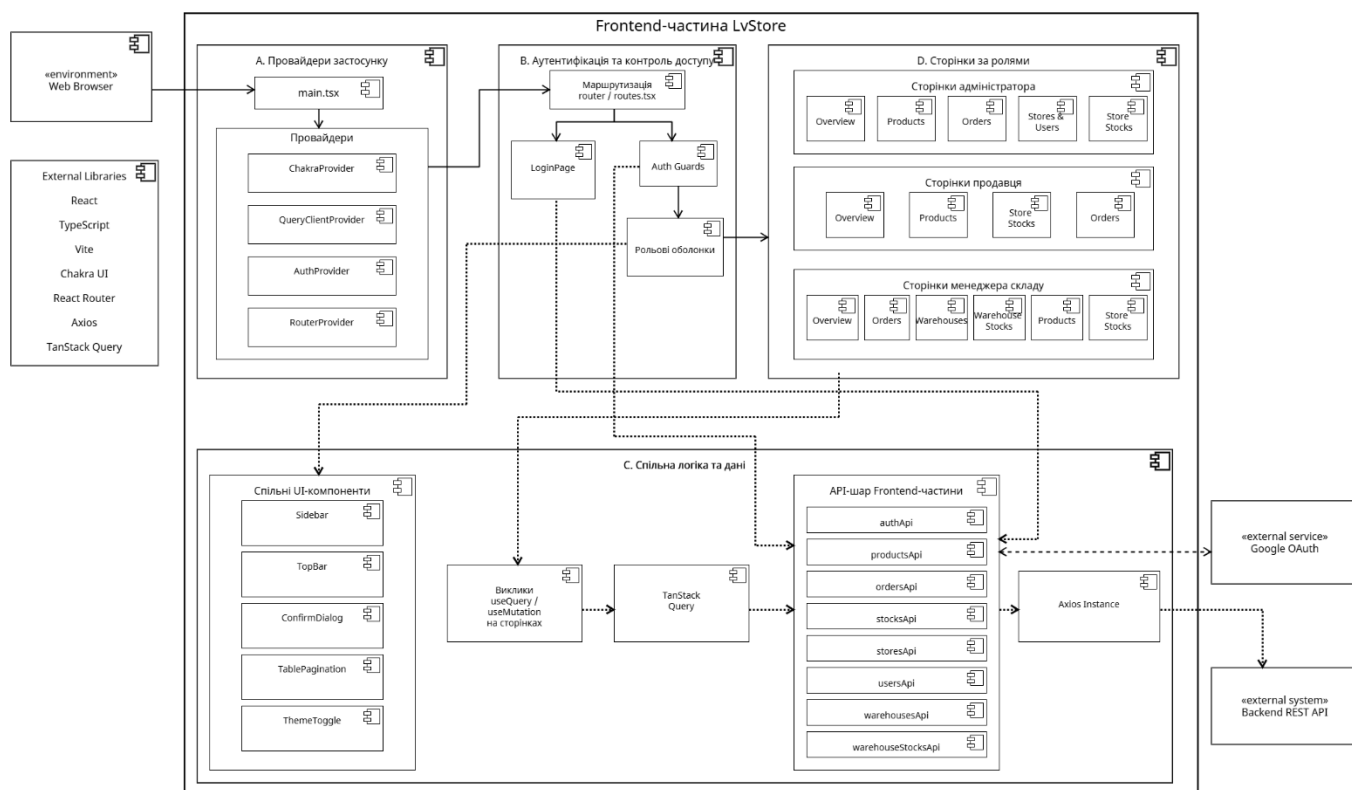


Рис. 2.2 Діаграма компонентів Frontend-частини системи LvStore

У лівій частині діаграми показано вхід до Frontend-частини через `main.tsx` і блок провайдерів застосунку. Провайдери відповідають за підключення UI-бібліотеки, маршрутизації, контексту автентифікації та механізму роботи із серверним станом. Блок `External Libraries` подано як інформаційний перелік основних технологій, використаних у клієнтській частині.

Блок автентифікації та контролю доступу відображає маршрутизацію, сторінку входу, `Auth Guards` і рольові оболонки.

Маршрутизатор визначає, до якої частини застосунку має перейти користувач: до сторінки входу або до захищених маршрутів. Guard-компоненти перевіряють наявність поточного користувача та його роль.

Окреме виділення рольових оболонок потрібне для того, щоб показати відмінність між перевіркою доступу та формуванням робочого інтерфейсу. Auth Guards відповідають саме за допуск до захищеної частини застосунку, тоді як рольові оболонки організовують структуру кабінету, зокрема розміщення навігації, верхньої панелі та вкладених сторінок.

У правій частині діаграми відображено сторінки за ролями адміністратора, продавця та менеджера складу. Для кожної ролі передбачено окремий набір сторінок, що відповідає її функціям у системі. Такий поділ дозволяє не змішувати можливості різних користувачів і підтримує логіку role-based доступу у Frontend-частині LvStore.

Блок спільної логіки та даних показує основний ланцюжок взаємодії сторінок із серверними даними. Сторінки за ролями використовують виклики `useQuery` та `useMutation`, після чого обробка запитів переходить до TanStack Query. Далі запити передаються до API-шару Frontend-частини, який через Axios Instance взаємодіє з Backend REST API. Окремо на діаграмі показано зовнішній сервіс Google OAuth, до якого звертається `authApi` під час авторизації через Google.

Такий розподіл компонентів дозволяє відокремити інтерфейсні частини від логіки отримання та зміни даних. Сторінки не звертаються до серверної частини напряму, а використовують окремий API-шар і налаштований Axios Instance, що робить структуру Frontend-частини більш зрозумілою та зручною для подальшого розширення.

Спільні UI-компоненти винесені в окремий блок, оскільки вони повторно використовуються в різних частинах інтерфейсу. До них належать елементи навігації, верхня панель, діалогові вікна підтвердження, пагінація таблиць і перемикач теми. Це дозволяє зберігати єдиний стиль інтерфейсу та зменшувати дублювання коду в різних сторінках застосунку.

2.6 Проєктування навігаційної карти вебзастосунку

Карта вебзастосунку показує загальну структуру сторінок. Вона не описує внутрішні компоненти коду, а демонструє, як організовано навігацію для користувача. У карті виділено головну сторінку, сторінку входу і три кабінети за ролями.

Карту вебзастосунку наведено на рисунку 2.3.

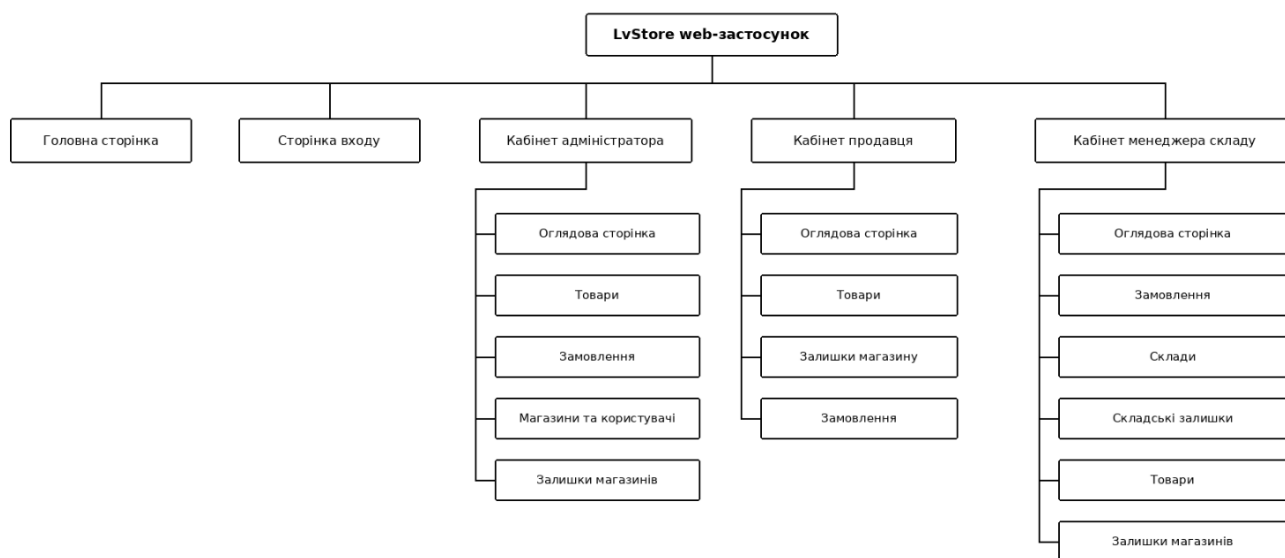


Рис. 2.3 Карта вебзастосунку LvStore

Головна сторінка є публічною і доступна без авторизації. Сторінка входу містить авторизацію та перенаправлення за роллю. Кабінет адміністратора містить огляд, товари, замовлення, магазини та користувачів, а також залишки магазинів. Кабінет продавця містить огляд, товари, залишки магазину і замовлення. Кабінет менеджера складу містить огляд, замовлення, склади, складські залишки, товари і залишки магазинів.

Карта дає змогу швидко зрозуміти, які сторінки існують у застосунку і як вони групуються за ролями. Вона також підтверджує, що секції головної сторінки не винесені в окремі сторінки, а залишаються блоками одного лендінгу [7, 16].

2.7 Проектування інтерфейсу користувача

Проектування інтерфейсу користувача виконувалося з урахуванням функціональних вимог, ролі моделі доступу та основних сценаріїв роботи користувачів у системі LvStore. Основними вимогами до інтерфейсу були зрозуміла навігація, швидкий доступ до робочих сторінок, мінімальна кількість зайвих дій під час виконання типових операцій, однакове розміщення елементів керування та візуальне розділення публічної і захищеної частин вебзастосунку. Для попереднього проектування макетів було використано платформу Figma, що дозволило визначити структуру сторінок, розташування навігації, таблиць, кнопок, форм і модальних вікон до етапу програмної реалізації [10, 11].

Розроблені прототипи стали основою для реалізації технічної архітектури системи менеджменту товарів для оптимізації мережі магазинів. Головна сторінка містить публічний блок представлення системи, навігацію, секції з перевагами, ролями, порядком початку роботи і контактами. У макеті передбачено кнопку переходу до входу. Макет головної сторінки наведено на рисунку 2.4.

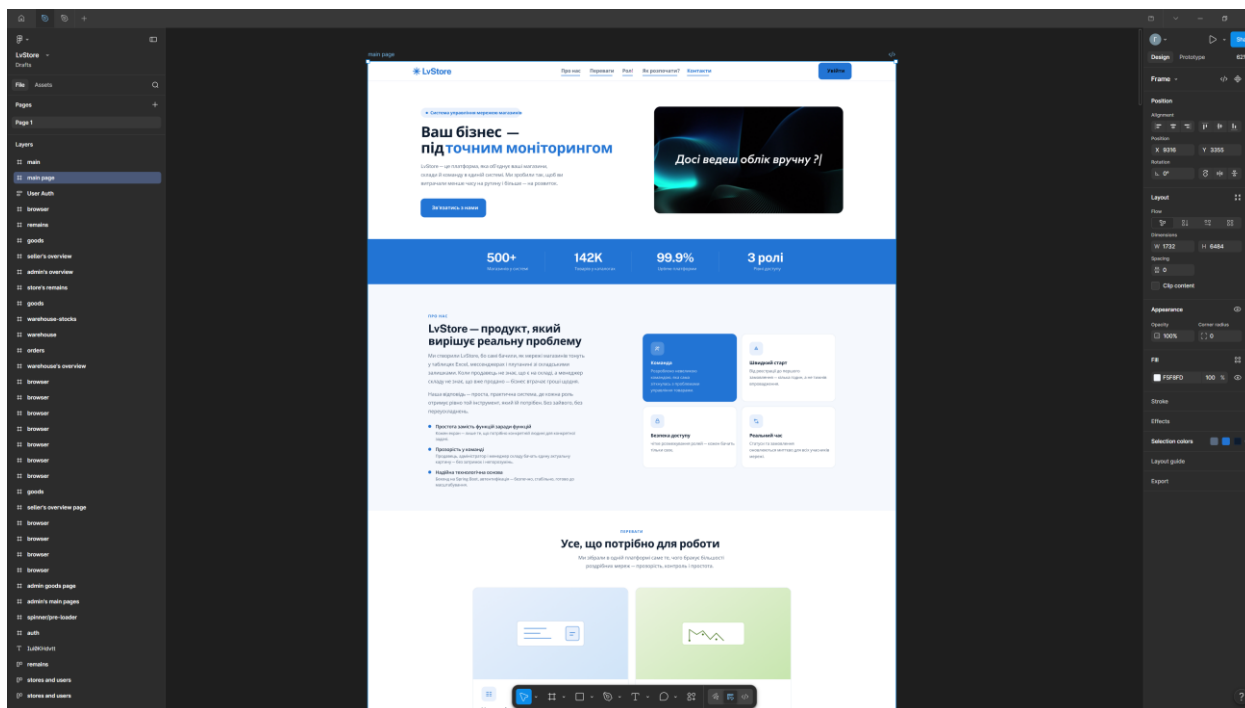


Рис. 2.4 Макет головної сторінки вебзастосунку

Сторінка авторизації була спроектована як окремий екран для входу користувача до системи. Її завдання полягає в тому, щоб забезпечити введення облікових даних, перевірку заповнення полів і можливість входу через Google OAuth. Основний сценарій використання передбачає, що користувач вводить email і пароль, погоджується з умовами використання та після успішної автентифікації переходить до кабінету відповідно до своєї ролі.

Макет сторінки авторизації наведено на рисунку 2.5.

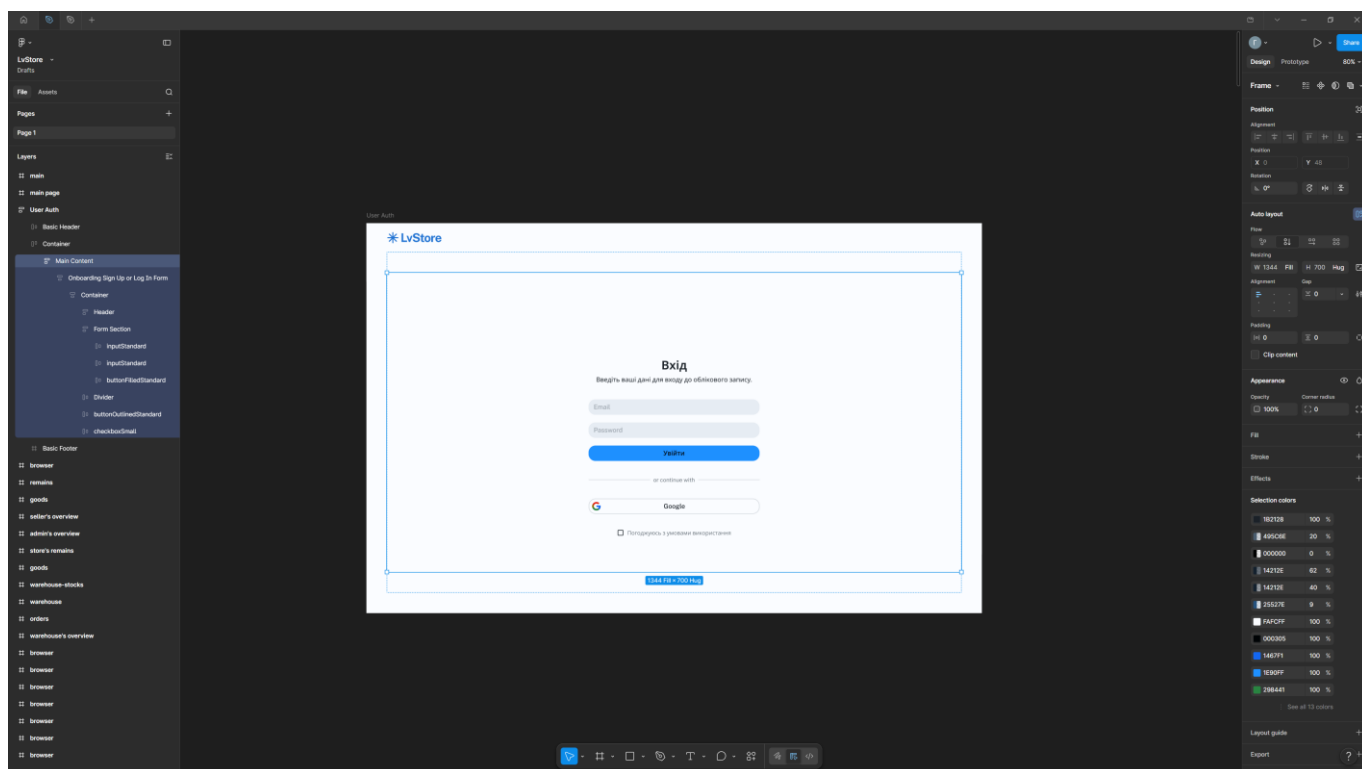


Рис. 2.5 Макет сторінки авторизації

У макеті сторінки авторизації форма розміщена по центру, що зменшує кількість відволікаючих елементів і концентрує увагу користувача на виконанні однієї дії - входу до системи. Для полів email і пароля передбачено зрозумілі підказки, а для некоректного введення даних - повідомлення про помилки. Це відповідає вимогам юзабіліті щодо зменшення кількості помилок користувача під час заповнення форми.

Кабінет адміністратора магазину проєктувався як найбільш функціонально насичена частина системи. Адміністратор працює з товарами, замовленнями, магазинами, користувачами та залишками, тому інтерфейс цієї ролі має забезпечувати швидке перемикання між розділами і зручну роботу з великими списками даних. Основний сценарій передбачає, що адміністратор відкриває потрібний розділ через бокове меню, переглядає таблицю записів і виконує дії створення, редагування або видалення.

Макет кабінету адміністратора магазину наведено на рисунку 2.6.

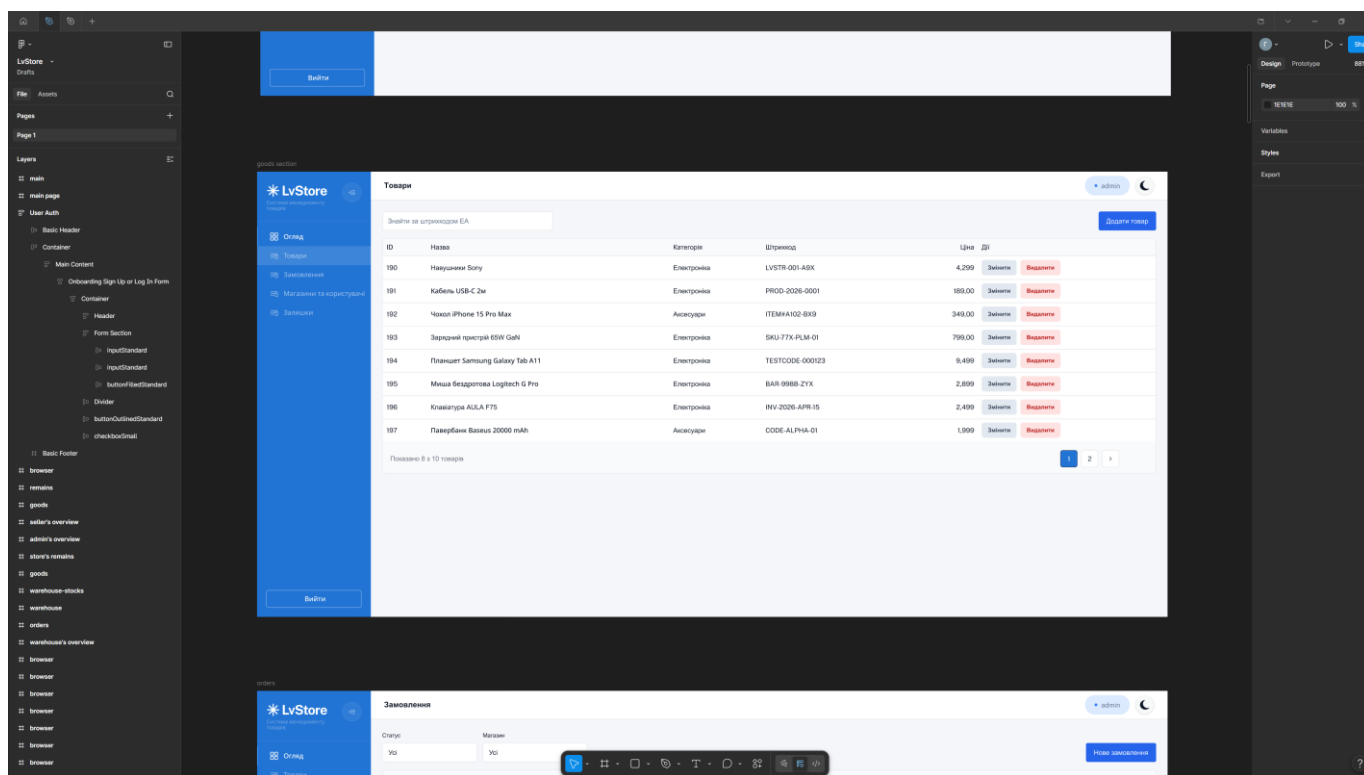


Рис. 2.6 Макет кабінету адміністратора магазину

У макеті кабінету адміністратора використано бокову навігацію, верхню панель, таблиці даних, кнопки дій і модальні вікна. Такий вибір елементів обґрунтований тим, що адміністратор працює з найбільшою кількістю сутностей і повинен швидко знаходити записи, керувати ними та виконувати критичні дії з додатковим підтвердженням. Однакова структура сторінок у межах кабінету спрощує роботу користувача і зменшує ризик випадкових змін.

Кабінет продавця був спроектований як спрощена робоча зона для щоденних операцій у торговій точці. На відміну від адміністратора, продавець не повинен керувати користувачами, магазинами або складами, тому в інтерфейсі залишено тільки ті сторінки, які безпосередньо пов'язані з його роботою. Основний сценарій передбачає перегляд товарів, перевірку залишків магазину, створення замовлення на поповнення та позначення отримання товару.

Макет кабінету продавця наведено на рисунку 2.7.

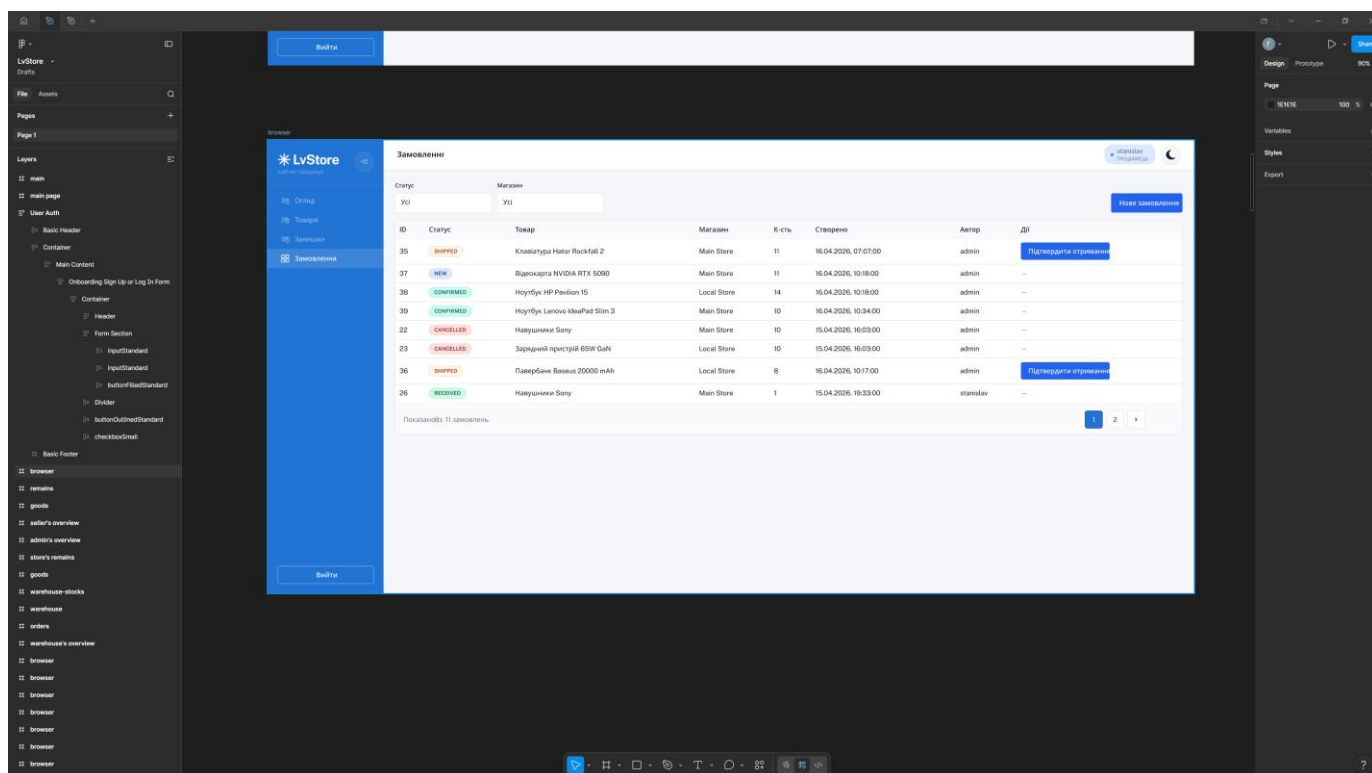


Рис. 2.7 Макет кабінету продавця

У макеті кабінету продавця передбачено бокове меню з обмеженим набором розділів, таблиці для перегляду товарів і залишків, а також сторінку замовлень. Такий підхід відповідає вимозі мінімізації інтерфейсу відповідно до задач користувача: продавець бачить лише потрібні функції, може швидко знайти товар або замовлення за допомогою таблиць, фільтрів і кнопок дій та не відволікається на адміністративні можливості.

Кабінет менеджера складу проєктувався для виконання операцій, пов'язаних зі складами, складськими залишками, товарами та обробкою замовлень. Ця роль має інший набір задач, ніж продавець або адміністратор магазину, оскільки основна увага приділяється контролю складських запасів і виконанню замовлень. Основний сценарій роботи передбачає перегляд замовлень, керування складами, перевірку складських залишків і виконання складських операцій.

Макет кабінету менеджера складу наведено на рисунку 2.8.

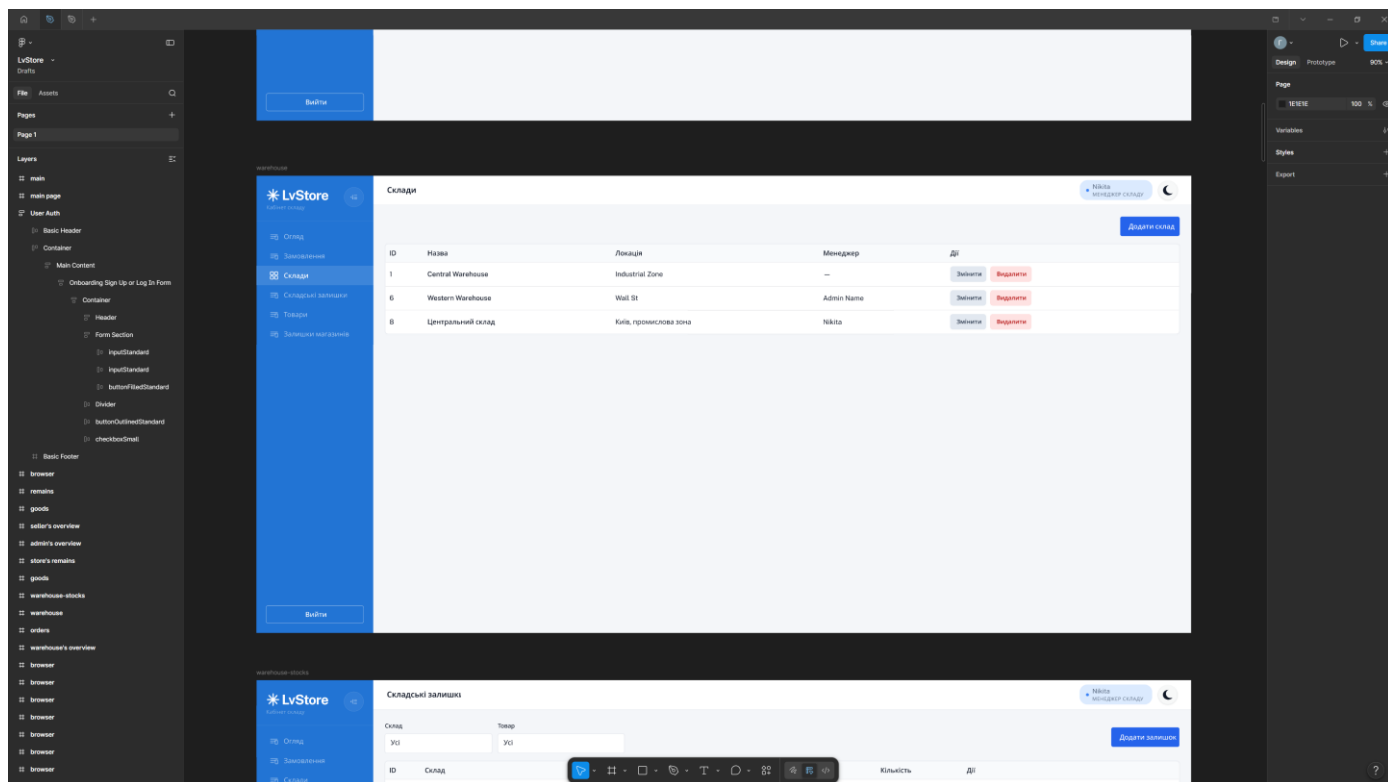


Рис. 2.8 Макет кабінету менеджера складу

У макеті кабінету менеджера складу передбачено сторінки замовлень, складів, складських залишків, товарів і залишків магазинів. Такий поділ дозволяє відокремити складські процеси від адміністративних дій магазину, а використання таблиць, статусів, кнопок дій, форм і модальних вікон забезпечує зручне виконання операцій без втрати контексту поточної сторінки. Це робить інтерфейс придатним для контролю складської частини системи та обробки заявок від магазинів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура та структура вебзастосунку

3.1.1 Файлова структура вебзастосунку

Організація структури проєкту є одним з базових етапів розробки вебзастосунку. Логічний поділ кодової бази полегшує навігацію, робить зв'язки між файлами зрозумілішими і спрощує подальший супровід. У вебзастосунку LvStore клієнтська частина реалізована за допомогою React, TypeScript, Vite, Chakra UI, React Router, Axios, TanStack Query, Jest і React Testing Library [12-20].

Структуру вебзастосунку у середовищі розробки наведено на рисунку 3.1.

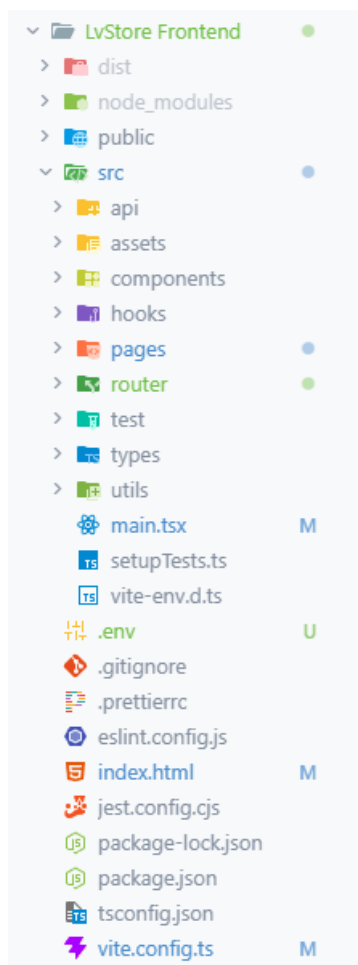


Рис. 3.1 Структура вебзастосунку у середовищі розробки

Основний код клієнтської частини розміщено в директорію «src». У цій частині проєкту знаходяться точка входу `main.tsx`, налаштування тестового середовища `setupTests.ts`, файл типів `vite-env.d.ts`, сторінки, маршрути, API-модулі, типи даних і допоміжна логіка. Такий поділ дозволяє відокремити прикладний код від конфігураційних файлів проєкту та службових ресурсів збірки.

Модулі для взаємодії із серверною частиною винесено в директорію «api». У цій частині розміщено `authApi`, `axiosInstance`, `ordersApi`, `productsApi`, `stocksApi`, `storesApi`, `usersApi`, `warehousesApi` і `warehouseStocksApi`. Кожен модуль відповідає за окрему групу запитів. Завдяки цьому сторінки не містять прямого HTTP-коду, а звертаються до готових функцій відповідного API-шару.

Статичні ресурси зберігаються в директорії «assets». Тут розміщуються іконки, зображення, SVG-файли та матеріали, імпортовані з макетів Figma. Ці ресурси використовуються на головній сторінці, сторінці входу та в елементах навігації. Винесення графічних матеріалів в окрему частину структури спрощує їх пошук і заміну під час доопрацювання інтерфейсу.

Директорія «components» включає компоненти загального призначення. Окремо виділено `layout`, де знаходяться компоненти `AppShell`, `Sidebar`, `TopBar`, `navConfig`, `useActiveSectionHash` і `useSectionNavigation`. Ці елементи пов'язані з побудовою загальної структури сторінок, навігацією та переходами між секціями публічної частини.

Користувацькі хуки винесено в директорію «hooks». Наприклад хук `useProductSearchTypewriter` реалізує допоміжну поведінку для пошукового поля. Винесення такої логіки окремо зменшує навантаження на компоненти сторінок і дозволяє повторно використовувати поведінку без дублювання коду.

У директорії «pages» згруповано сторінки застосунку. Усередині цієї частини виділено auth, main, admin, seller і warehouse. Auth відповідає за сторінку авторизації. Main містить головну сторінку та її секції. Admin, seller і warehouse формують окремі рольові зони, що відповідає логіці доступу в системі.

Маршрути застосунку описано в директорії «router». Основним файлом тут є routes.tsx, де визначено публічні сторінки, сторінку входу і захищені маршрути для трьох ролей. Такий підхід дозволяє централізовано керувати переходами між сторінками і підтримувати рольову модель доступу.

У директорії «types» зібрано типи даних. Вони використовуються в API-модулях, сторінках і формах, що допомагає підтримувати узгодженість між відповідями REST API та frontend-логікою. Допоміжні функції розміщено в «utils», зокрема обробку повідомлень про помилки API. Окрема частина test містить мок-файли для коректної роботи тестів із ресурсами.

У результаті файлова структура відповідає практичній задачі управління товарами, залишками і замовленнями. Рольові кабінети не змішуються між собою, API-запити відокремлені від інтерфейсу, а типи даних використовуються як спільна основа для сторінок і модулів взаємодії із сервером. Це робить проєкт зручнішим для супроводу та подальшого розширення.

3.1.2 Організація маршрутизації та role-based доступу

Маршрутизація у LvStore реалізована за допомогою React Router. Основна конфігурація маршрутів розміщена у файлі routes.tsx. У цьому файлі визначено публічний маршрут головної сторінки, маршрут сторінки входу і три захищені групи маршрутів для продавця, адміністратора та менеджера складу.

Налаштування маршрутизації наведено на рисунку 3.2 [16].

```

79 );
80 );
81 const WarehouseProductsPage = lazyPage(
82   () => import("../pages/warehouse/products/WarehouseProductsPage"),
83   "WarehouseProductsPage",
84 );
85 const WarehouseStocksPage = lazyPage(
86   () => import("../pages/warehouse/stocks/WarehouseStocksPage"),
87   "WarehouseStocksPage",
88 );
89 export const router = createBrowserRouter([
90   {
91     path: "/",
92     element: <AppShell />,
93     children: [{ index: true, element: <MainPage /> }],
94   },
95   { path: "/login", element: <LoginPage /> },
96   {
97     path: "/seller",
98     element: <SellerAuthGuard />,
99     children: [
100       {
101         element: <SellerShell />,
102         children: [
103           { index: true, element: <Navigate to="products" replace /> },
104           { path: "dashboard", element: <SellerOverviewPage /> },
105           { path: "products", element: <SellerProductsPage /> },
106           { path: "stocks", element: <SellerStocksPage /> },
107           { path: "orders", element: <SellerOrdersPage /> },
108         ],
109       },
110     ],
111   },
112   {
113     path: "/admin",
114     element: <AdminAuthGuard />,
115     children: [
116       {
117         element: <AdminShell />,
118         children: [
119           { index: true, element: <Navigate to="dashboard" replace /> },
120           { path: "dashboard", element: <AdminOverviewPage /> },
121           { path: "products", element: <AdminProductsPage /> },
122           { path: "orders", element: <AdminOrdersPage /> },
123           { path: "stores-users", element: <AdminStoresUsersPage /> },
124           { path: "stocks", element: <AdminStocksPage /> },
125         ],
126       },
127     ],
128   },
129   {
130     path: "/warehouse",
131     element: <WarehouseAuthGuard />,
132     children: [
133       {
134         element: <WarehouseShell />,
135         children: [
136           { index: true, element: <Navigate to="dashboard" replace /> },
137           { path: "dashboard", element: <WarehouseOverviewPage /> },
138           { path: "orders", element: <WarehouseOrdersPage /> },
139           { path: "warehouses", element: <WarehouseWarehousesPage /> },
140           { path: "warehouse-stocks", element: <WarehouseWarehouseStocksPage /> },
141           { path: "products", element: <WarehouseProductsPage /> },
142           { path: "stocks", element: <WarehouseStocksPage /> },
143         ],
144       },
145     ],
146   },
147 ],

```

Рис. 3.2 Налаштування маршрутизації у routes.tsx

Особливістю оновленої реалізації є використання відкладеного завантаження маршрутних компонентів через React.lazy. Для цього у проєкті створено допоміжну функцію lazyPage, яка дозволяє підключати компоненти сторінок, guard-компоненти та рольові оболонки через динамічний import. Такий підхід зменшує початковий обсяг коду, який завантажується під час відкриття застосунку, оскільки сторінки кабінетів підвантажуються лише тоді, коли користувач переходить до відповідної частини системи.

У файлі `main.tsx` маршрутизатор підключено через `RouterProvider`, який обгорнуто в `Suspense`. Для стану очікування використовується компонент `RouteFallback`, що відображається під час завантаження відкладених маршрутів. Це дозволяє уникнути ситуації, коли користувач переходить між сторінками, а інтерфейс тимчасово залишається без візуального стану завантаження. Таким чином, маршрутизація поєднує поділ сторінок на окремі частини та контрольоване відображення інтерфейсу під час їх підвантаження.

Кореневий маршрут відкриває головну сторінку через компоненти `AppShell` та `MainPage`, які також підключаються через механізм відкладеного завантаження. Сторінка входу розміщена за маршрутом `/login`. Для ролі продавця використовується шлях `/seller`, для адміністратора `/admin`, для менеджера складу `/warehouse`. Кожна рольова зона має власний `guard`-компонент, який перевіряє доступ до відповідного маршруту.

Перевірка прав доступу реалізована не в самих сторінках кабінетів, а в окремих `guard`-компонентах: `SellerAuthGuard`, `AdminAuthGuard` та `WarehouseAuthGuard`. Їх завдання полягає у перевірці поточного користувача і його ролі перед відкриттям захищеної частини застосунку. Після проходження перевірки користувач потрапляє до відповідної оболонки: `SellerShell`, `AdminShell` або `WarehouseShell`. Саме оболонка формує робочий макет кабінету з боковим меню і вкладеними сторінками.

У межах кабінету адміністратора визначено сторінки `dashboard`, `products`, `orders`, `stores-users` та `stocks`. У кабінеті продавця визначено сторінки `dashboard`, `products`, `stocks` і `orders`. У кабінеті менеджера складу визначено сторінки `dashboard`, `orders`, `warehouses`, `warehouse-stocks`, `products` і `stocks`. Для початкового переходу всередині рольових зон використовується перенаправлення через `Navigate`, що дозволяє відкривати базову сторінку кабінету без ручного введення повного вкладки маршруту.

Role-based доступ реалізовано не тільки через візуальне приховування пунктів меню. Обмеження працює на рівні маршрутів. Якщо користувач спробує перейти до чужої ролі через URL, відповідний guard-компонент повинен заблокувати такий доступ.

3.1.3 Організація взаємодії з API

Взаємодія з API організована через окрему групу модулів у директорії «api». Сторінки не створюють HTTP-запити напряму. Вони звертаються до функцій API-шару, а ці функції використовують Axios Instance.

Фрагмент реалізації взаємодії з API наведено на рисунку 3.3 [17, 18].

```
1 import axios from "axios";
2
3 export function getApiBaseUrl(): string {
4   const env = import.meta.env;
5   const fromBase = env.VITE_API_BASE_URL;
6   const fromLegacy = env.VITE_API_URL;
7   if (typeof fromBase === "string" && fromBase.trim() !== "") {
8     return fromBase.trim();
9   }
10  if (typeof fromLegacy === "string" && fromLegacy.trim() !== "") {
11    return fromLegacy.trim();
12  }
13  return "http://localhost:8080";
14 }
15
16 export const axiosInstance = axios.create({
17   baseURL: getApiBaseUrl(),
18   withCredentials: true,
19 });
```

Рис. 3.3 Фрагмент реалізації взаємодії з API

Axios Instance налаштований окремо і визначає базову адресу серверної частини. Адреса може братися зі змінних середовища VITE_API_BASE_URL або VITE_API_URL. Якщо ці значення не вказані, використовується локальна адреса. Такий підхід дозволяє запускати застосунок у різних середовищах без переписування сторінок.

API-модулі мають прикладне призначення. «AuthApi» відповідає за вхід, вихід, отримання поточного користувача і Google OAuth. «ProductsApi» відповідає за товари, «ordersApi» за замовлення, «stocksApi» за залишки магазинів, «storesApi» за магазини, «usersApi» за користувачів, «warehousesApi» за склади, «warehouseStocksApi» за складські залишки.

Усі функції API-шару мають типізовані вхідні та вихідні дані. Для цього використовуються інтерфейси і типи, винесені до спільної директорії «types». Наприклад, під час створення товару, користувача, замовлення або складського залишку сторінка передає не довільний об'єкт, а структуру, яка відповідає визначеному типу. Це зменшує ризик помилок під час розробки і робить взаємодію між інтерфейсом та серверною частиною більш передбачуваною.

Особливістю налаштування Axios Instance є використання параметра withCredentials. Завдяки цьому клієнтська частина може виконувати запити до захищених серверних маршрутів із передаванням облікових даних користувача. Це важливо для роботи авторизації, отримання поточного користувача через запит auth/me та перевірки доступу до рольових кабінетів. У результаті API-шар використовується не тільки для отримання товарів або замовлень, а й для підтримки загальної логіки автентифікації.

Сторінки використовують TanStack Query для отримання і зміни даних. Для читання застосовується useQuery, для створення, редагування або видалення useMutation. Після успішної дії виконується invalidateQueries для відповідного ключа. Це дозволяє одразу оновити таблицю і показати користувачу актуальні дані.

3.2 Реалізація користувацького інтерфейсу вебзастосунку

3.2.1 Реалізація головної сторінки вебзастосунку

Головна сторінка реалізована як публічна частина LvStore. Вона відкривається без авторизації і містить інформацію про призначення системи, переваги, ролі користувачів, порядок початку роботи та контакти.

Фрагмент реалізації головної сторінки наведено на рисунку 3.4.

```

LvStore Frontend > src > pages > main > MainPage.tsx > ...
11 import { MainStatsBar } from "../components/mainStatsBar";
12 import { useMainLandingQueries } from "../useMainLandingQueries";
13
14 export function MainPage() {
15   const { orderStatusCounts } = useMainLandingQueries();
16
17   return (
18     <Box
19       minH="100vh"
20       bg="#f5f8fd"
21       overflowX="clip"
22     >
23       <MainHeroSection />
24       <MainStatsBar />
25       <MainAboutSection />
26       <MainFeaturesSection />
27       <MainRolesSection />
28       <MainOrderStatusesSection orderStatusCounts={orderStatusCounts} />
29       <MainHowItWorksSection />
30       <MainCtaSection />
31       <MainContactSection />
32       <MainFooterSection />
33     </Box>
34   );
35 }

```

Рис. 3.4 Фрагмент реалізації головної сторінки

Головна сторінка розділена на окремі компоненти. «MainHeroSection» відповідає за перший екран, «MainAboutSection» за опис системи, «MainFeaturesSection» за переваги, «MainRolesSection» за ролі, «MainHowItWorksSection» за порядок початку роботи, «MainContactSection» за контакти.

Навігація з головної сторінки на сторінку входу реалізована у компоненті «TopBar».

Фрагмент коду наведено на рисунку 3.5.

```

    <Button
    >
      <NavLink to="/login" style={{ textDecoration: "none", color: "inherit" }}>
        Увійти
      </NavLink>
    </Button>
  </Flex>
</Box>
);
}

```

Рис. 3.5 Фрагмент реалізації навігації з головної сторінки на сторінку входу

У фрагменті реалізовано кнопку «Увійти», яка розміщена у верхній панелі головної сторінки та виконує роль основного переходу до сторінки авторизації. Для її створення використано компонент Button із бібліотеки Chakra UI, а навігаційний перехід реалізовано через NavLink на маршрут /login. Завдяки цьому елемент одночасно є частиною інтерфейсу та пов'язаний із маршрутизацією React Router. Користувач може перейти від публічної інформаційної частини вебзастосунку до сторінки входу без додаткових проміжних дій, що відповідає логіці головної сторінки як стартової точки роботи з LvStore.

Оформлення кнопки задано через властивості компонента: фон, колір тексту, рамку, заокруглення, внутрішні відступи, висоту, шрифт і стан при наведенні. Це дозволяє зробити кнопку візуально помітною та зберегти єдиний стиль верхньої панелі разом із логотипом і навігаційними пунктами. У компоненті TopBar також реалізовано переходи до секцій головної сторінки через навігаційні елементи, тому кнопка входу відокремлюється від інформаційної навігації і позначає перехід саме до захищеної робочої частини системи.

Такий підхід робить сценарій взаємодії зрозумілим: спочатку користувач ознайомлюється з можливостями LvStore, а потім переходить до авторизації для роботи у відповідному кабінеті.

Екранну форму головної сторінки вебзастосунку наведено на рисунку 3.6.

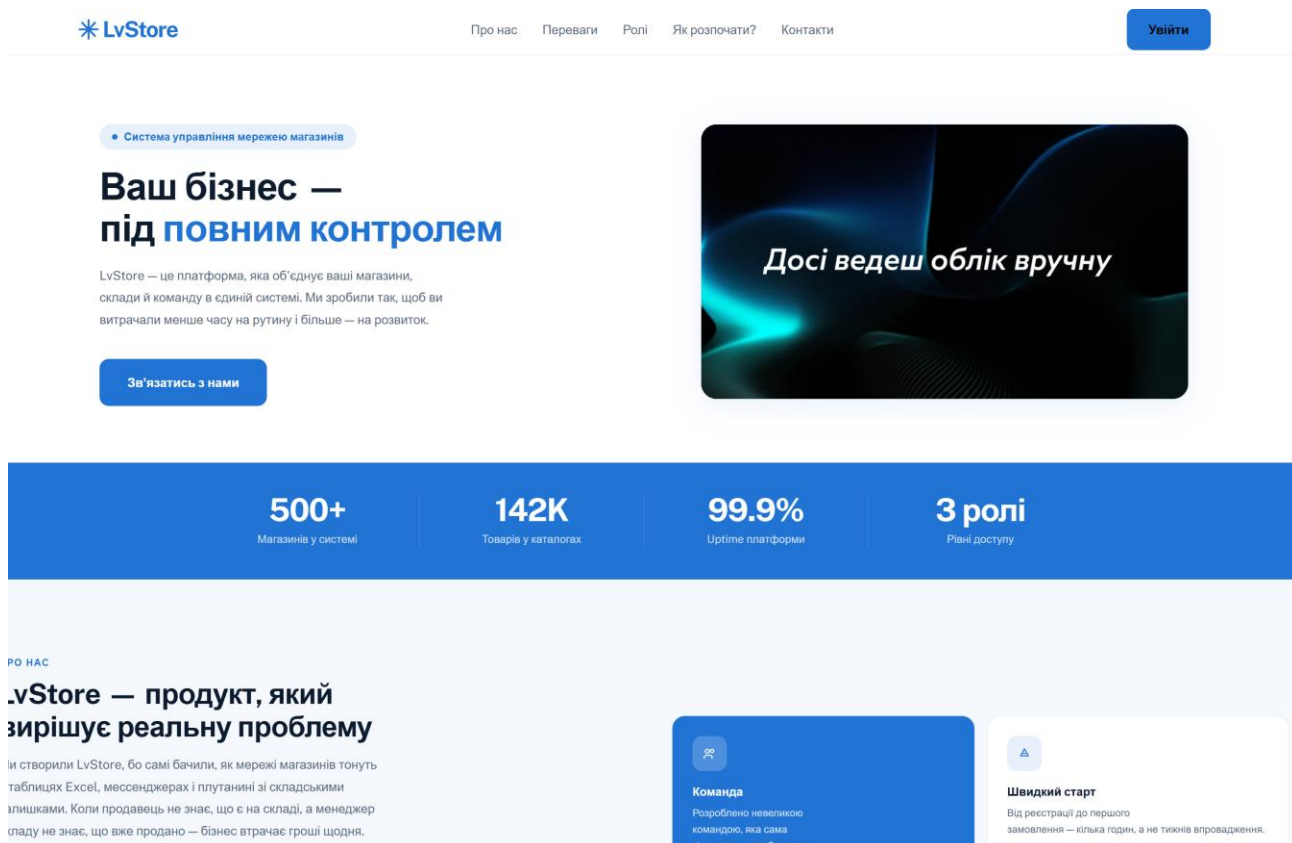


Рис. 3.6 Екранна форма головної сторінки вебзастосунку

На екранній формі видно верхню панель, логотип LvStore, навігаційні пункти та основний блок лендінгу. Перший екран пояснює призначення системи і показує, що застосунок орієнтований на об'єднання магазинів, складів і команди в єдиному інформаційному середовищі. Публічна частина не потребує авторизації, але містить перехід до сторінки авторизації для користувачів, які вже працюють у системі.

Головна сторінка побудована як послідовність інформаційних секцій. На ній подано опис системи, переваги LvStore, рольову модель користувачів, , порядок початку роботи та контактну форму. Така структура дозволяє не перевантажувати перший екран великою кількістю інформації, а поступово пояснювати користувачу, для чого призначений застосунок і які задачі він вирішує.

3.2.2 Реалізація сторінки авторизації користувача

Сторінка авторизації відповідає за вхід користувача до системи. Екранну форму сторінки авторизації наведено на рисунку 3.7.

Рис. 3.7 Екранна форма сторінки авторизації

Користувач заходить у систему через кнопку «Увійти». Після введення логіну та пароля відбувається перевірка даних, і в разі відповідності ролі та даних користувач отримує доступ до свого кабінету та всіх його функцій. Якщо дані некоректні або користувач не погодився з умовами використання, форма показує повідомлення про помилку. Також сторінка підтримує вхід через Google OAuth.

Сторінка авторизації реалізована як окремий компонент, який поєднує елементи інтерфейсу, локальний стан форми, перевірку введених даних і виклики API-шару. Для кожного поля передбачено окремий стан помилки, що дозволяє показувати користувачу конкретне повідомлення біля відповідного елемента форми. Такий підхід робить процес входу зрозумілим і зменшує кількість ситуацій, коли користувач не розуміє причину відмови в авторизації.

Фрагмент реалізації перевірки полів форми авторизації наведено на рисунку 3.8.

```
const validate = (): boolean => {  
  let ok = true;  
  if (!email.trim()) {  
    setEmailError("Вкажіть email");  
    ok = false;  
  } else {  
    setEmailError(null);  
  }  
  if (!password) {  
    setPasswordError("Вкажіть пароль");  
    ok = false;  
  } else {  
    setPasswordError(null);  
  }  
  if (!termsAccepted) {  
    setTermsError("Потрібно погодитися з умовами використання.");  
    ok = false;  
  } else {  
    setTermsError(null);  
  }  
  return ok;  
};
```

Рис. 3.8 Фрагмент реалізації перевірки полів форми авторизації

У наведеному фрагменті реалізовано функцію `validate`, яка виконує попередню перевірку введених користувачем даних перед відправленням форми. Функція повертає логічне значення, яке показує, чи може форма перейти до наступного етапу обробки. Такий підхід дозволяє не виконувати запит до серверної частини, якщо обов'язкові поля залишилися порожніми.

Перевірка пошти та пароля виконується через очищення рядка від зайвих пробілів за допомогою `trim`. Якщо поле не заповнене, у стан записується повідомлення вкажіть пошту та пароль. Якщо значення присутнє, помилка очищується, що дозволяє одразу оновити стан інтерфейсу після виправлення введених даних.

Фрагмент реалізації обробки входу користувача наведено на рисунку 3.9.

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setFormError(null);
  if (!validate()) {
    return;
  }
  setLoginBusy(true);
  try {
    await login({ email: email.trim(), password });
    const user = await getCurrentUser();
    navigate(getPostLoginPath(user.role), { replace: true });
  } catch (err) {
    setFormError(getLoginErrorMessage(err));
  } finally {
    setLoginBusy(false);
  }
};
```

Рис. 3.9 Фрагмент реалізації обробки входу користувача

У фрагменті реалізовано функцію `handleSubmit`, яка виконується після надсилання форми авторизації. На початку обробник скасовує стандартну поведінку браузера за допомогою `preventDefault`, щоб сторінка не перезавантажувалася після натискання кнопки входу. Далі очищується загальна помилка форми, яка могла залишитися після попередньої невдалої спроби авторизації.

Перед виконанням запиту до серверної частини викликається функція `validate`. Якщо хоча б одна умова перевірки не виконана, подальша обробка зупиняється. Це означає, що сервер отримує запит лише тоді, коли користувач заповнив `email`, пароль і погодився з умовами використання.

Після успішної локальної перевірки встановлюється стан `loginBusy`, який показує, що процес входу перебуває в активному стані. Далі викликається функція `login`, якій передаються `email` і пароль.

Email додатково обробляється через `trim`, щоб випадкові пробіли на початку або в кінці рядка не впливали на коректність авторизації.

Якщо авторизація проходить успішно, сторінка отримує дані поточного користувача через `getCurrentUser`. На основі ролі користувача визначається подальший маршрут за допомогою `getPostLoginPath`, після чого виконується перенаправлення до відповідного кабінету. У разі помилки викликається `getLoginErrorMessage`, а стан `loginBusy` очищується в блоці `finally`, що гарантує завершення індикатора завантаження незалежно від результату запиту.

Фрагмент реалізації входу через Google OAuth наведено на рисунку 3.10

```
const handleGoogleLogin = () => {  
  setFormError(null);  
  if (!termsAccepted) {  
    setTermsError("Потрібно погодитися з умовами використання.");  
    return;  
  }  
  window.location.assign(getGoogleOAuthAuthorizationUrl());  
};
```

Рис. 3.10 Фрагмент реалізації входу через Google OAuth

У наведеному фрагменті реалізовано функцію `handleGoogleLogin`, яка відповідає за альтернативний спосіб входу до системи через Google OAuth. На початку функція очищує загальну помилку форми, щоб попереднє повідомлення не залишалося на екрані під час нової спроби авторизації. Це забезпечує коректне оновлення стану інтерфейсу.

Перед переходом до зовнішнього механізму авторизації також перевіряється згода користувача з умовами використання. Якщо згода не підтверджена, у стан `termsError` записується відповідне повідомлення, а подальше виконання функції зупиняється. Завдяки цьому однакова вимога застосовується як для звичайного входу через email і пароль, так і для входу через Google.

Якщо користувач погодився з умовами, виконується перенаправлення браузера на адресу авторизації Google. Для формування саме цієї адреси використовується функція `getGoogleOAuthAuthorizationUrl`, яка отримує відповідний маршрут із API-шару. Такий підхід дозволяє не прописувати адресу OAuth безпосередньо в компоненті сторінки.

Перехід виконується через `window.location.assign`, оскільки авторизація через Google передбачає вихід за межі поточної сторінки та перехід до зовнішнього механізму перевірки облікового запису. Після завершення OAuth-процесу серверна частина повертає користувача до системи, де подальша логіка доступу вже залежить від отриманих даних користувача та його ролі.

3.2.3 Реалізація кабінету адміністратора магазину

Кабінет адміністратора магазину є робочою зоною для керування товарами, магазинами, користувачами, залишками і замовленнями.

Екранну форму кабінету адміністратора магазину наведено на рисунку 3.11.

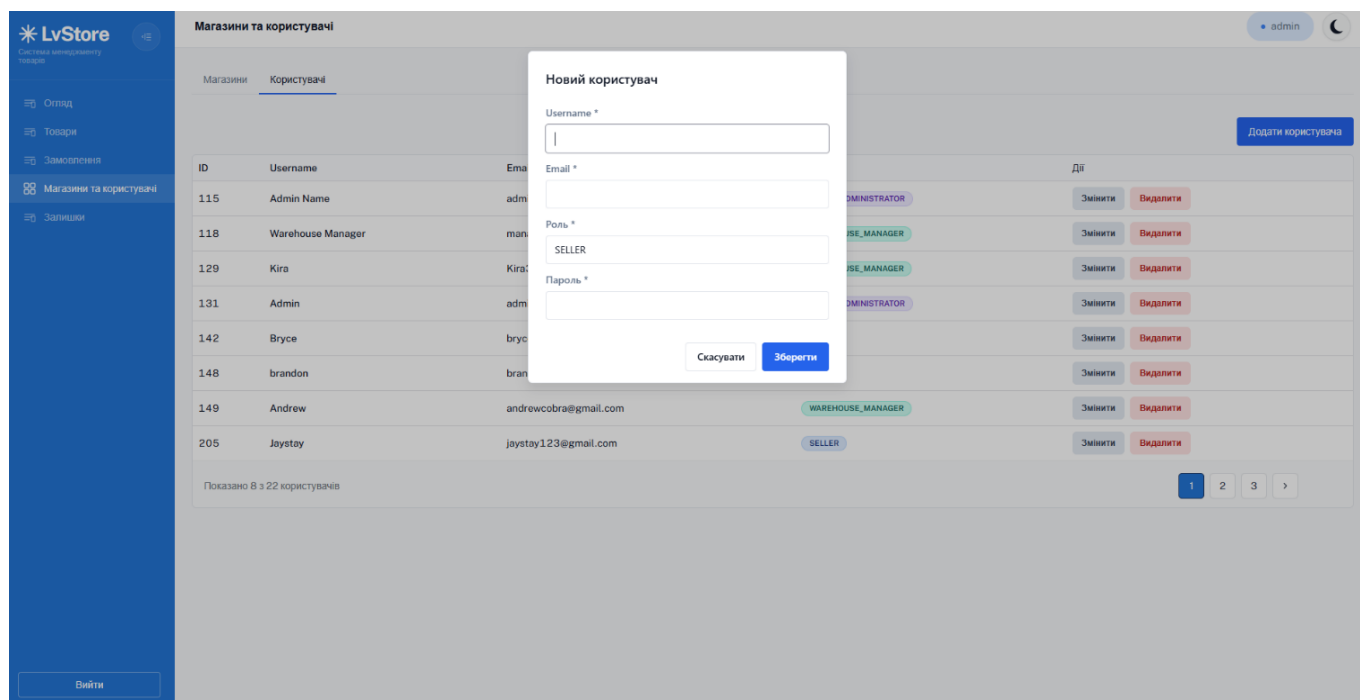


Рис. 3.11 Екранна форма кабінету адміністратора магазину

Адміністратор має найширший доступ серед ролей системи. Він може працювати з користувачами, переглядати залишки магазинів, створювати, підтверджувати, скасовувати і редагувати замовлення. У кабінеті використовується бокова навігація, таблиці, кнопки дій і модальні вікна.

Фрагмент реалізації створення нового користувача адміністратором наведено на рисунку 3.12.

```
const saveUserMut = useMutation({
  mutationFn: async () => {
    if (!uUsername.trim() || !uEmail.trim()) {
      throw new Error("Ім'я користувача та email обов'язкові.");
    }
    if (!editingUser && !uPassword.trim()) {
      throw new Error("Пароль обов'язковий при створенні.");
    }
    if (editingUser) {
      const body: UpdateUserPayload = {
        id: editingUser.id,
        username: uUsername.trim(),
        email: uEmail.trim(),
        role: uRole,
      };
      if (uPassword.trim() !== "") {
        body.password = uPassword.trim();
      }
      return updateUser(body);
    }
    const body: CreateUserPayload = {
      username: uUsername.trim(),
      email: uEmail.trim(),
      role: uRole,
      password: uPassword.trim(),
    };
    return createUser(body);
  },
  onSuccess: async () => {
    await qc.invalidateQueries({ queryKey: ["users"] });
    await qc.invalidateQueries({ queryKey: ["stores"] });
    setUserDialog(false);
  },
  onError: (e) => {
    setUserFormErr(
      e instanceof Error && !isAxiosError(e)
        ? e.message
        : getApiErrorMessage(e, "Не вдалося зберегти користувача."),
    );
  },
});
```

Рис. 3.12 Фрагмент реалізації створення нового користувача адміністратором

У AdminStoresUsersPage.tsx мутація saveUserMut перевіряє обов'язкові поля користувача. Якщо створюється новий користувач, пароль є обов'язковим. Якщо редагується існуючий користувач, пароль додається тільки тоді, коли адміністратор його ввів. Після успішного збереження інвалідуються запити users і stores, а діалогове вікно закривається.

Адміністраторський кабінет також містить інтерфейс для роботи з товарною номенклатурою. Через відповідну сторінку адміністратор може додавати нові товари, вказувати їх назву, категорію, штрихкод, артикул, ціну та опис. Такий підхід дозволяє централізовано наповнювати систему товарами, які надалі використовуються в замовленнях, залишках магазинів і складських операціях

Екранну форму створення нового товару в кабінеті адміністратора магазину наведено на рисунку 3.13.

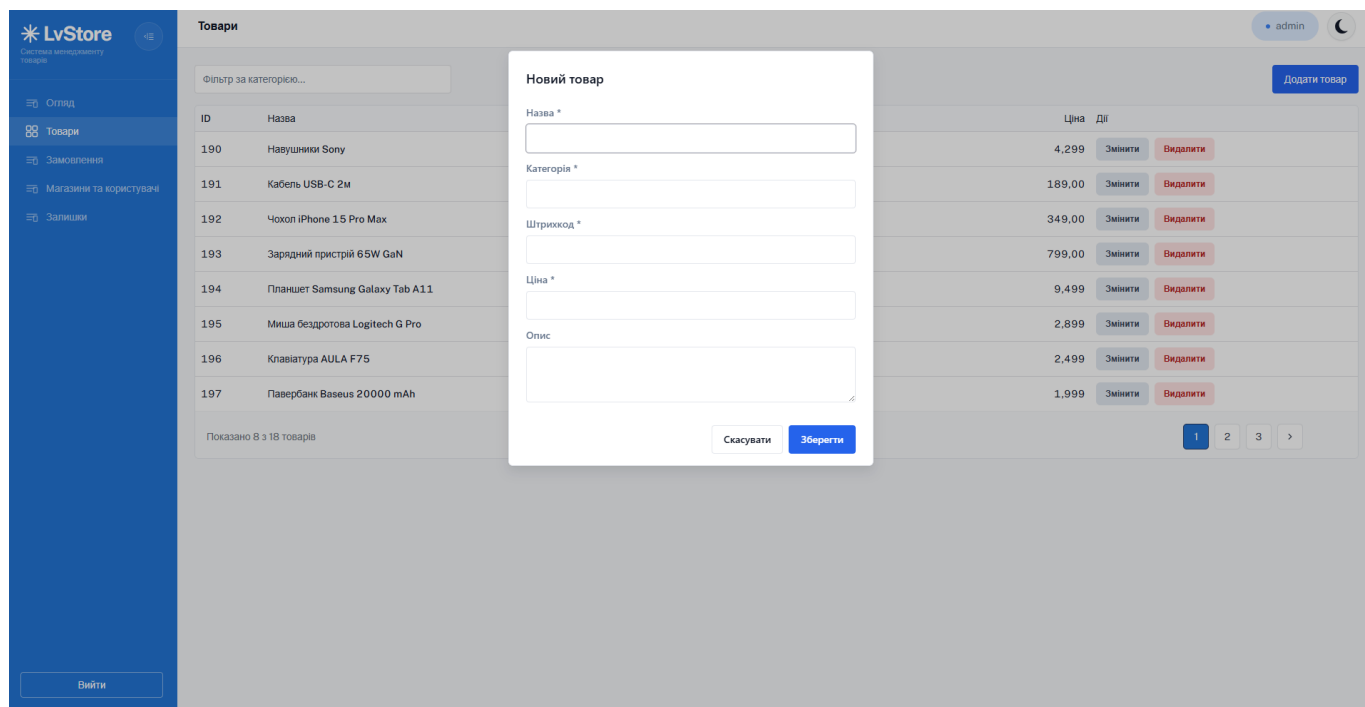


Рис. 3.13 Екранна форма створення нового товару адміністратором магазину

Фрагмент реалізації створення нового товару адміністратором наведено на рисунку 3.14.

```
const saveMut = useMutation({
  mutationFn: async () => {
    const price = Number(priceStr.replace(",", "."));
    if (!name.trim() || !category.trim() || !barcode.trim() || Number.isNaN(price)) {
      throw new Error("Заповніть обов'язкові поля коректно.");
    }
    if (editing) {
      const body: UpdateProductPayload = {
        id: editing.id,
        name: name.trim(),
        category: category.trim(),
        barcode: barcode.trim(),
        price,
        description: description.trim() || undefined,
      };
      return updateProduct(body);
    }
    const body: CreateProductPayload = {
      name: name.trim(),
      category: category.trim(),
      barcode: barcode.trim(),
      price,
      description: description.trim() || undefined,
    };
    return createProduct(body);
  },
  onSuccess: async () => {
    await qc.invalidateQueries({ queryKey: ["products"] });
    setDialogOpen(false);
  },
  onError: (e) => {
    setFormError(
      e instanceof Error && !isAxiosError(e)
        ? e.message
        : getApiErrorMessage(e, "Не вдалося зберегти товар."),
    );
  },
});
```

Рис. 3.14 Фрагмент реалізації створення нового товару адміністратором

У AdminProductsPage.tsx реалізовано логіку створення і редагування товару. Ціна перетворюється в числовий формат, а назва, категорія і штрихкод перевіряються на заповнення. Якщо користувач редагує товар, створюється UpdateProductPayload. Якщо створює новий товар, створюється CreateProductPayload. Після успішної мутації оновлюється список товарів.

3.2.4 Реалізація кабінету продавця

Кабінет продавця призначений для роботи з товарами, залишками магазину і замовленнями на поповнення.

Екранну форму кабінету продавця наведено на рисунку 3.15.

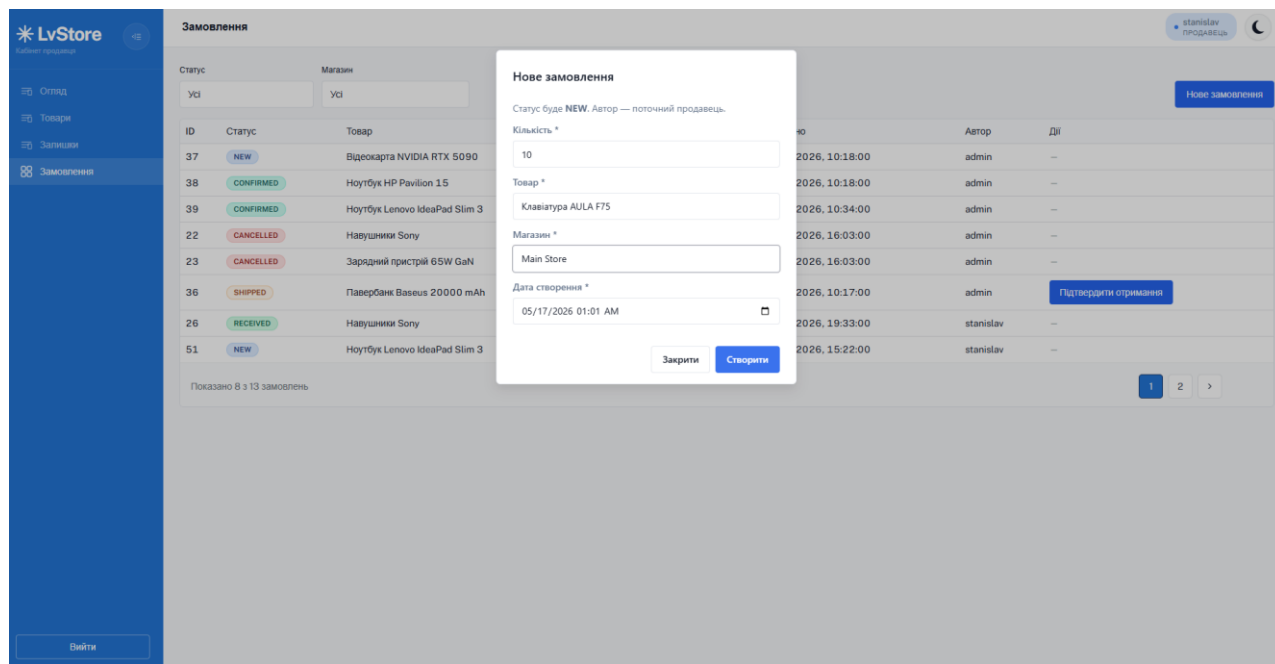


Рис. 3.15 Екранна форма кабінету продавця

На екранній формі реалізовано роботу продавця з замовленнями. У межах кабінету продавець має обмежений, але достатній набір функцій для щоденної роботи торгової точки: він може переглядати товарну номенклатуру, контролювати залишки магазину, переглядати список замовлень, створювати нове замовлення та позначати отримання товару після його виконання. Під час створення замовлення відкривається модальне вікно, у якому вказуються кількість, товар, магазин і дата створення.

Для роботи зі списками використано таблиці, фільтри, пагінацію та кнопки дій, що дозволяє швидко знаходити потрібний запис без переходу до адміністративних функцій. У цьому кабінеті відсутні сторінки керування користувачами, магазинами або складами, оскільки такі дії не належать до зони відповідальності продавця.

Фрагмент реалізації створення нового замовлення продавцем наведено на рисунку 3.16.

```
const onCreate = () => {
  const me = meQ.data;
  if (!me) {
    setFormError("Не вдалося отримати дані користувача.");
    return;
  }
  const qty = Number(cQuantity);
  const pid = Number(cProductId);
  const sid = Number(cStoreId);
  if (!Number.isFinite(qty) || qty < 1) {
    setFormError("Вкажіть коректну кількість.");
    return;
  }
  if (!Number.isFinite(pid) || !Number.isFinite(sid)) {
    setFormError("Оберіть товар і магазин.");
    return;
  }
  const createdAt = `${cCreatedAt}:00`;
  setFormError(null);
  createMut.mutate({
    status: OrderStatus.NEW,
    quantity: qty,
    createdAt,
    productId: pid,
    storeId: sid,
    creatorId: me.id,
  });
};
```

Рис. 3.16 Фрагмент реалізації створення нового замовлення продавцем

У SellerOrdersPage.tsx функція onCreate спочатку перевіряє наявність даних поточного користувача. Далі вона перетворює кількість, ідентифікатор товару і ідентифікатор магазину в числа. Якщо кількість некоректна або товар і магазин не обрані, у формі показується помилка. Якщо дані правильні, створюється замовлення зі статусом NEW, кількістю, датою, товаром, магазином і автором.

3.2.5 Реалізація кабінету менеджера складу

Кабінет менеджера складу призначений для роботи зі складами, складськими залишками, товарами і замовленнями. Екранну форму кабінету менеджера складу наведено на рисунку 3.14.

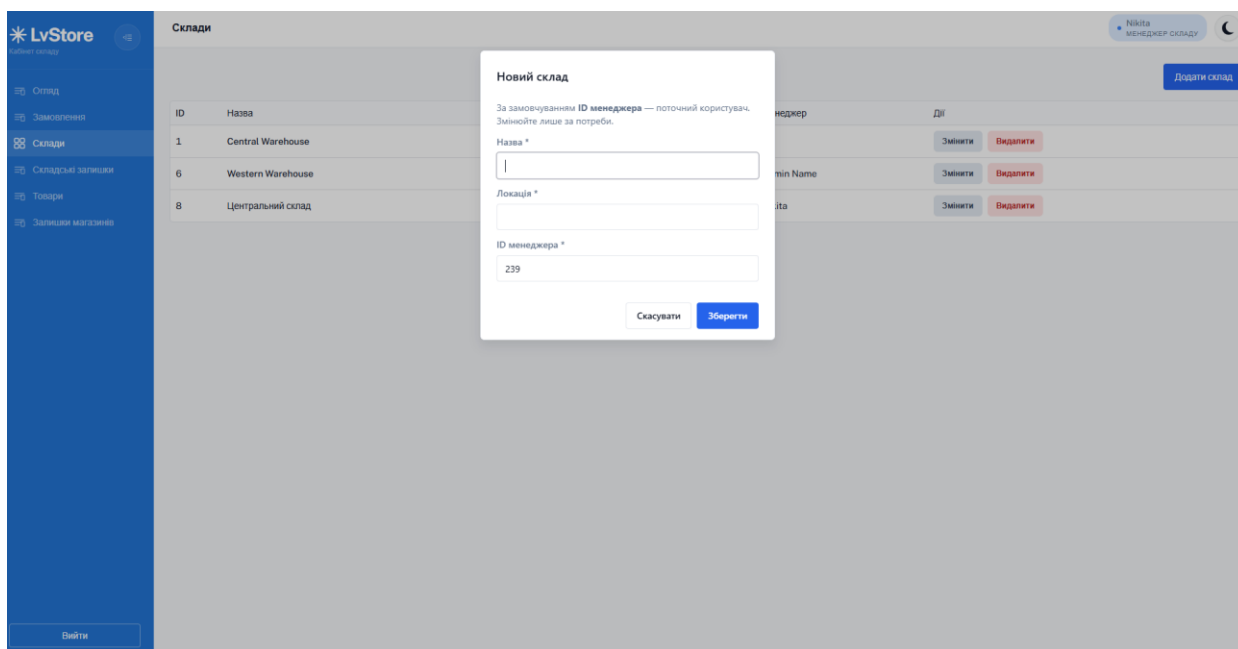


Рис. 3.17 Екранна форма кабінету менеджера складу

На екранній формі реалізовано робочу область менеджера складу, зокрема сценарій створення нового складу в системі. У модальному вікні користувач вводить назву складу, його локацію та ідентифікатор відповідального менеджера, після чого ці дані використовуються для формування нового складського об'єкта. Такий функціонал належить саме до складської частини LvStore, оскільки менеджер складу відповідає за підтримку структури складів, контроль складських залишків і подальшу обробку замовлень, що надходять від магазинів. На відміну від продавця, ця роль не створює заявки на поповнення, а працює з їх виконанням, відвантаженням і контролем доступних товарів на складі. У кабінеті також передбачено доступ до списку замовлень, товарної номенклатури, складських залишків і залишків магазинів, що дозволяє оцінювати наявність товарів перед виконанням заявки.

Така організація інтерфейсу забезпечує менеджеру складу окреме робоче середовище, орієнтоване не на продажі чи адміністрування користувачів, а на підтримку складської взаємодії між складом і торговими точками.

Фрагмент реалізації створення нового складу менеджером складу наведено на рисунку 3.18.

```
const saveMut = useMutation({
  mutationFn: async () => {
    const managerId = Number(managerIdStr);
    if (!name.trim() || !location.trim() || Number.isNaN(managerId)) {
      throw new Error("Заповніть назву, локацію та коректний ID менеджера.");
    }
    if (editing) {
      return updateWarehouse({
        id: editing.id,
        name: name.trim(),
        location: location.trim(),
        managerId,
      });
    }
    return createWarehouse({
      name: name.trim(),
      location: location.trim(),
      managerId,
    });
  },
  onSuccess: async () => {
    await qc.invalidateQueries({ queryKey: ["warehouses"] });
    setDialogOpen(false);
  },
  onError: (e) => {
    setFormError(
      e instanceof Error && !isAxiosError(e)
        ? e.message
        : getApiErrorMessage(e, "Не вдалося зберегти склад."),
    );
  },
});
```

Рис. 3.18 Фрагмент реалізації створення нового складу менеджером складу

У WarehouseWarehousesPage.tsx мутація saveMut отримує managerId із форми, перевіряє назву, локацію і коректність ідентифікатора менеджера. Якщо редагується існуючий склад, викликається updateWarehouse. Якщо створюється новий склад, викликається createWarehouse. Після успішного збереження інвалідується queryKey warehouses і закривається діалогове вікно.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ВЕБЗАСТОСУНКУ

Тестування Frontend-частини виконувалося для перевірки коректності авторизації, рольового доступу, основних сторінок, форм, модальних вікон, обробки помилок і адаптивності інтерфейсу. У роботі поєднано ручне функціональне тестування та перевірку відображення на різних розмірах екрана. Такий підхід дозволив оцінити не лише працездатність окремих елементів, а й цілісність основних сценаріїв взаємодії користувача із системою.

Для LvStore особливе значення має перевірка форм. Адміністратор створює користувачів і товари, продавець створює замовлення, менеджер складу створює склади. Якщо форма приймає некоректні дані або не показує помилку, це порушує робочий процес. Тому тестування виконувалося як для правильних, так і для помилкових сценаріїв.

4.1 Кросбраузерне тестування

Кросбраузерне тестування є важливим етапом перевірки вебзастосунку, оскільки користувачі можуть відкривати систему в різних браузерах. Навіть якщо інтерфейс побудований на однаковому коді, окремі елементи можуть відображатися по-різному через особливості обробки стилів, таблиць, форм, шрифтів або адаптивної верстки. Для LvStore така перевірка потрібна, оскільки користувачі працюють із таблицями товарів, залишків, замовлень і формами створення записів [8, 11, 27].

Під час кросбраузерного тестування перевірялося відображення основних сторінок застосунку. Основною метою було встановити, чи зберігається однакова логіка роботи інтерфейсу незалежно від браузера.

Відображення вебзастосунку в Google Chrome наведено на рисунку 4.1.

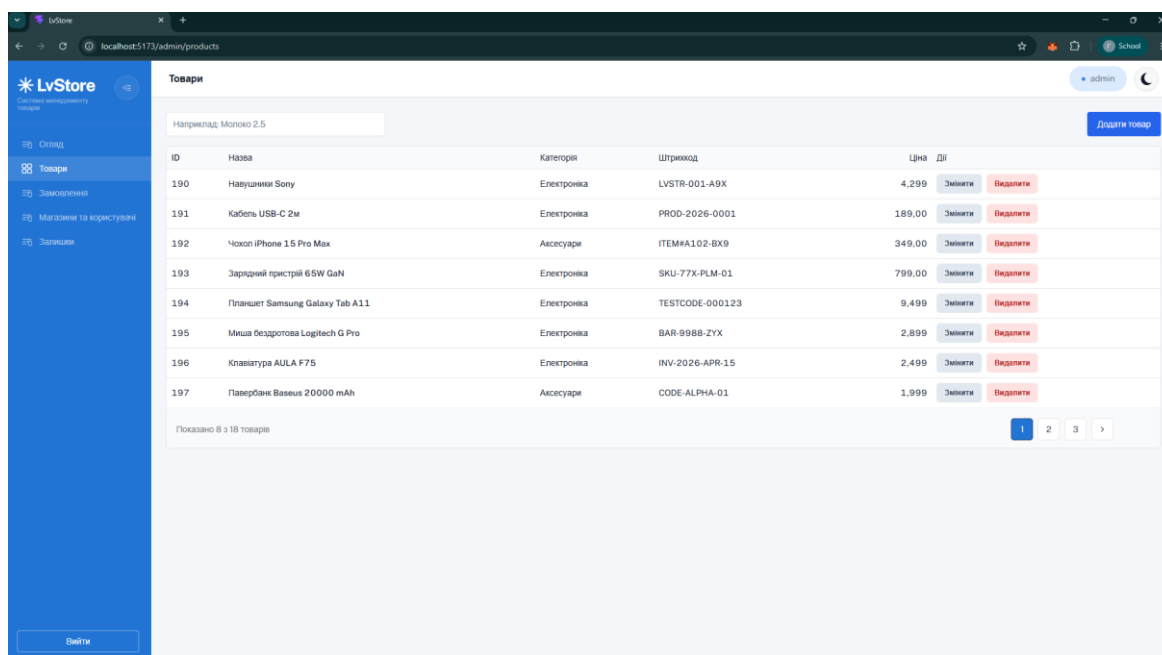


Рис. 4.1 Відображення вебзастосунку в браузері Google Chrome

Відображення вебзастосунку в Mozilla Firefox наведено на рисунку 4.2.

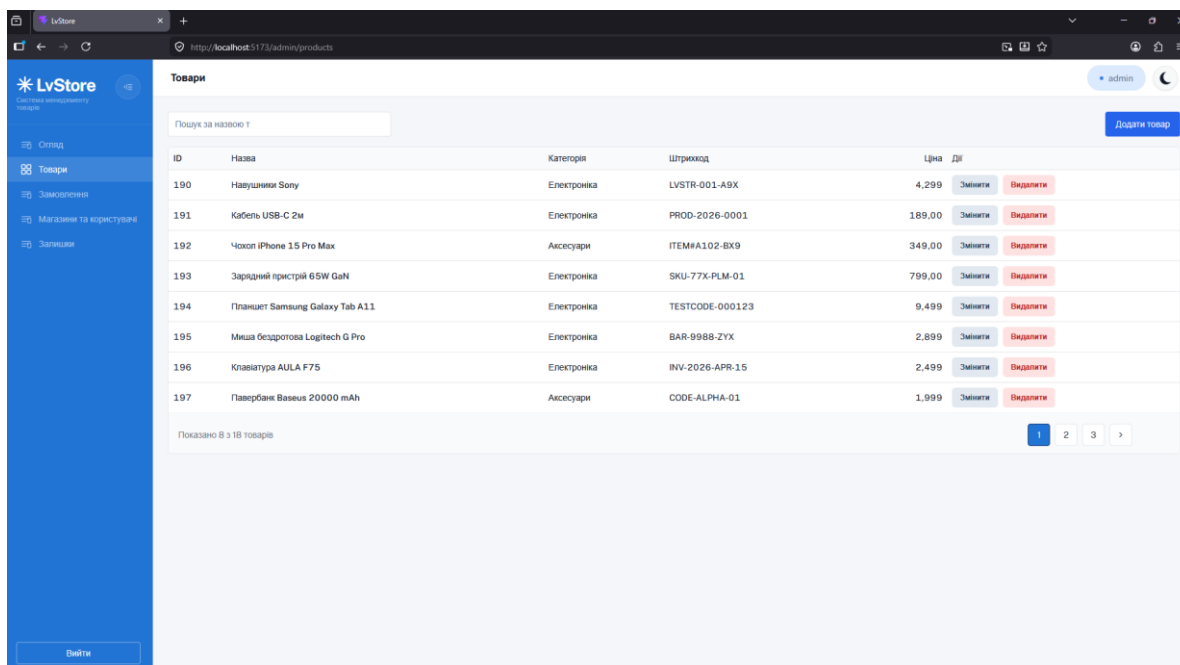


Рис. 4.2 Відображення вебзастосунку в браузері Mozilla Firefox

Відображення вебзастосунку в браузері Microsoft Edge наведено на рисунку 4.3.

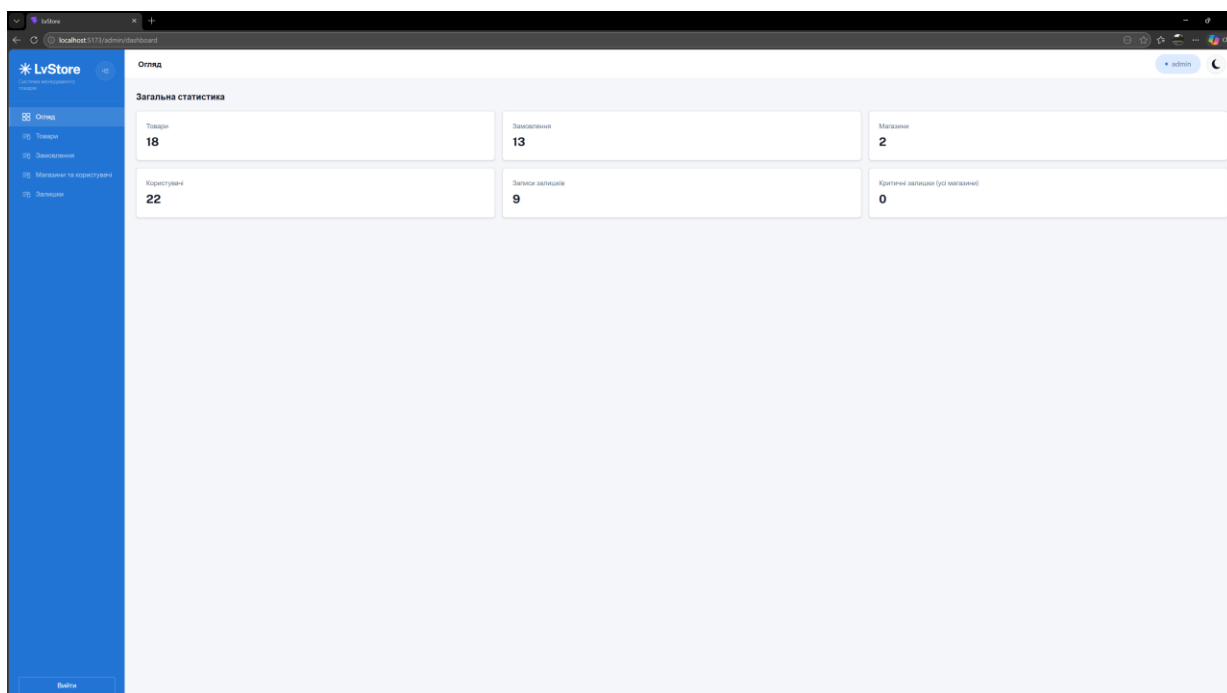


Рис. 4.3 Відображення вебзастосунку в браузері Microsoft Edge

4.2 Тестування адаптивності під різні пристрої

Адаптивність перевірялася у браузері за допомогою інструментів розробника. Доцільно зафіксувати три стани: десктопну версію, планшетну версію і мобільну версію. Для LvStore основний робочий сценарій орієнтований на десктоп і планшет, оскільки сторінки містять таблиці товарів, залишків і замовлень. Мобільна версія перевіряється для коректного відображення основної навігації, форм входу і базових списків.

Під час тестування адаптивності оцінювалося не лише зменшення ширини сторінки, а й збереження зручності роботи з елементами інтерфейсу. Це важливо, оскільки користувачі системи можуть відкривати застосунок як на робочому комп'ютері, так і на планшеті або меншому екрані.

Десктопну версію застосунку наведено на рисунку 4.4.

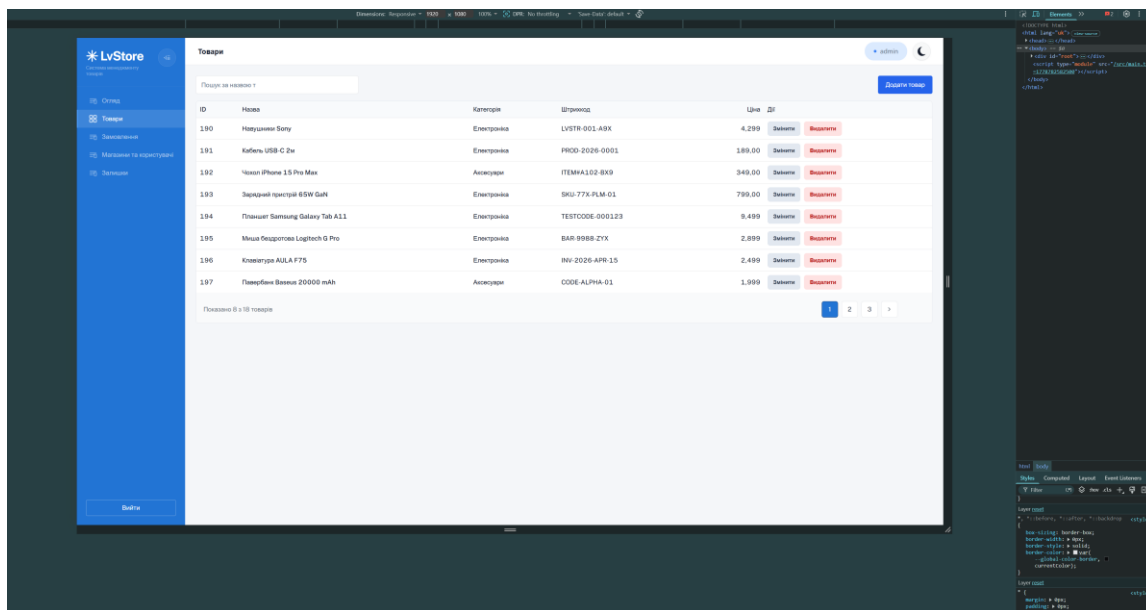


Рис. 4.4 Десктопна версія застосунку

Десктопна версія перевірялася на ширині приблизно 1440 px і 1920 px. У цьому режимі таблиці мають достатньо місця для колонок, бокова навігація залишається відкритою, а верхня панель не перекриває робочу область. Кнопки дій, пагінація і модальні вікна залишаються доступними.

Планшетну версію застосунку наведено на рисунку 4.5.

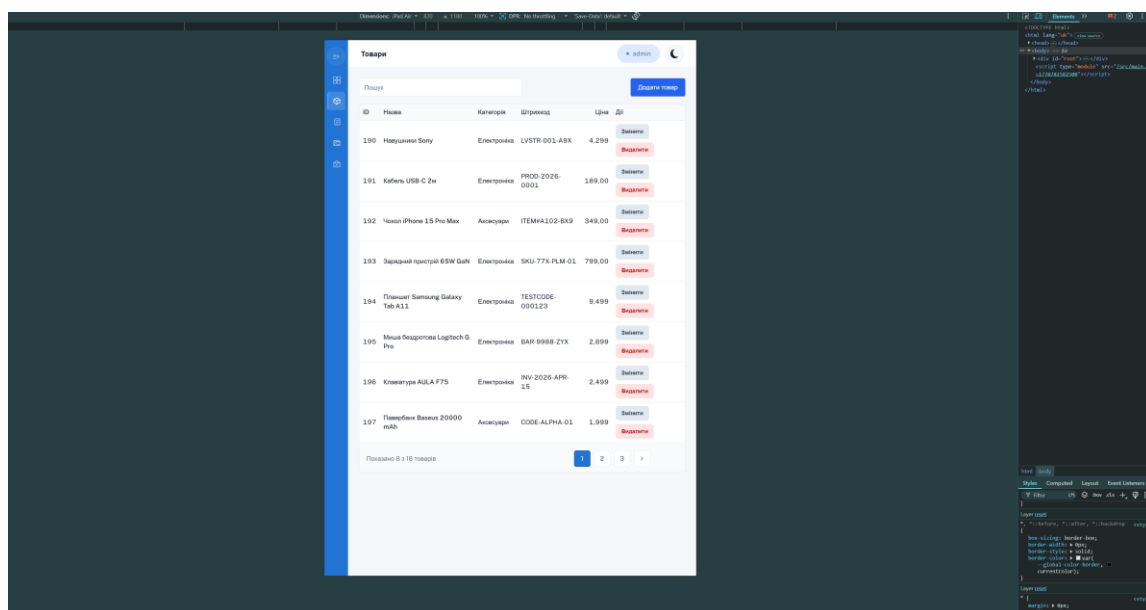


Рис. 4.5 Планшетна версія застосунку

Планшетна версія перевірялася для ширини приблизно 768 px і 1024 px. У цьому режимі зменшується доступна ширина, тому перевірялися таблиці, навігація, кнопки і модальні вікна.

Мобільну версію застосунку наведено на рисунку 4.6.

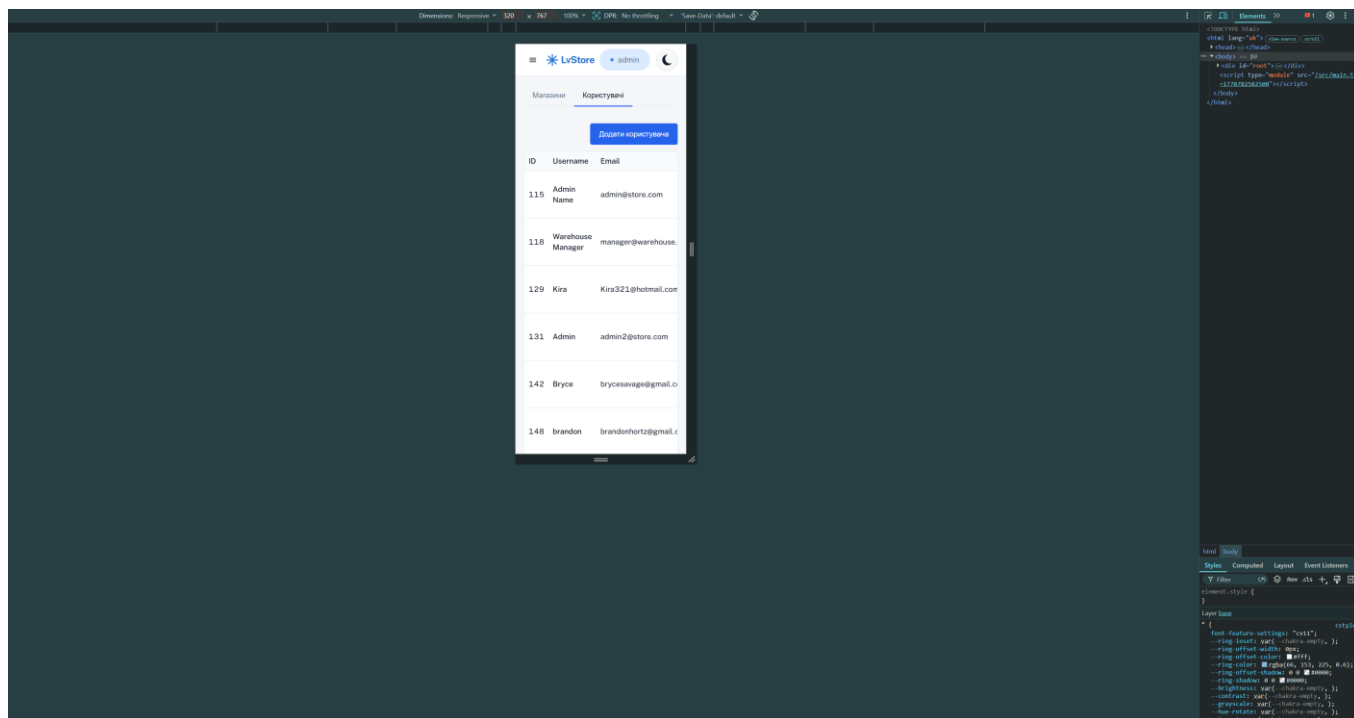


Рис. 4.6 Мобільна версія застосунку

Мобільна версія перевірялася для ширини близько 320 px і 430 px. Для LvStore мобільний режим не є основним для роботи з великими таблицями, але він повинен коректно відображати головну сторінку, сторінку входу і базову навігацію.

4.3 Функціональне тестування

Функціональне тестування проводилося для основних сценаріїв входу, створення записів, відкриття модальних вікон і перевірки обов'язкових полів. Тест-кейси для функціонального тестування наведено в таблиці 4.1.

Тест-кейси для функціонального тестування ОК/NOK

№	Сценарій	Кроки	Очікуваний результат	Статус
1	Вхід з порожніми полями	Відкрити сторінку входу, натиснути кнопку входу без введення даних.	Показано повідомлення про необхідність вказати email і пароль.	ОК
2	Вхід з неправильними даними	Ввести неправильний email або пароль, підтвердити умови, натиснути кнопку входу.	Показано повідомлення про неправильний email або пароль.	ОК
3	Вхід через Google без прийняття умов	Натиснути кнопку Google OAuth без погодження з умовами.	Система не починає OAuth і показує повідомлення про необхідність погодження.	ОК
4	Видалення товару адміністратором	У кабінеті адміністратора натиснути видалення товару.	Відкривається ConfirmDialog, дія виконується лише після підтвердження.	ОК
5	Скасування замовлення менеджером складу	У кабінеті складу вибрати замовлення і натиснути скасування.	Відкривається підтвердження, після підтвердження статус змінюється або показується помилка API.	ОК
6	Створення товару без обов'язкових полів	Відкрити форму створення товару і залишити назву, категорію або штрихкод порожніми.	Форма не зберігається, користувач бачить повідомлення про некоректне заповнення.	ОК

Під час тестування сторінки авторизації перевірялося, що система не дозволяє виконати вхід без обов'язкових даних.

Приклад некоректного заповнення полів логіну наведено на рисунку 4.7.

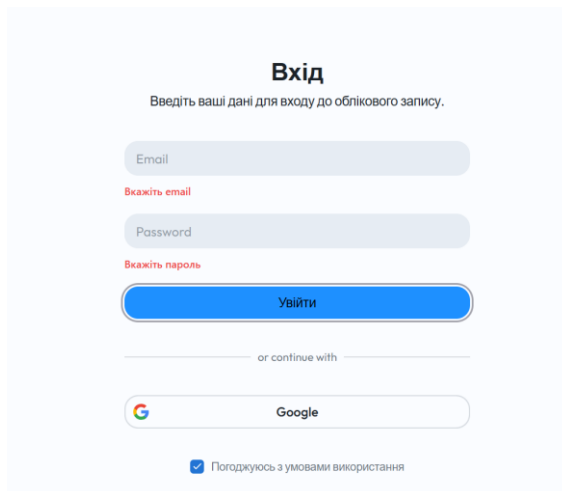


Рис. 4.7 Приклад некоректного заповнення полів логіну

На цьому рисунку перевіряється NOK-сценарій. Користувач не заповнює обов'язкові поля або вводить дані некоректно. Система повинна зупинити відправлення форми і показати повідомлення про помилки. Така перевірка підтверджує, що валідація виконується до запиту на сервер.

Приклад правильного заповнення форми наведено на рисунку 4.8.

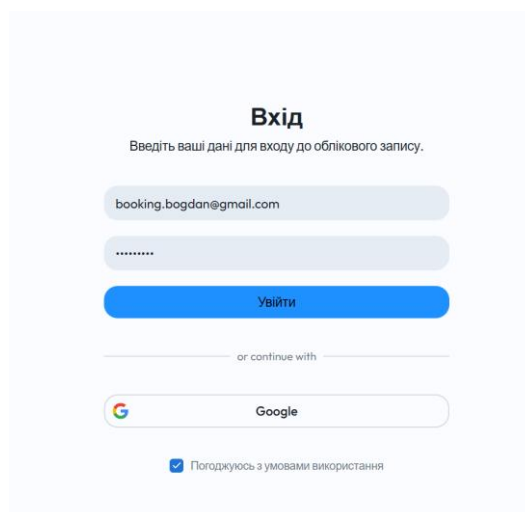


Рис. 4.8 Приклад правильного заповнення форми

На цьому рисунку перевіряється ОК-сценарій. Користувач заповнює email, пароль і погоджується з умовами використання. Після натискання кнопки входу система повинна виконати авторизацію, отримати дані користувача і перейти до кабінету за роллю.

Крім сторінки входу перевірялися форми створення товару, користувача, замовлення і складу. У кожному випадку тестувалося, що порожні або некоректні поля не зберігаються, а правильні дані викликають відповідну дію і оновлюють таблицю.

Під час тестування видалення товару адміністратором перевірялося, що система не виконує критичну дію одразу після натискання кнопки видалення. Перед остаточним видаленням запису має відкриватися модальне вікно підтвердження, яке дає користувачу можливість скасувати дію або підтвердити її виконання.

Приклад відкриття вікна підтвердження видалення товару наведено на рисунку 4.9.

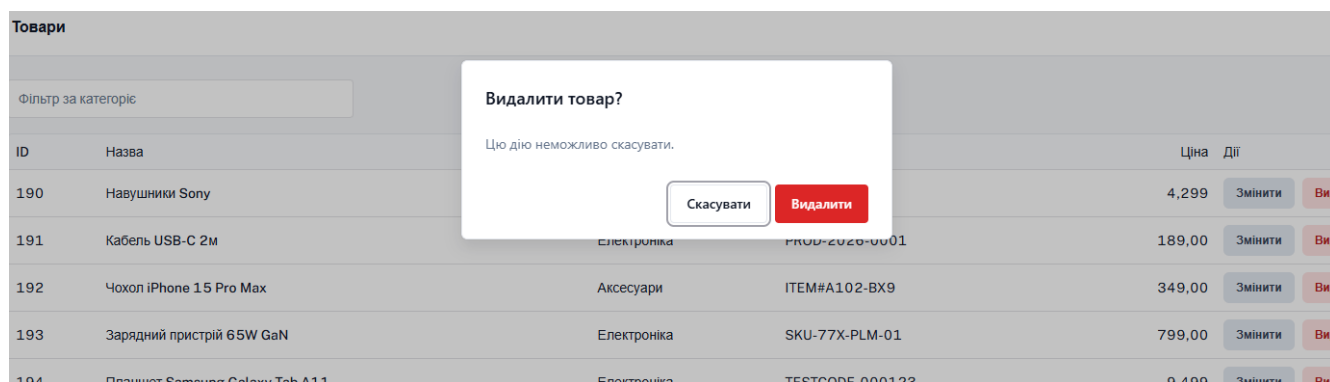


Рис. 4.9 Приклад підтвердження видалення товару адміністратором

На цьому рисунку перевіряється ОК-сценарій. Адміністратор натискає кнопку видалення товару, після чого система відкриває ConfirmDialog. Видалення не виконується автоматично, а відбувається лише після підтвердження дії користувачем. Така перевірка підтверджує, що у системі передбачено захист від випадкового видалення важливих записів.

Під час тестування скасування замовлення менеджером складу перевірялося, що зміна статусу замовлення виконується лише після додаткового підтвердження. Це потрібно для того, щоб користувач не міг випадково скасувати замовлення під час перегляду списку або роботи з таблицею.

Приклад підтвердження скасування замовлення наведено на рисунку 4.10.

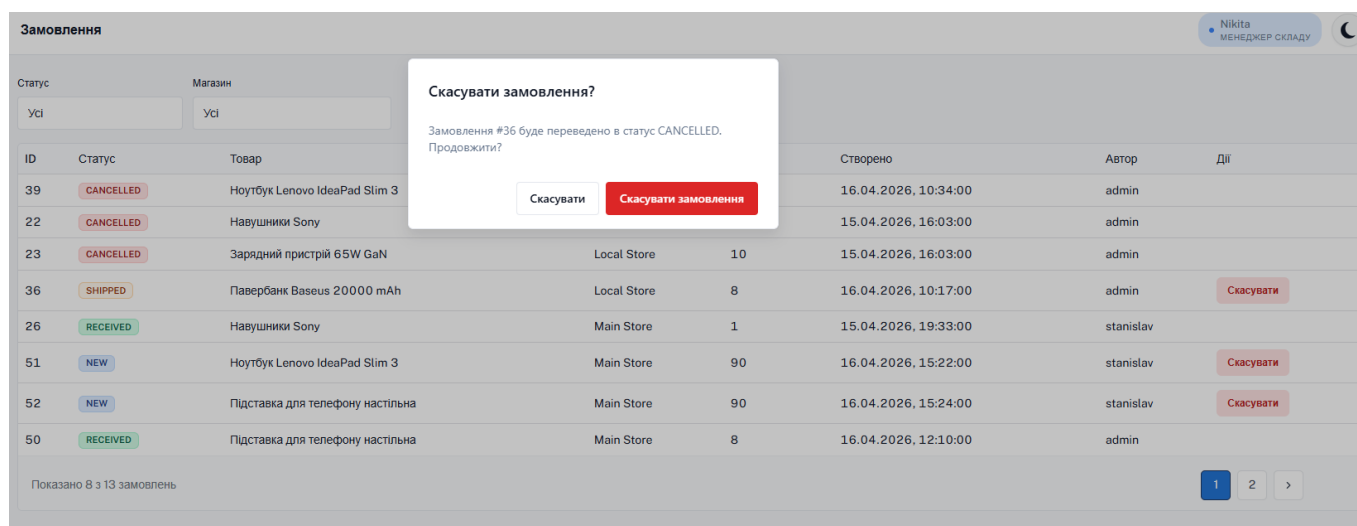


Рис. 4.10 Приклад підтвердження скасування замовлення менеджером складу

На цьому рисунку перевіряється ОК-сценарій. Менеджер складу вибирає замовлення та натискає дію скасування. Система відкриває вікно підтвердження, після чого дія виконується лише у разі згоди користувача. Якщо серверна частина повертає помилку, користувач повинен отримати відповідне повідомлення. Така перевірка підтверджує коректність роботи з критичними діями над замовленнями.

Окреме значення має те, що замовлення пов'язане з процесом поповнення товарів у магазині. Неправильне скасування може порушити логіку обробки заявки між магазином і складом. Тому тестування цього сценарію підтверджує, що система захищає користувача від випадкової зміни статусу та залишає можливість скасувати дію до її остаточного виконання.

Під час тестування створення товару перевірялося, що форма не дозволяє зберегти запис без заповнення обов'язкових полів. До таких полів належать назва товару, категорія, штрихкод та інші дані, необхідні для подальшої роботи з товарною номенклатурою.

Приклад створення товару без обов'язкових полів наведено на рисунку 4.11.

Товари

Пошук за назвою товару...

ID Назва

190	Навушники Sony	4,299	Змінити	Видалити
191	Кабель USB-C 2м	189,00	Змінити	Видалити
192	Чохол iPhone 15 Pro Max	349,00	Змінити	Видалити
193	Зарядний пристрій 65W GaN	799,00	Змінити	Видалити
194	Планшет Samsung Galaxy Tab A11	9,499	Змінити	Видалити
195	Миша бездротова Logitech G Pro	2,899	Змінити	Видалити
196	Клавіатура AULA F75	2,499	Змінити	Видалити
197	Павербанк Baseus 20000 mAh	1,999	Змінити	Видалити

Показано 8 з 18 товарів

Додати товар

Новий товар

Заповніть обов'язкові поля коректно.

Назва *

Ноутбук Acer Nitro 5

Категорія *

Штрихкод *

Ціна *

Опис

Скасувати Зберегти

Рис. 4.11 Приклад створення товару без обов'язкових полів

На цьому рисунку перевіряється NOK-сценарій. Адміністратор відкриває форму створення товару, але залишає обов'язкові поля порожніми або вводить некоректні значення. Система не повинна зберігати такий запис і має повідомити користувача про помилку заповнення. Така перевірка підтверджує, що дані проходять валідацію до створення нового товару в системі.

За результатами функціонального тестування встановлено, що основні сценарії роботи Frontend-частини виконуються коректно. Система реагує на помилкові дії користувача, блокує відправлення некоректних форм, відкриває модальні вікна підтвердження для критичних операцій і оновлює дані після успішного виконання дій. Це підтверджує працездатність основних модулів вебзастосунку LvStore.

4.4 Нефункціональне тестування продуктивності

Для додаткової оцінки якості Frontend-частини вебзастосунку було проведено нефункціональне тестування за допомогою інструмента Google Lighthouse. Перевірка виконувалася у браузері Google Chrome через інструменти розробника. Метою цього тестування було оцінити продуктивність, доступність інтерфейсу та дотримання базових практик розробки для основних сторінок LvStore.

Під час тестування було перевірено головну сторінку, сторінку авторизації, сторінку адміністратора, сторінку продавця та сторінку менеджера складу. Для кожної сторінки фіксувалися показники Performance, Accessibility і Best Practices, а також значення First Contentful Paint та Largest Contentful Paint. Це дозволило оцінити не тільки коректність роботи сторінок, а й швидкість їх первинного відображення для користувача.

Аудит головної сторінки вебзастосунку було виконано для перевірки публічної частини LvStore. Ця сторінка відкривається завантаження, коректне відображення елементів і доступність навігації.

Результати аудиту головної сторінки наведено на рисунку 4.12.

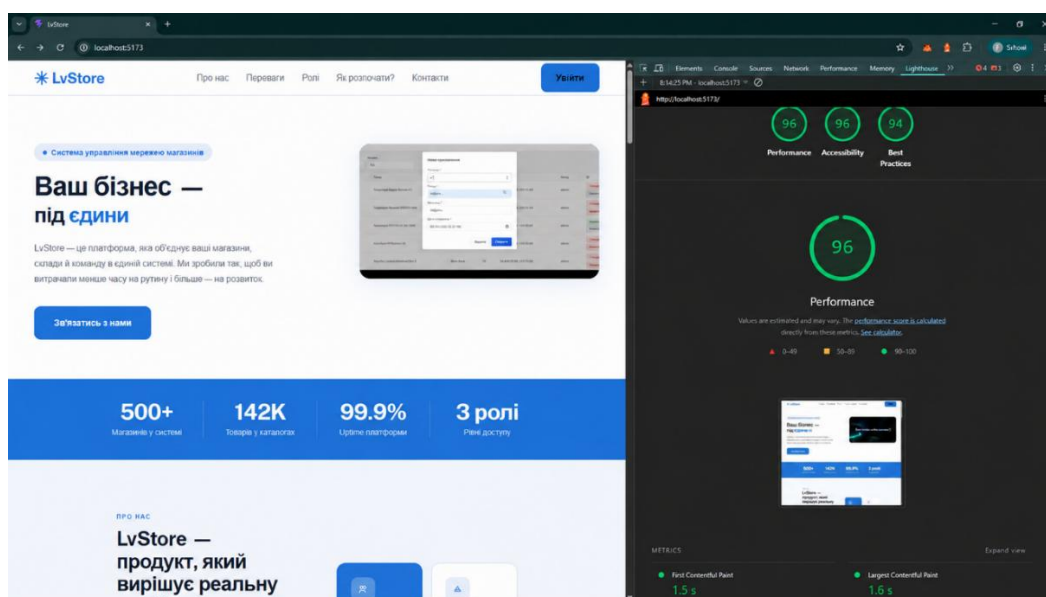


Рис. 4.12 Результати аудиту головної сторінки у Google Lighthouse

За результатами перевірки головна сторінка отримала показник Performance 96, Accessibility 96 і Best Practices 94. Такі значення свідчать про достатньо високу якість реалізації публічної частини вебзастосунку. Показник First Contentful Paint становив 1,5 с, а Largest Contentful Paint — 1,6 с, що підтверджує швидке відображення основного вмісту сторінки.

Отримані результати є важливими, оскільки головна сторінка є першою точкою взаємодії користувача із системою. Високий показник продуктивності означає, що інформаційні секції, навігація та кнопка переходу до авторизації завантажуються без суттєвої затримки. Показники Accessibility і Best Practices підтверджують, що сторінка загалом відповідає базовим вимогам зручності та якості вебінтерфейсу.

Аудит сторінки авторизації проводився для перевірки швидкості та якості відображення форми входу. Для цієї сторінки важливими є швидке завантаження, доступність полів введення, кнопки входу, Google OAuth та елементів перевірки умов використання.

Результати аудиту сторінки авторизації наведено на рисунку 4.13.

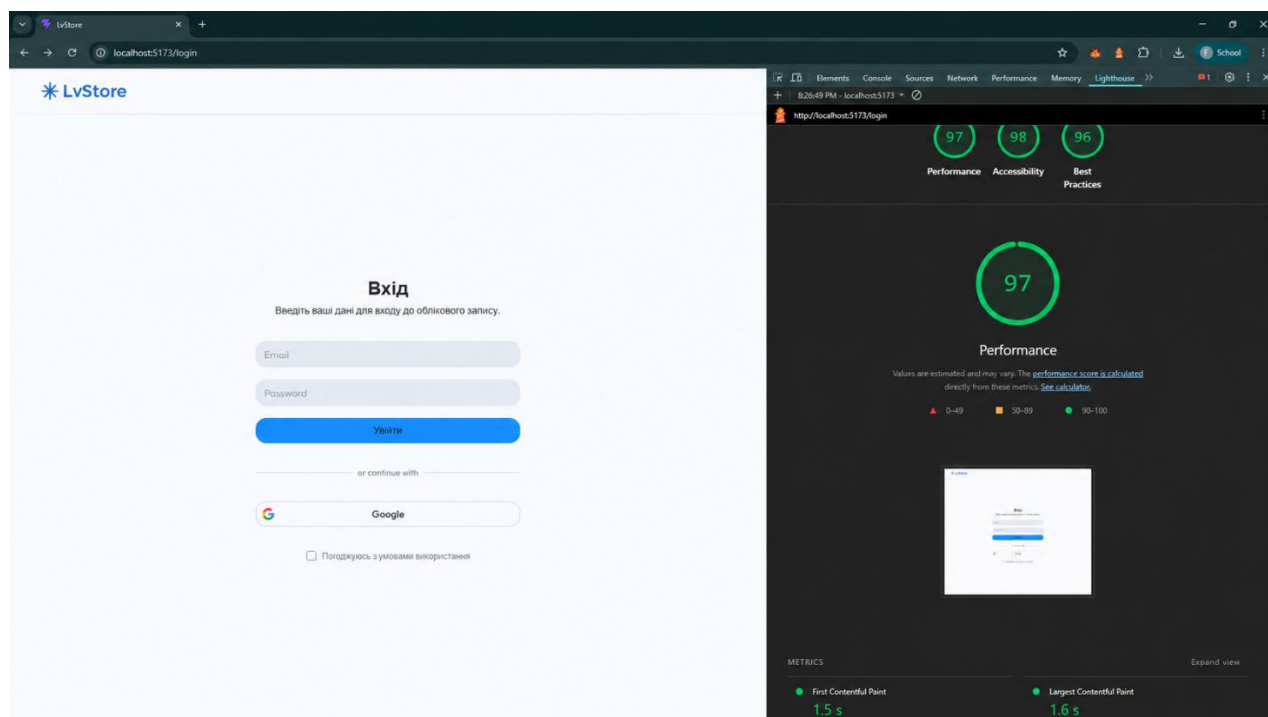


Рис. 4.13 Результати аудиту сторінки авторизації у Google Lighthouse

Сторінка авторизації отримала показник Performance 97, Accessibility 98 і Best Practices 96. Це є найвищим результатом серед перевірених сторінок, що пояснюється меншою кількістю складних елементів інтерфейсу порівняно з рольовими кабінетами. First Contentful Paint становив 1,5 с, а Largest Contentful Paint — 1,6 с.

Такі результати підтверджують, що сторінка входу не створює зайвого навантаження під час відкриття і швидко надає користувачу доступ до форми авторизації. Високий показник Accessibility є важливим для цієї сторінки, оскільки користувач повинен без труднощів взаємодіяти з полями введення, кнопками та повідомленнями про помилки.

Аудит сторінки адміністратора було проведено на прикладі сторінки керування товарами. Ця сторінка містить таблицю, пошук, кнопки редагування, видалення та створення нового товару, тому вона є більш навантаженою за кількістю UI-елементів.

Результати аудиту сторінки адміністратора наведено на рисунку 4.14.

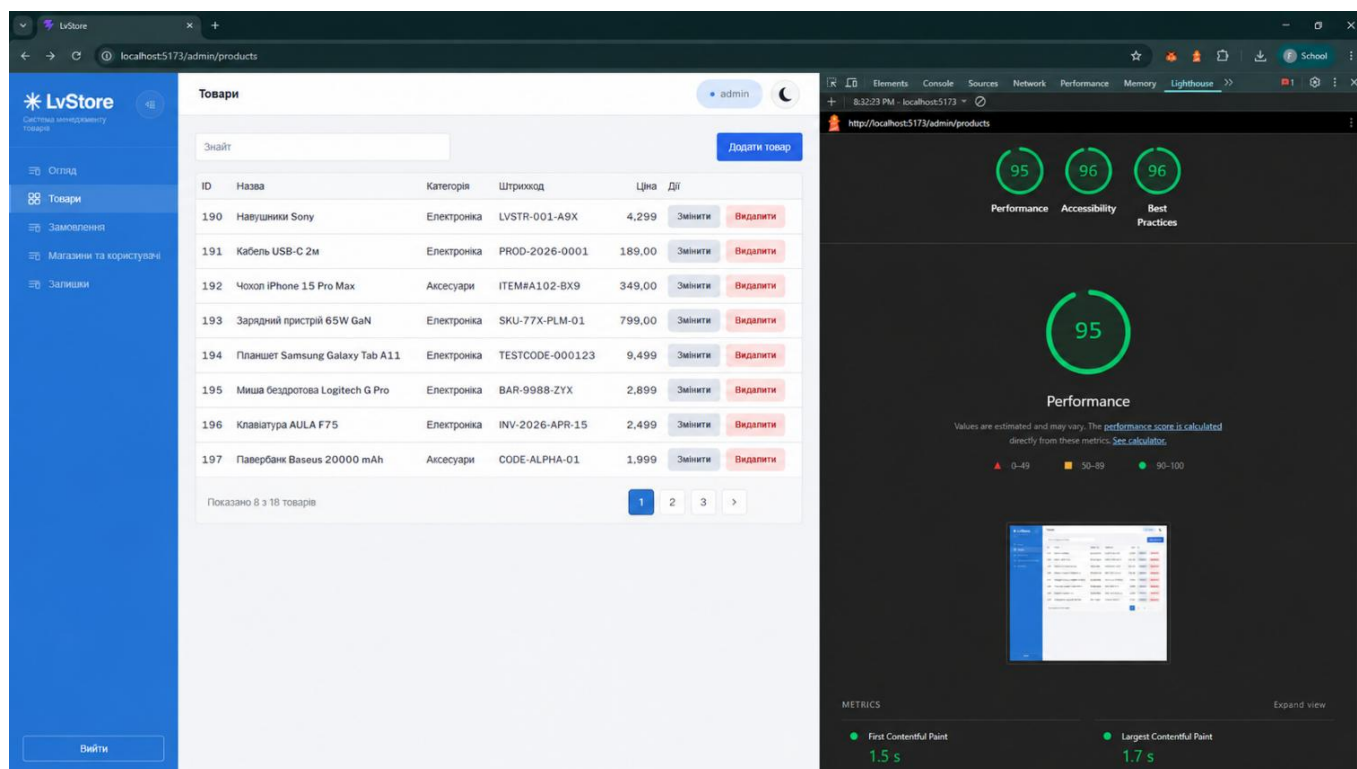


Рис. 4.14 Результати аудиту сторінки адміністратора у Google Lighthouse

Сторінка адміністратора отримала показник Performance 95, Accessibility 96 і Best Practices 96. Незважаючи на наявність таблиці товарів, кнопок дій, пагінації та бокової навігації, сторінка зберігає високий рівень продуктивності. Значення First Contentful Paint становило 1,5 с, а Largest Contentful Paint — 1,7 с.

Отримані результати показують, що адміністративний інтерфейс залишається придатним для швидкої роботи з даними. Для цієї ролі важливо, щоб таблиці, кнопки керування та елементи навігації не сповільнювали взаємодію з системою, оскільки адміністратор виконує найбільшу кількість операцій із товарами, замовленнями, користувачами та залишками.

Аудит сторінки продавця було виконано на прикладі сторінки замовлень. Ця сторінка містить таблицю замовлень, фільтри за статусом і магазином, кнопку створення нового замовлення та дію підтвердження отримання товару.

Результати аудиту сторінки продавця наведено на рисунку 4.15.

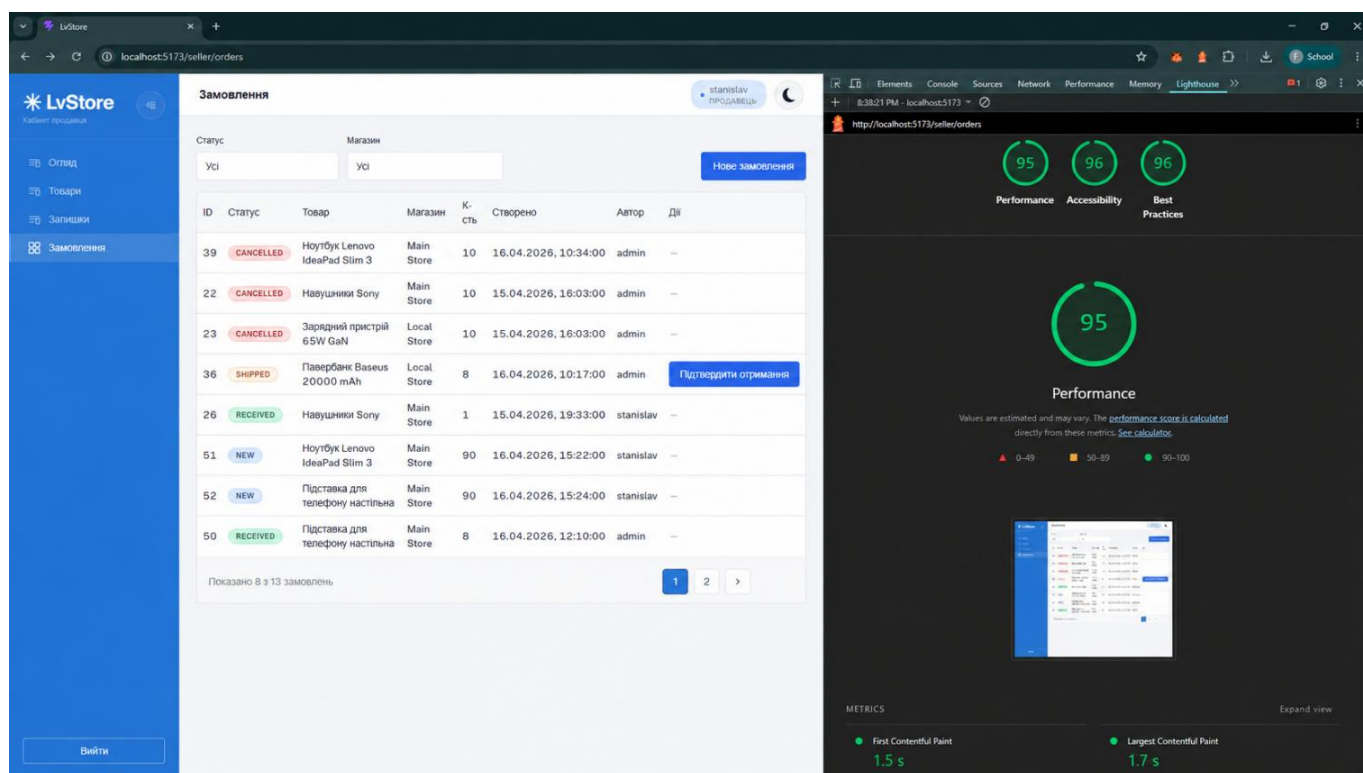


Рис. 4.15 Результати аудиту сторінки продавця у Google Lighthouse

Сторінка продавця отримала показник Performance 95, Accessibility 96 і Best Practices 96. Значення First Contentful Paint становило 1,5 с, а Largest Contentful Paint — 1,7 с. Це свідчить про те, що навіть сторінка з таблицею, статусами замовлень і кнопками дій завантажується без помітної затримки для користувача.

Для продавця продуктивність цієї сторінки має практичне значення, оскільки вона використовується для перегляду замовлень, створення заявок на поповнення та підтвердження отримання товарів. Результати аудиту показують, що інтерфейс підтримує швидку роботу з основними сценаріями без перевантаження сторінки зайвими елементами.

Аудит сторінки менеджера складу було проведено на прикладі сторінки керування складами. Ця сторінка містить список складів, дані про локацію, відповідального менеджера, а також кнопки створення, редагування і видалення записів.

Результати аудиту сторінки менеджера складу наведено на рисунку 4.16.

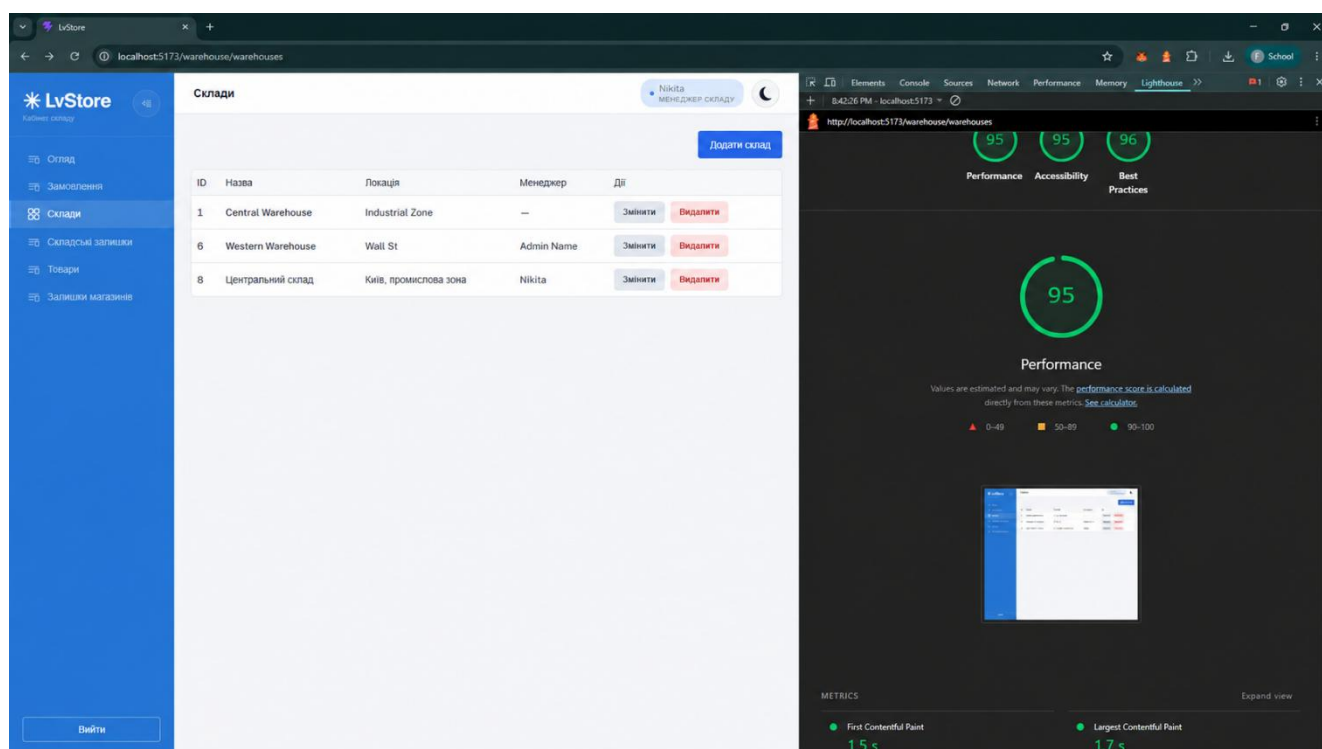


Рис. 4.16 Результати аудиту сторінки менеджера складу у Google Lighthouse

Сторінка менеджера складу отримала показник Performance 95, Accessibility 95 і Best Practices 96. First Contentful Paint становив 1,5 с, а Largest Contentful Paint — 1,7 с. Отримані значення підтверджують, що складська частина системи також має стабільні показники продуктивності та коректно відображає основні елементи інтерфейсу.

Для менеджера складу важливо швидко отримувати доступ до інформації про склади, складські залишки, товари та замовлення. Результати Google Lighthouse показують, що сторінка зберігає достатній рівень швидкодії навіть за наявності таблиці та кнопок керування. Це підтверджує придатність реалізованого інтерфейсу для виконання операційних задач у межах складської ролі.

За результатами аудиту Google Lighthouse встановлено, що всі перевірені сторінки LvStore мають високі показники продуктивності, доступності та відповідності базовим практикам веброзробки. Показники Performance перебувають у межах 95–97, Accessibility — 95–98, Best Practices — 94–96. Значення First Contentful Paint для перевірених сторінок становить 1,5 с, а Largest Contentful Paint — від 1,6 до 1,7 с. Це підтверджує, що Frontend-частина вебзастосунку забезпечує швидке відображення основних сторінок і може використовуватися для подальшої роботи з ролями, таблицями, формами та даними системи.

Окремо варто зазначити, що високі результати були отримані не лише для публічних сторінок, а й для рольових кабінетів, які містять більше інтерфейсних елементів: бокову навігацію, таблиці, фільтри, кнопки дій і пагінацію. Це свідчить про те, що реалізована структура компонентів не створює надмірного навантаження на клієнтську частину та зберігає стабільну швидкодію під час роботи з основними сценаріями адміністратора, продавця і менеджера складу.

Отже, результати нефункціонального тестування показують, що розроблена Frontend-частина LvStore є придатною не тільки з погляду функціональної працездатності, а й з погляду продуктивності та зручності використання.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто поставленої мети - розроблено Frontend-частину веборієнтованої інформаційної системи LvStore, призначеної для спрощення управління товарними залишками й обробки внутрішніх замовлень у мережі магазинів. На основі виконаного дослідження зроблено висновок, що для роздрібної мережі критично важливими є централізоване відображення актуальних даних, чітке розмежування ролей користувачів та наявність зручного вебінтерфейсу для щоденної роботи з товарами, залишками і замовленнями.

У ході аналізу предметної галузі встановлено, що основними проблемами управління товарними запасами в мережі магазинів є неактуальність інформації про залишки, складність координації дій між магазинами і складом, висока ймовірність помилок під час створення та обробки внутрішніх замовлень, а також потреба в чіткому розподілі функцій між адміністратором магазину, продавцем і менеджером складу. Проведений аналіз аналогів показав, що системи Торгсофт, WooCommerce та Loyverse POS лише частково покривають поставлену задачу, оскільки не забезпечують у повному обсязі рольово-орієнтовану модель взаємодії та спеціалізацію саме на внутрішніх процесах мережі магазинів. На підставі цього обґрунтовано доцільність розробки власного Frontend-рішення для системи LvStore.

На основі аналізу предметної галузі та існуючих рішень сформовано функціональні й нефункціональні вимоги до Frontend-частини вебзастосунку. Зроблено висновок, що ефективна клієнтська частина системи управління товарними запасами повинна забезпечувати автентифікацію та авторизацію користувачів, рольове розмежування доступу, перегляд товарів і залишків, створення та обробку замовлень, а також керування користувачами, магазинами і складами відповідно до ролі. Водночас обґрунтовано, що для практичного

використання система має відповідати вимогам щодо продуктивності, браузерної сумісності, адаптивності та безпечної взаємодії з серверною частиною.

У процесі аналізу технологій для реалізації Frontend-частини досліджено можливості React, TypeScript, Vite, Chakra UI, React Router, Axios, TanStack Query, Jest і React Testing Library. На підставі проведеного аналізу зроблено висновок, що саме цей технологічний стек є доцільним для розробки клієнтської частини системи LvStore, оскільки він забезпечує компонентний підхід до побудови інтерфейсу, статичну типізацію даних, швидке локальне розгортання та збірку застосунку, зручне створення інтерфейсних компонентів, організацію маршрутизації, стабільну взаємодію з REST API, ефективну роботу із серверним станом і можливість автоматизованого тестування.

У результаті проєктування клієнтської частини LvStore розроблено карту вебзастосунку, діаграму варіантів використання та діаграму компонентів Frontend-частини. Зроблено висновок, що обрана архітектура з поділом на публічну частину, сторінку авторизації та три рольові кабінети - адміністратора магазину, продавця і менеджера складу - відповідає логіці предметної галузі та забезпечує зручну організацію доступу до функцій системи. Така структура дозволяє послідовно реалізувати сценарії роботи користувачів і створює основу для подальшого розширення застосунку без порушення загальної архітектури.

Під час програмної реалізації Frontend-частини створено головну сторінку, сторінку входу, рольові кабінети, систему маршрутизації, guard-компоненти для перевірки доступу, API-шар для взаємодії з серверною частиною та набір спільних UI-компонентів. Доведено, що використання патерну Auth Guards у поєднанні з рольовою маршрутизацією забезпечує надійне розмежування доступу між користувачами системи, а застосування TanStack Query у клієнтській архітектурі забезпечує актуальність серверних даних і швидке відображення інформації після виконання операцій створення, редагування або видалення записів. У результаті користувачі отримують доступ лише до тих сторінок і дій, які відповідають їхній ролі в системі.

Проведене тестування підтвердило працездатність розробленої Frontend-частини та її відповідність поставленим функціональним і нефункціональним вимогам. Автоматизовані та ручні перевірки засвідчили коректність авторизації, валідації форм, відкриття модальних вікон підтвердження, створення записів, скасування критичних дій та обмеження доступу відповідно до ролі користувача. Кросбраузерне та адаптивне тестування показало, що інтерфейс коректно відображається в сучасних браузерах і на різних типах пристроїв. Реалізовані архітектурні та технологічні рішення забезпечують стабільну роботу клієнтської частини системи.

Отже, кваліфікаційна робота не лише дала практичний результат у вигляді реалізованої Frontend-частини системи LvStore, а й дозволила зробити обґрунтований висновок про доцільність використання рольово-орієнтованої клієнтської архітектури для задач управління товарами, залишками та внутрішніми замовленнями в мережі магазинів. Розроблене рішення забезпечує зручну взаємодію користувачів із системою, підтримує основні бізнес-сценарії предметної галузі та може бути використане як основа для подальшого розвитку повноцінної інформаційної системи у взаємодії з серверною частиною.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Гридько Б.А., Яскевич В.О. Розробка веб-системи менеджменту товарних запасів для мережі магазинів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 442-443.

2. Гридько Б.А. Рольово-орієнтована архітектура клієнтської частини вебсистеми менеджменту товарних запасів. Матеріали II Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Silver E. A., Pyke D. F., Thomas D. J. Inventory and Production Management in Supply Chains. 5th ed. Boca Raton : CRC Press, 2022. 824 p.
2. Zipkin P. H. Foundations of Inventory Management. New York : McGraw-Hill, 2024. 528 p.
3. Chopra S. Supply Chain Management: Strategy, Planning, and Operation. 8th ed. Harlow : Pearson, 2023. 528 p.
4. Торгсофт: програма обліку товарів та автоматизація торгівлі. URL: <https://torgsoft.ua/>
5. Inventory Management for WooCommerce Documentation. WooCommerce. URL: <https://woocommerce.com/document/inventory-management-for-woocommerce-2/>
6. Inventory Management Software. Loyverse POS. Inventory Management Software. Loyverse POS, система обліку. URL: <https://loyverse.com/advanced-inventory>
7. ISO/IEC/IEEE 29148:2024. Systems and software engineering - Life cycle processes - Requirements engineering. Geneva : ISO, 2024. 104 p.
8. ISO/IEC 25010:2023. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model. Geneva : ISO, 2023.
9. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley, 2021. 464 p.
10. Garrett J. J. The Elements of User Experience: User-Centered Design for the Web and Beyond. 2nd ed. Berkeley : New Riders, 2022. 192 p.
11. Nielsen J., Budiu R. Mobile Usability. 2nd ed. Berkeley : New Riders, 2024. 280 p.
12. Яскевич В. О. JavaScript-бібліотека React : навч. посіб. для студентів спеціальності «Комп'ютерні науки». Київ : Київський університет імені Бориса Грінченка, 2023.

13. The TypeScript Handbook. Microsoft. The TypeScript Handbook. Microsoft, довідник. URL: <https://www.typescriptlang.org/docs/handbook/intro.html>
14. Getting Started. Vite. URL: <https://vite.dev/guide/> (date of access: 19.05.2026).
15. Chakra UI Documentation. Chakra UI. URL: <https://www.chakra-ui.com/>
16. createBrowserRouter. React Router Documentation. createBrowserRouter. React Router Documentation, маршрутизація. URL: <https://reactrouter.com/api/data-routers/createBrowserRouter>
17. First steps. Axios Documentation. URL: <https://axios.rest/pages/getting-started/first-steps>
18. TanStack Overview. TanStack Query React Documentation, бібліотека. URL: <https://tanstack.com/query/latest/docs/framework/react/overview>
19. Getting Started. Jest Documentation. URL: <https://jestjs.io/docs/getting-started>
20. React Testing Library. Testing Library, тестування. URL: <https://testing-library.com/docs/react-testing-library/intro/>
21. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. OWASP Foundation, 2021. URL: <https://owasp.org/www-project-top-ten/>
22. Transport Layer Security (TLS). MDN Web Docs, протокол. URL: https://developer.mozilla.org/enUS/docs/Web/Security/Defenses/Transport_Layer_Security
23. First Contentful Paint (FCP). Chrome for Developers, метрика. URL: <https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint>
24. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2020. 705 p.
25. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. Harlow : Pearson, 2020. 368 p.
26. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Harlow : Pearson, 2021. 864 p.
27. Crispin L., Gregory J. Agile Testing Condensed: A Brief Introduction. 2nd ed. Raleigh : Pragmatic Bookshelf, 2023. 160 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



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



Система менеджменту товарів для оптимізації мережі магазинів.
Спец-частина: розробка Frontend-частини системи

Виконав студент 4 курсу
групи ПД-41
Грицько Богдан Андрійович
Керівник роботи

канд.техн.наук, доцент Яскевич Владислав Олександрович
Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення управління товарними залишками й обробки замовлень у роздрібній мережі за рахунок розробки клієнтської частини веб-орієнтованої інформаційної системи LvStore.

Об'єкт дослідження: процес розробки веб-орієнтованого програмного забезпечення для обробки даних про товари, товарні залишки та замовлення в інформаційній системі роздрібної мережі.

Предмет дослідження: методи, засоби та технології побудови клієнтської частини веб-орієнтованої інформаційної системи управління товарними залишками та замовленнями в роздрібній мережі.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь управління товарними залишками та внутрішніми замовленнями в мережі магазинів.
2. Дослідити існуючі програмні рішення для обліку товарів, запасів і замовлень, визначити їх переваги та недоліки.
3. Сформулювати функціональні й нефункціональні вимоги до Frontend-частини системи LvStore.
4. Обґрунтувати вибір технологічного стеку для реалізації Frontend-частини системи.
5. Спроекувати структуру вебзастосунку, рольову модель доступу та основні компоненти клієнтської частини.
6. Реалізувати клієнтську частину вебсистеми з кабінетами адміністратора, продавця та менеджера складу.
7. Провести тестування Frontend-частини та перевірити коректність роботи основних сценаріїв користувачів.

3

АНАЛІЗ АНАЛОГІВ

Показник	Торгсофт	WooCommerce	Loyverse POS	LvStore
Спеціалізація на внутрішній логістиці	Частково	Частково	Частково	Так
Автоматизація внутрішніх замовлень	Частково	Ні	Частково	Так
Жорстка багаторівнева рольова модель	Так	Ні	Ні	Так
Захист даних від випадкового видалення	Так	Частково	Так	Так
Портативність та ізолюваність розгортання	Ні	Частково	Ні	Так

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

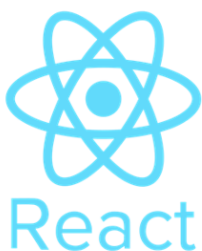
1. Автентифікація та авторизація користувачів (вхід у систему та надання доступу до функцій відповідно до ролі користувача).
2. Розмежування прав доступу (система ролей: продавець, адміністратор магазину, менеджер складу).
3. Моніторинг товарної номенклатури та залишків (забезпечення оперативного доступу до актуальної інформації про кількість та стан товарів для всіх авторизованих ролей).
4. Обробка та менеджмент замовлень (створення внутрішніх замовлень на поповнення запасів, перевірка наявності товару та зміна статусів виконання).
5. Контроль товарних запасів (моніторинг залишків у магазинах та на складах, відстеження критичного рівня запасів).

Нефункціональні вимоги:

1. Продуктивність. Час початкового відображення основного контенту (First Contentful Paint) не повинен перевищувати 1,5 секунди при стандартному 4G-з'єднанні.
2. Сумісність. Клієнтський вебзастосунок повинен коректно працювати в актуальних версіях Google Chrome, Mozilla Firefox та Microsoft Edge. Інтерфейс повинен коректно відображатися на екранах шириною від 360 px до 1920 px.
3. Безпека обміну даними. Взаємодія клієнтської частини із серверною повинна здійснюватися через HTTPS із використанням протоколу TLS 1.2 або вище. Доступ до функцій системи повинен надаватися лише авторизованим користувачам відповідно до їхньої ролі.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ


VITE

chakra

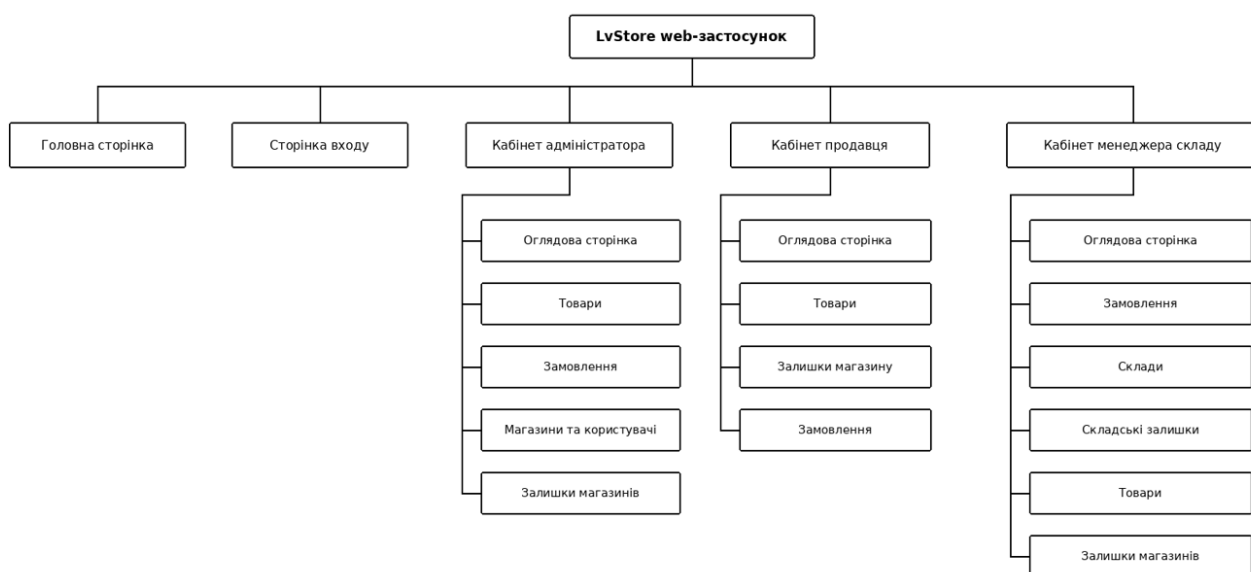
A X 1 O S

React Router

Jest

6

КАРТА ВЕБЗАСТОСУНКУ



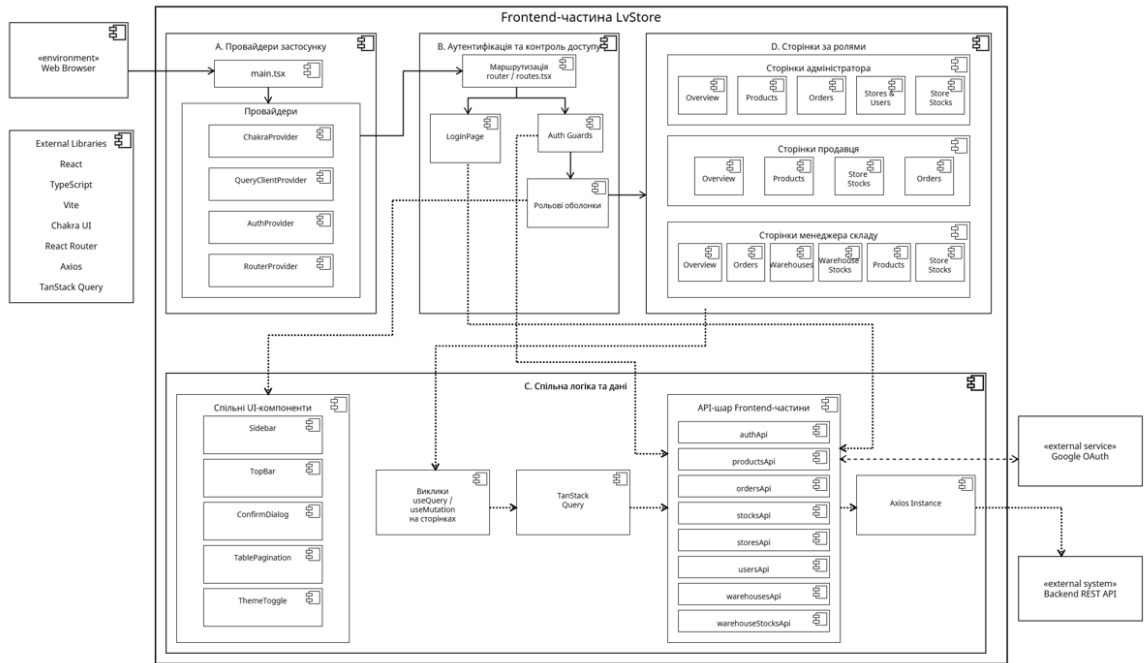
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



8

ДІАГРАМА КОМПОНЕНТІВ



9

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

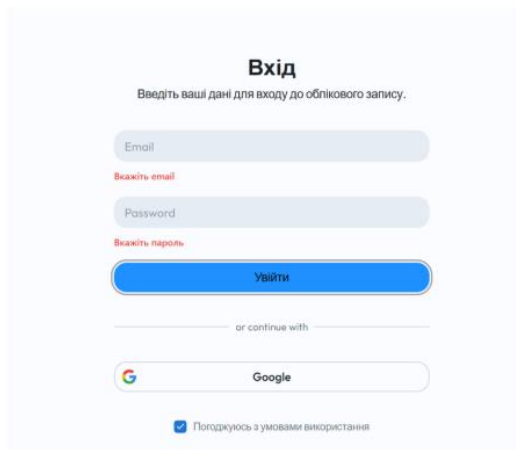


Рис. 4.1 Результат тестування валідації форми авторизації користувача

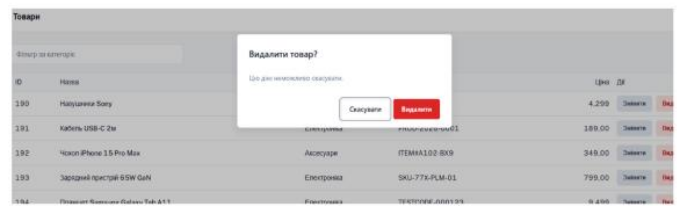


Рис. 4.2 Результат тестування модального вікна видалення товару

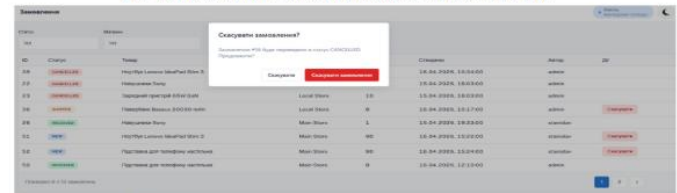


Рис. 4.3 Результат тестування модального вікна скасування замовлення

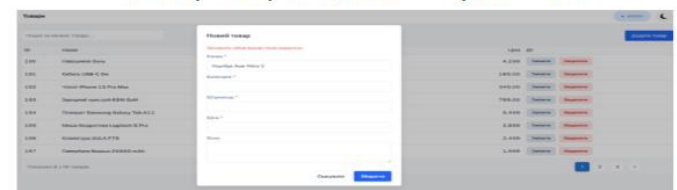


Рис. 4.4 Результат тестування форми створення нового товару

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Гридько Б.А., Яскевич В.О. Розробка веб-системи менеджменту товарних запасів для мережі магазинів. Матеріали Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025. – С. 442–443.
2. Гридько Б.А. Рольово-орієнтована архітектура клієнтської частини вебсистеми менеджменту товарних запасів. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. (подано до друку).

12

ВИСНОВКИ

1. Дослідження предметної галузі дозволило сформулювати вимоги до системи з урахуванням ролей та бізнес-процесів.
2. Аналіз аналогів підтвердив, що існуючі рішення не забезпечують потрібну рольову модель, тому розробка власного Frontend LvStore є обґрунтованою.
3. На основі аналізу технологій обрано стек React, TypeScript, Vite, Chakra UI, React Router, Axios, TanStack Query, Jest — як оптимальний для задач системи.
4. Архітектура Frontend-частини з рольовим доступом і кабінетами користувачів забезпечує зручну організацію функцій.
5. Доведено, що Auth Guards та TanStack Query гарантують надійне розмежування доступу і швидкість відображення даних (FCP < 1.5с).
6. Тестування підтвердило відповідність Frontend-частини функціональним і нефункціональним вимогам.

13

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```

import { QueryClient, QueryClientProvider } from
"@tanstack/react-query";
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import { RouterProvider } from "react-router-dom";
import { router } from "../router/routes";

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 60_000,
      retry: 1,
    },
  },
});

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <QueryClientProvider client={queryClient}>
      <ChakraProvider value={defaultSystem}>
        <RouterProvider router={router} />
      </ChakraProvider>
    </QueryClientProvider>
  </StrictMode>,
);
import { createBrowserRouter, Navigate } from "react-router-dom";
import { AppShell } from "../components/layout/AppShell";
import { AdminAuthGuard } from
"../pages/admin/AdminAuthGuard";
import { AdminShell } from "../pages/admin/AdminShell";
import { AdminOverviewPage } from
"../pages/admin/dashboard/AdminOverviewPage";
import { AdminOrdersPage } from
"../pages/admin/orders/AdminOrdersPage";
import { AdminProductsPage } from
"../pages/admin/products/AdminProductsPage";
import { AdminStocksPage } from
"../pages/admin/stocks/AdminStocksPage";
import { AdminStoresUsersPage } from
"../pages/admin/stores-users/AdminStoresUsersPage";
import { LoginPage } from "../pages/auth/LoginPage";
import { MainPage } from "../pages/main/MainPage";
import { SellerAuthGuard } from
"../pages/seller/SellerAuthGuard";
import { SellerShell } from "../pages/seller/SellerShell";
import { SellerOverviewPage } from
"../pages/seller/dashboard/SellerOverviewPage";
import { SellerOrdersPage } from
"../pages/seller/orders/SellerOrdersPage";
import { SellerProductsPage } from
"../pages/seller/products/SellerProductsPage";
import { SellerStocksPage } from
"../pages/seller/stocks/SellerStocksPage";
import { WarehouseAuthGuard } from
"../pages/warehouse/WarehouseAuthGuard";
import { WarehouseShell } from
"../pages/warehouse/WarehouseShell";
import { WarehouseOverviewPage } from
"../pages/warehouse/dashboard/WarehouseOverviewPage";
import { WarehouseOrdersPage } from
"../pages/warehouse/orders/WarehouseOrdersPage";
import { WarehouseProductsPage } from
"../pages/warehouse/products/WarehouseProductsPage";

```

```

import { WarehouseStocksPage } from
"../pages/warehouse/stocks/WarehouseStocksPage";
import { WarehouseWarehouseStocksPage } from
"../pages/warehouse/warehouse-
stocks/WarehouseWarehouseStocksPage";
import { WarehouseWarehousesPage } from
"../pages/warehouse/warehouses/WarehouseWarehousesPage";

export const router = createBrowserRouter([
  {
    path: "/",
    element: <AppShell />,
    children: [{ index: true, element: <MainPage /> }],
  },
  { path: "/login", element: <LoginPage /> },
  {
    path: "/seller",
    element: <SellerAuthGuard />,
    children: [
      {
        element: <SellerShell />,
        children: [
          { index: true, element: <Navigate to="products" replace
/> },
          { path: "dashboard", element: <SellerOverviewPage /> },
          { path: "products", element: <SellerProductsPage /> },
          { path: "stocks", element: <SellerStocksPage /> },
          { path: "orders", element: <SellerOrdersPage /> },
        ],
      },
    ],
  },
  {
    path: "/admin",
    element: <AdminAuthGuard />,
    children: [
      {
        element: <AdminShell />,
        children: [
          { index: true, element: <Navigate to="dashboard" replace
/> },
          { path: "dashboard", element: <AdminOverviewPage
/> },
          { path: "products", element: <AdminProductsPage /> },
          { path: "orders", element: <AdminOrdersPage /> },
          { path: "stores-users", element: <AdminStoresUsersPage
/> },
          { path: "stocks", element: <AdminStocksPage /> },
        ],
      },
    ],
  },
  {
    path: "/warehouse",
    element: <WarehouseAuthGuard />,
    children: [
      {
        element: <WarehouseShell />,
        children: [
          { index: true, element: <Navigate to="dashboard" replace
/> },
          { path: "dashboard", element: <WarehouseOverviewPage
/> },
          { path: "orders", element: <WarehouseOrdersPage /> },

```

```

    { path: "warehouses", element:
<WarehouseWarehousesPage /> },
    { path: "warehouse-stocks", element:
<WarehouseWarehouseStocksPage /> },
    { path: "products", element: <WarehouseProductsPage
/> },
    { path: "stocks", element: <WarehouseStocksPage /> },
  ],
},
],
},
});

import {
  Box,
  Button,
  Field,
  Flex,
  Image,
  Input,
  Text,
  VisuallyHidden,
  VStack,
} from "@chakra-ui/react";
import { useEffect, useId, useState } from "react";
import { NavLink, useNavigate, useSearchParams } from
"react-router-dom";
import { imgAuthCheckEmpty, imgAuthDivider,
imgAuthGoogle, imgAuthLogomark } from
"../assets/figmaAuthAssets";
import { getCurrentUser, getGoogleOAuthAuthorizationUrl,
login } from "../api/authApi";
import { getLoginErrorMessage } from "../loginError";
import { getPostLoginPath } from "../useLoginMutation";
import { LvStoreLoginSpinner } from
"/LvStoreLoginSpinner";

const pageBg = "#fafcff";
const inputBg = "rgba(37, 82, 126, 0.09)";
const headerText = "#1b2128";
const placeholderColor = "rgba(20, 33, 46, 0.4)";
const checkboxLabel = "rgba(20, 33, 46, 0.62)";
const btnBg = "#1e90ff";
const btnLabel = "#000305";
const logoBlue = "#2274d4";
const googleBtnBg = "#fafcff";
const googleBtnBorder = "rgba(73, 92, 110, 0.2)";
const dividerLine = "rgba(20, 33, 46, 0.14)";
const orContinueColor = "rgba(20, 33, 46, 0.62)";

const inputShell = {
  bg: inputBg,
  borderWidth: "1.5px",
  borderColor: "transparent",
  rounded: "16px",
  px: "12px",
  py: "8px",
  minH: "36px",
  w: "full",
  fontFamily: "'Outfit', 'Noto Sans', 'Public Sans', sans-serif",
  fontSize: "15px",
  fontWeight: "normal",
  lineHeight: "20px",
  color: headerText,
  _placeholder: { color: placeholderColor },
  _focusVisible: {
    outline: "none",
    borderColor: "rgba(34, 116, 212, 0.55)",
    boxShadow: "0 0 1px rgba(34, 116, 212, 0.25)",
  },
};

```

```

function CheckFilled16() {
  return (
    <Box
      boxSize="16px"
      rounded="4px"
      bg={logoBlue}
      display="flex"
      alignItems="center"
      justifyContent="center"
      flexShrink={0}
      aria-hidden
    >
      <svg width="10" height="8" viewBox="0 0 12 10"
fill="none" aria-hidden>
        <path
          d="M1 5l3 3 7-7"
          stroke="white"
          strokeWidth="1.5"
          strokeLinecap="round"
          strokeLinejoin="round"
        />
      </svg>
    </Box>
  );
}

function randomLoginUiDelayMs(): number {
  return 5000 + Math.floor(Math.random() * 2001);
}

export function LoginPage() {
  const navigate = useNavigate();
  const [searchParams] = useSearchParams();
  const emailId = useId();
  const passwordId = useId();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [termsAccepted, setTermsAccepted] = useState(false);
  const [emailError, setEmailError] = useState<string |
null>(null);
  const [passwordError, setPasswordError] = useState<string |
null>(null);
  const [termsError, setTermsError] = useState<string |
null>(null);
  const [formError, setFormError] = useState<string |
null>(null);
  const [loginBusy, setLoginBusy] = useState(false);

  const loading = loginBusy;

  useEffect(() => {
    const oauth = searchParams.get("oauth2");
    const oauthError = searchParams.get("error");
    if (oauth === "access_denied" || oauthError !== null) {
      setFormError(
        "Вхід через Google неможливий: обліковий запис з
цим email не знайдено в системі, email не підтверджено в
Google або вхід скасовано. Зверніться до адміністратора.",
      );
    }
  }, [searchParams]);

  const validate = (): boolean => {
    let ok = true;
    if (!email.trim()) {
      setEmailError("Вкажіть email");
      ok = false;
    } else {
      setEmailError(null);
    }
  }
}

```

```

    }
    if (!password) {
      setPasswordError("Вкажіть пароль");
      ok = false;
    } else {
      setPasswordError(null);
    }
    if (!termsAccepted) {
      setTermsError("Потрібно погодитися з умовами використання.");
      ok = false;
    } else {
      setTermsError(null);
    }
    return ok;
    import axios from "axios";

export function getApiBaseUrl(): string {
  const env = import.meta.env;
  const fromBase = env.VITE_API_BASE_URL;
  const fromLegacy = env.VITE_API_URL;
  if (typeof fromBase === "string" && fromBase.trim() !== "") {
    return fromBase.trim();
  }
  if (typeof fromLegacy === "string" && fromLegacy.trim() !== "") {
    return fromLegacy.trim();
  }
  return "http://localhost:8080";
}

export const axiosInstance = axios.create({
  baseURL: getApiBaseUrl(),
  withCredentials: true,
});

import type { AuthUser, LoginRequest, LoginResponse } from "../types";
import { axiosInstance, getApiBaseUrl } from "../axiosInstance";

/** Spring Security OAuth2 client: початок входу через Google (лише для користувачів, уже створених у БД). */
export function getGoogleOAuthAuthorizationUrl(): string {
  return `${getApiBaseUrl()}/oauth2/authorization/google`;
}

export async function login(body: LoginRequest): Promise<LoginResponse> {
  const { data } = await
  axiosInstance.post<LoginResponse>("/auth/token", body);
  return data;
}

export async function getCurrentUser(): Promise<AuthUser> {
  const { data } = await
  axiosInstance.get<AuthUser>("/auth/me");
  return data;
}

export async function logout(): Promise<void> {
  await axiosInstance.post("/auth/logout");
}

import "@testing-library/jest-dom/jest-globals";
import { ChakraProvider, defaultSystem } from "@chakra-ui/react";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { act, render, screen, waitFor } from "@testing-library/react";

```

```

import userEvent from "@testing-library/user-event";
import { AxiosError } from "axios";
import { describe, expect, it, jest } from "@jest/globals";
import React from "react";
import { createMemoryRouter, RouterProvider } from "react-router-dom";
import { getCurrentUser, getGoogleOAuthAuthorizationUrl, login } from "../api/authApi";
import { LoginPage } from "../LoginPage";

const mockGoogleUrl =
"http://localhost:8080/oauth2/authorization/google";

jest.mock("../api/authApi", () => ({
  login: jest.fn(),
  getCurrentUser: jest.fn(),
  getGoogleOAuthAuthorizationUrl: jest.fn(() => mockGoogleUrl),
}));

function renderLoginAt(initialPath = "/login") {
  const queryClient = new QueryClient({
    defaultOptions: { queries: { retry: false }, mutations: { retry: false } },
  });
  const router = createMemoryRouter(
    [
      { path: "/login", element: <LoginPage /> },
      {
        path: "/admin/dashboard",
        element: <div data-testid="dest-admin">Admin
dest</div>,
      },
      {
        path: "/seller/products",
        element: <div data-testid="dest-seller">Seller dest</div>,
      },
      {
        path: "/warehouse/dashboard",
        element: <div data-testid="dest-warehouse">Warehouse
dest</div>,
      },
    ],
    { initialEntries: [initialPath] },
  );
  return render(
    <QueryClientProvider client={queryClient}>
      <ChakraProvider value={defaultSystem}>
        <RouterProvider router={router} />
      </ChakraProvider>
    </QueryClientProvider>,
  );
}

describe("LoginPage", () => {
  it("renders login heading and fields", () => {
    renderLoginAt();
    expect(screen.getByRole("heading", { name: "Вхід" })).toBeInTheDocument();
    expect(screen.getByTestId("login-email")).toBeInTheDocument();
    expect(screen.getByTestId("login-password")).toBeInTheDocument();
    expect(screen.getByTestId("login-submit")).toBeInTheDocument();
    expect(screen.getByTestId("login-google")).toBeInTheDocument();
    expect(screen.getByText("or continue with")).toBeInTheDocument();
  });
});

```

```

it("shows OAuth error when redirected with
oauth2=access_denied", () => {
  renderLoginAt("/login?oauth2=access_denied");
  expect(screen.getByTestId("login-form-
error")).toBeInTheDocument();
  expect(screen.getByText(/Вхід через Google
неможливий/i)).toBeInTheDocument();
});

it("requires terms before starting Google login", async () => {
  const user = userEvent.setup();
  const assignSpy = jest.spyOn(window.location,
"assign").mockImplementation(() => {});
  renderLoginAt();
  await user.click(screen.getByTestId("login-google"));
  expect(await screen.findByText("Потрібно погодитися з
умовами використання.")).toBeInTheDocument();
  expect(assignSpy).not.toHaveBeenCalled();
  assignSpy.mockRestore();
});

it("navigates to Google OAuth URL when terms accepted and
Google clicked", async () => {
  const user = userEvent.setup();
  const assignSpy = jest.spyOn(window.location,
"assign").mockImplementation(() => {});
  renderLoginAt();
  await user.click(screen.getByRole("button", { name:
/Погоджуюсь з умовами/i }));
  await user.click(screen.getByTestId("login-google"));

  expect(getGoogleOAuthAuthorizationUrl).toHaveBeenCalled();
};

expect(assignSpy).toHaveBeenCalledWith(mockGoogleUrl);
assignSpy.mockRestore();
});

it("shows validation errors when submitting empty form",
async () => {
  const user = userEvent.setup();
  renderLoginAt();
  await user.click(screen.getByTestId("login-submit"));
  expect(await screen.findByText("Вкажіть
email")).toBeInTheDocument();
  expect(screen.getByText("Вкажіть
пароль")).toBeInTheDocument();
});

it("runs login then getCurrentUser and redirects by role",
async () => {
  jest.useFakeTimers();
  const user = userEvent.setup({ advanceTimers:
jest.advanceTimersByTime });
  jest.mocked(login).mockResolvedValue({ token: "cookie-
set-by-server" });
  jest.mocked(getCurrentUser).mockResolvedValue({
    id: 1,
    email: "admin@gmail.com",
    username: "admin",
    role: "STORE_ADMINISTRATOR",
  });

  renderLoginAt();
  await user.type(screen.getByTestId("login-email"),
"admin@gmail.com");

```