МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально науковий інститут ІТ та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «Розробка фронтенд-частини інтернет-магазину
з продажу електронних засобів»

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Ковень Станіслав Сергійович

Керівник: ***Ляховчук С.В., викладач кафедри ІТАД***

Рецензент: ***кандидат технічних наук, доцент,***
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «20» травня 2026 р.

Острог, 2026

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1. Розробити головну сторінку магазину з секціями рекомендованих товарів, категорій та промо-банерами.

2. Розробити сторінку каталогу товарів із фільтрацією за категорією, брендом і ціною та детальну сторінку товару з галереєю зображень, характеристиками та відгуками.

3. Реалізувати кошик, список бажань, порівняння товарів та сторінку оформлення замовлення з вибором доставки і купонами на знижку.

4. Розробити систему реєстрації та входу користувача, особистий кабінет з історією замовлень.

5. Розробити адмін-панель для управління товарами, категоріями, замовленнями та користувачами.

АНОТАЦІЯ

кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Розробка фронтенд-частини інтернет-магазину з продажу електронних засобів.

Автор: Ковень Станіслав Сергійович

Науковий керівник: викладач кафедри інформаційних технологій та аналітики даних Ляховчук Сергій Васильович.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 95 с., 27 рис., 5 табл., 7 додатків, 27 джерел.

Ключові слова: ІНТЕРНЕТ-МАГАЗИН, ФРОНТЕНД, NEXT.JS, REACT, TYPESCRIPT, ZUSTAND, TAILWIND CSS, ЕЛЕКТРОННА КОМЕРЦІЯ, КЕРУВАННЯ СТАНОМ, АДАПТИВНИЙ ІНТЕРФЕЙС.

Короткий зміст праці:

Кваліфікаційна робота присвячена проєктуванню та розробці клієнтської частини веб-застосунку електронної комерції — інтернет-магазину з продажу електронних засобів TechStore. Об'єктом дослідження є процес розробки клієнтської частини сучасного веб-застосунку електронної комерції, а предметом — методи, інструменти та архітектурні рішення, що дають змогу побудувати швидку, адаптивну й зручну в супроводі фронтенд-частину інтернет-магазину електроніки.

У роботі проаналізовано український ринок електронної комерції та інтерфейсні рішення інтернет-магазинів-аналогів Rozetka, Comfy і Foxtrot. На основі аналізу сформульовано функціональні та нефункціональні вимоги до TechStore, побудовано UML-діаграми, ER-подібну модель документної бази MongoDB, діаграму класів Zustand-сторів і діаграму послідовностей для сценарію оформлення замовлення.

У практичній частині реалізовано клієнтську частину TechStore з використанням Next.js 14, React 18, TypeScript, Tailwind CSS, CSS Modules, Zustand, Axios, Firebase Authentication, React Hook Form, Zod, Jest і Playwright. Застосунок містить головну сторінку, каталог товарів із фільтрацією та сортуванням, картку товару, кошик, checkout, особистий кабінет, список бажань, порівняння товарів і адміністративну панель. Реалізовано світлу й темну тему, адаптивний інтерфейс, оптимізацію зображень і розгортання фронтенду на Vercel.

Результати тестування підтверджують працездатність основних сценаріїв. Для клієнтської частини зафіксовано 53 успішних Jest-тести, для серверної частини — 27 тестів; загалом 80 юніт-тестів проходять без помилок. Покриття коду за рядками: фронтенд — 19,10 %, бекенд — 19,25 %. Playwright e2e-тести пройдені повністю — 44 із 44 сценаріїв. Загальна кількість автоматизованих тестів і сценаріїв — 124. Lighthouse-аудит продакшн-версії показав Performance 97 на мобільному та 100 на десктопі, SEO 100, LCP 2,6 с / 0,6 с, TBT 40 мс / 0 мс. Результати роботи можуть бути використані як основа для подальшого розвитку навчально-практичного прототипу інтернет-магазину електроніки.

_____

(підпис автора)

ANNOTATION

of the qualification work
for obtaining the educational degree of Bachelor

Topic: Development of the Front-End Part of an Online Store for Selling Electronic Devices.

Author: Stanislav Serhiyovych Koven

Scientific Supervisor: Lecturer of the Department of Information Technology and Data Analytics, Serhii Vasyliovych Liakhovchuk.

Defended on "……" …… 20__.

Explanatory note to the qualification work: 95 p., 27 fig., 5 tab., 7 appendices, 27 sources.

Keywords: ONLINE STORE, FRONT-END, NEXT.JS, REACT, TYPESCRIPT, ZUSTAND, TAILWIND CSS, E-COMMERCE, STATE MANAGEMENT, RESPONSIVE INTERFACE.

Brief content of the work:

The qualification work is devoted to the design and development of the client-side of an e-commerce web application — the TechStore online store for selling electronic devices. The object of the research is the process of developing the client-side of a modern e-commerce web application, while the subject is the methods, tools, and architectural solutions that make it possible to build a fast, responsive, and maintainable front-end for an electronics online store.

The work analyzes the Ukrainian e-commerce market and the interface solutions of comparable online stores: Rozetka, Comfy, and Foxtrot. Based on this analysis, functional and non-functional requirements for TechStore are formulated. UML diagrams, an ER-like model of the MongoDB document database, a class diagram of Zustand stores, and a sequence diagram for the order checkout scenario are developed.

The practical part implements the TechStore client-side using Next.js 14, React 18, TypeScript, Tailwind CSS, CSS Modules, Zustand, Axios, Firebase Authentication, React Hook Form, Zod, Jest, and Playwright. The application includes the home page, product catalog with filtering and sorting, product page, shopping cart, checkout, user account, wishlist, product comparison, and administrative panel. A light/dark theme system, responsive interface, image optimization, and Vercel deployment are implemented.

The testing results confirm the operability of the main scenarios. For the client-side, 53 Jest tests pass, and for the server-side 27 tests pass; 80 unit tests pass in total. Code line coverage: front-end — 19.10 %, back-end — 19.25 %. Playwright end-to-end testing was completed successfully — 44 out of 44 scenarios passed. The total number of automated tests and scenarios is 124. The Lighthouse audit of the production version showed Performance 97 on mobile and 100 on desktop, SEO 100, LCP 2.6 s / 0.6 s, TBT 40 ms / 0 ms. The results of the work can be used as a basis for further development of an educational and practical electronics online store prototype.

_____

(author's signature)

ВСТУП

Електронна комерція в Україні протягом останніх років стала одним із основних каналів роздрібної торгівлі. Особливо це стосується сегмента електроніки та побутової техніки, де покупець перед оформленням замовлення зазвичай порівнює характеристики, ціни, умови доставки, гарантії та відгуки. За даними EVO, наведеними у дослідженні Promodo, у 2023 році майже 10 млн українців здійснювали онлайн-покупки, з них 1,5 млн зробили це вперше, а загальний обсяг онлайн-продажів товарів і послуг перевищив 182 млрд грн [3].

Зростання онлайн-торгівлі підвищує вимоги до якості інтерфейсів інтернет-магазинів. Для користувача важливими є швидке завантаження сторінок, зрозуміла структура каталогу, коректна робота фільтрів, зручний кошик і мінімальна кількість дій під час оформлення замовлення. У сегменті електроніки ці вимоги мають особливе значення, оскільки товари часто мають велику кількість технічних характеристик, а ціна покупки може суттєво відрізнятися залежно від моделі, комплектації та продавця.

Окрему роль відіграє мобільний сценарій використання. Значна частина користувачів переглядає товари зі смартфонів, тому інтерфейс інтернет-магазину має однаково коректно працювати на різних розмірах екрана. Для такого типу застосунків адаптивність уже не є додатковою перевагою, а належить до базових вимог.

Продуктивність сторінок безпосередньо впливає на поведінку користувачів. За даними Google/SOASTA, збільшення часу завантаження мобільної сторінки з 1 до 5 секунд підвищує ймовірність відмови на 90 % [5]. Тому під час розробки клієнтської частини інтернет-магазину необхідно враховувати не лише функціональність, а й швидкість рендерингу, обсяг JavaScript-коду, оптимізацію зображень і стабільність інтерфейсу.

Додатковим чинником для українського ринку є нестабільність умов доступу до мережі, зокрема використання мобільного інтернету, робота під час відключень електроенергії та переривання сесій користувача. У таких умовах важливими стають

збереження стану кошика, коректна обробка помилок мережі та можливість продовжити оформлення замовлення без втрати введених даних.

У TechStore ці вимоги враховано через поєднання компонентної моделі React, маршрутизації Next.js, типізації TypeScript, глобального стану Zustand, оптимізації зображень і базового тестування. Під час реалізації найбільше уваги приділялося тим сценаріям, які користувач виконує найчастіше: пошуку товару, роботі з фільтрами, додаванню позицій у кошик, оформленню замовлення та перегляду інформації в особистому кабінеті [4].

Метою кваліфікаційної роботи є проєктування та розробка клієнтської частини інтернет-магазину з продажу електронних засобів TechStore з використанням сучасного фронтенд-стеку, що забезпечує адаптивність, продуктивність, оптимізацію для пошукових систем і зручний користувацький досвід.

Для досягнення поставленої мети визначено такі завдання:

1) проаналізувати стан українського ринку електронної комерції в сегменті електроніки, виявити поведінкові особливості цільової аудиторії та сформулювати ключові вимоги до інтерфейсу;

2) провести огляд провідних інтернет-магазинів-аналогів (Rozetka, Comfy, Foxtrot), виявити сильні й слабкі сторони їхніх інтерфейсних рішень та виокремити практики, придатні до запозичення;

3) сформулювати функціональні та нефункціональні вимоги до системи TechStore;

4) спроєктувати структуру даних та модель взаємодії клієнтської частини з серверною (UML-діаграми використання, класів, ER-подібна діаграма документної моделі);

5) обґрунтувати вибір технологічного стеку для реалізації фронтенду й описати ключові архітектурні рішення;

6) реалізувати клієнтську частину магазину, що включає каталог, картку товару, кошик, оформлення замовлення, особистий кабінет та адміністративну панель;

7) налаштувати розгортання фронтенду на Vercel, бекенду на Railway та базове тестування клієнтської і серверної частин;

8) підготувати керівництво користувача та продемонструвати працездатність системи.

Об'єктом дослідження є процес розробки клієнтської частини сучасного веб-застосунку електронної комерції. Предметом — методи, інструменти та архітектурні рішення, що дозволяють побудувати швидку, адаптивну й зручну в супроводі фронтенд-частину інтернет-магазину з продажу електроніки.

У роботі застосовано загальнонаукові та спеціальні методи. Аналіз і синтез — для опрацювання літератури з електронної комерції та документації фреймворків; порівняння — для зіставлення наявних аналогів і кандидатів-технологій; класифікація — для впорядкування вимог. Зі спеціальних: методи об'єктно-орієнтованого моделювання (UML), інфологічного моделювання (ER-діаграми), прототипування інтерфейсу та практичної реалізації коду.

Декларація про використання генеративних інструментів. Під час підготовки роботи генеративні інструменти використовувалися як допоміжний засіб: для впорядкування структури тексту, пошуку стилістичних неточностей, скорочення повторів і перекладу окремих фрагментів анотації. Змістові рішення, опис архітектури, добір джерел, скріншоти, діаграми, програмна реалізація та підсумкові висновки перевірені й доопрацьовані автором. Фактичні твердження звірено з документацією та джерелами, наведеними у списку використаних джерел.

Структура роботи: вступ, три розділи, висновки, список використаних джерел та додатки. Перший розділ — «Загальні положення» — аналіз ринку, огляд аналогів, постановка задачі. Другий — «Інформаційне та математичне забезпечення» — проєктування системи: моделі даних, діаграми, алгоритмічне забезпечення. Третій — «Програмне та технічне забезпечення» — опис стеку, вимоги, опис реалізації, керівництво користувача.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища

Електронна комерція — це форма комерційної взаємодії, що здійснюється з використанням електронних мереж. В Україні правові засади такої діяльності визначаються Законом України «Про електронну комерцію» [1]. У межах цієї роботи електронна комерція розглядається у прикладному аспекті — як процес купівлі товарів через інтернет-магазин із використанням вебінтерфейсу, електронного каталогу, кошика, системи оформлення замовлення та інтеграцій із зовнішніми сервісами.

Інтернет-магазин не обмежується відображенням переліку товарів. Його клієнтська частина забезпечує навігацію каталогом, пошук, фільтрацію, перегляд характеристик, взаємодію з кошиком, авторизацію користувача та оформлення замовлення. Серверна частина відповідає за збереження даних, перевірку наявності товарів, обробку замовлень, авторизацію, роботу з оплатою та доставкою. Отже, фронтенд є тим рівнем системи, через який користувач безпосередньо взаємодіє з функціями магазину.

Сегмент електроніки має низку особливостей, які впливають на вимоги до інтерфейсу. Товари цієї категорії мають велику кількість технічних параметрів: тип екрана, обсяг пам’яті, процесор, ємність акумулятора, підтримувані стандарти зв’язку тощо. Тому сторінка товару повинна подавати характеристики структуровано, а каталог має підтримувати фільтрацію за основними параметрами.

У межах однієї товарної категорії може бути значна різниця в ціні, тому фільтрація за вартістю, брендом і характеристиками є одним із ключових інструментів навігації. Для користувача важливо не лише бачити перелік товарів, а й швидко звузити вибір до моделей, які відповідають його бюджету та технічним вимогам.

Для українського ринку важливо також враховувати поширені способи доставки та оплати. У формі оформлення замовлення користувач повинен мати

можливість швидко обрати місто, відділення або поштомат служби доставки, а також зручний спосіб оплати. Це зменшує кількість ручного введення та знижує ризик помилок під час оформлення замовлення.

Після 2022 року на умови використання інтернет-магазинів вплинули перебої з електропостачанням і нестабільний доступ до мережі. Частина сесій відбувається через мобільний інтернет або переривається під час оформлення замовлення. Тому клієнтська частина повинна коректно зберігати стан кошика, обробляти помилки мережі та не втрачати введені дані користувача.

Поведінка покупця електроніки зазвичай не обмежується одним відвідуванням сайту. Користувач може переглянути кілька моделей, порівняти характеристики, повернутися до товару пізніше або відкласти його у список бажань. Через це у TechStore доцільно реалізувати список бажань, порівняння товарів і блок нещодавно переглянутих позицій.

Кількісні характеристики ринку також впливають на проєктні рішення. За звітами EVO і Promodo, частка онлайн-продажів у роздрібному обігу України перевищила 10 %, а сегмент електроніки та побутової техніки традиційно входить до трьох найбільших за оборотом [2, 3]. Понад дві третини відвідувань українських інтернет-магазинів припадає на мобільні пристрої, тому адаптивність і швидкість роботи на смартфоні є базовою вимогою. Близько 80 % замовлень доставляються через сервіс «Нова пошта», тому інтерфейс оформлення замовлення повинен зручно підтримувати цей сценарій доставки [3].

У роботі виділено три типові сценарії: пошук конкретної моделі, вибір товару в межах категорії та ознайомлювальний перегляд каталогу. Перший сценарій потребує швидкого пошуку, другий — гнучких фільтрів і порівняння, третій — добірок, рекомендацій та зрозумілої навігації. Ці сценарії використано як основу для подальшого формування вимог до TechStore.

Загальну логіку роботи системи — хто що може робити і який набір дій доступний кожному типу користувача — відображено на рисунку 1.1 у вигляді UML-діаграми використання.

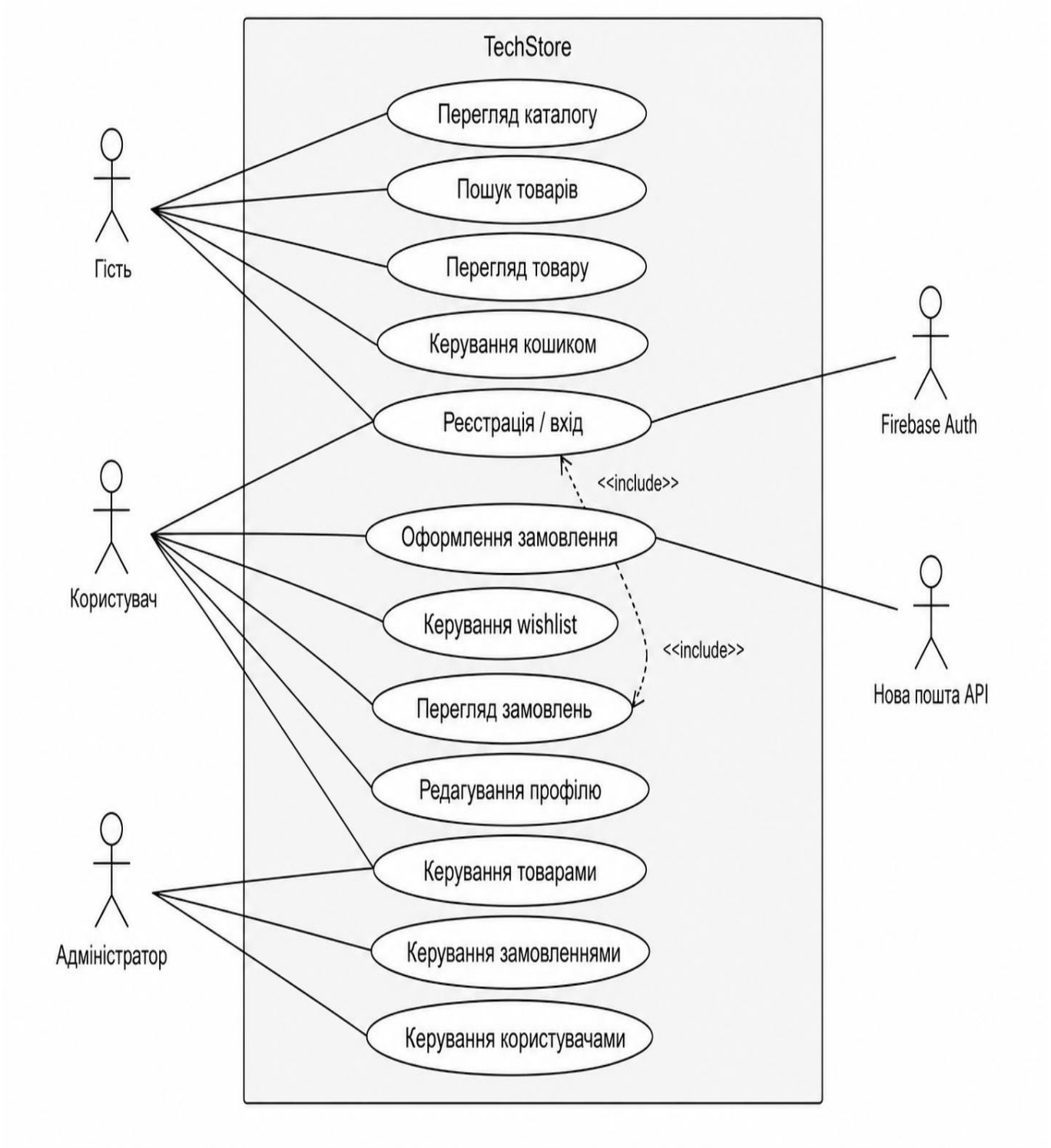


Рис. 1.1. Діаграма використання інтернет-магазину TechStore

Серед дієвих осіб — гість, зареєстрований користувач, адміністратор, а також зовнішні системи: платіжний шлюз, служба доставки. Адміністративний інтерфейс принципово відрізняється від клієнтського: йому не потрібна мінімалістичність — тут навпаки зручніше мати щільні таблиці, масові дії, розгорнуті фільтри.

Описані особливості предметного середовища визначають функціональні та нефункціональні вимоги до TechStore і є основою для подальшого аналізу аналогів.

1.2. Огляд наявних аналогів

На українському ринку є кілька великих гравців: Rozetka, Comfy, Foxtrot, Eldorado, Citrus, «Епіцентр». Детально розглянуто три магазини — Rozetka, Comfy і Foxtrot. Вони охоплюють різні підходи: від маркетплейсу зі збірним асортиментом до спеціалізованої мережі з власним каталогом.

1.2.1. Rozetka

Rozetka є одним із найвідоміших українських онлайн-майданчиків і функціонує не лише як інтернет-магазин, а й як маркетплейс. Для цієї роботи Rozetka розглядається як приклад складного каталогу з великою кількістю категорій, фільтрів, продавців і сценаріїв взаємодії користувача з товаром.

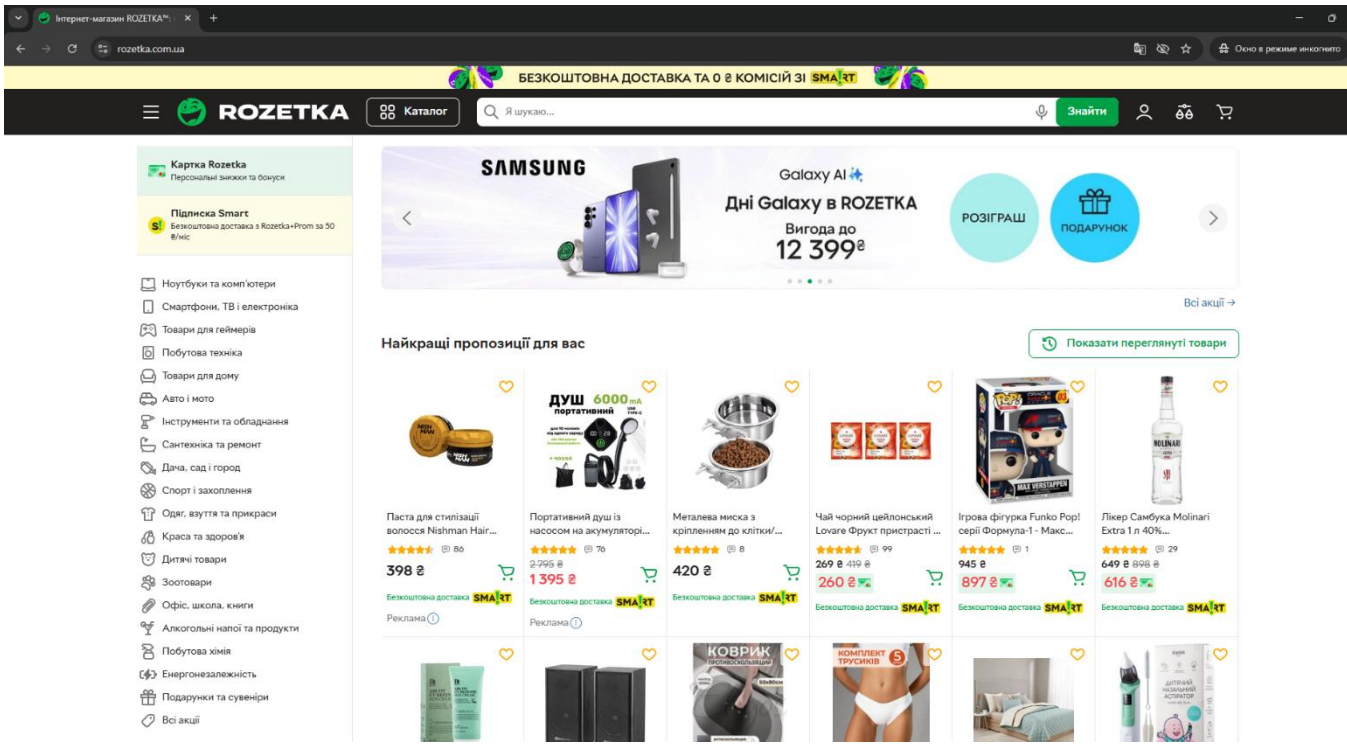


Рис. 1.2. Головна сторінка інтернет-магазину Rozetka

Структура сторінок Rozetka відповідає типовій моделі великого маркетплейсу: ліва частина інтерфейсу використовується для категорій і фільтрів, центральна частина — для банерів, добірок і сітки товарів, а верхня панель — для пошуку та

сортування. До сильних рішень цього інтерфейсу належить інтеграція фільтрів з URL, завдяки чому користувач може зберегти або передати посилання на конкретну вибірку. Додатковою перевагою є відображення кількості товарів біля окремих фільтрів, що дає змогу оцінити результат фільтрації до її застосування.

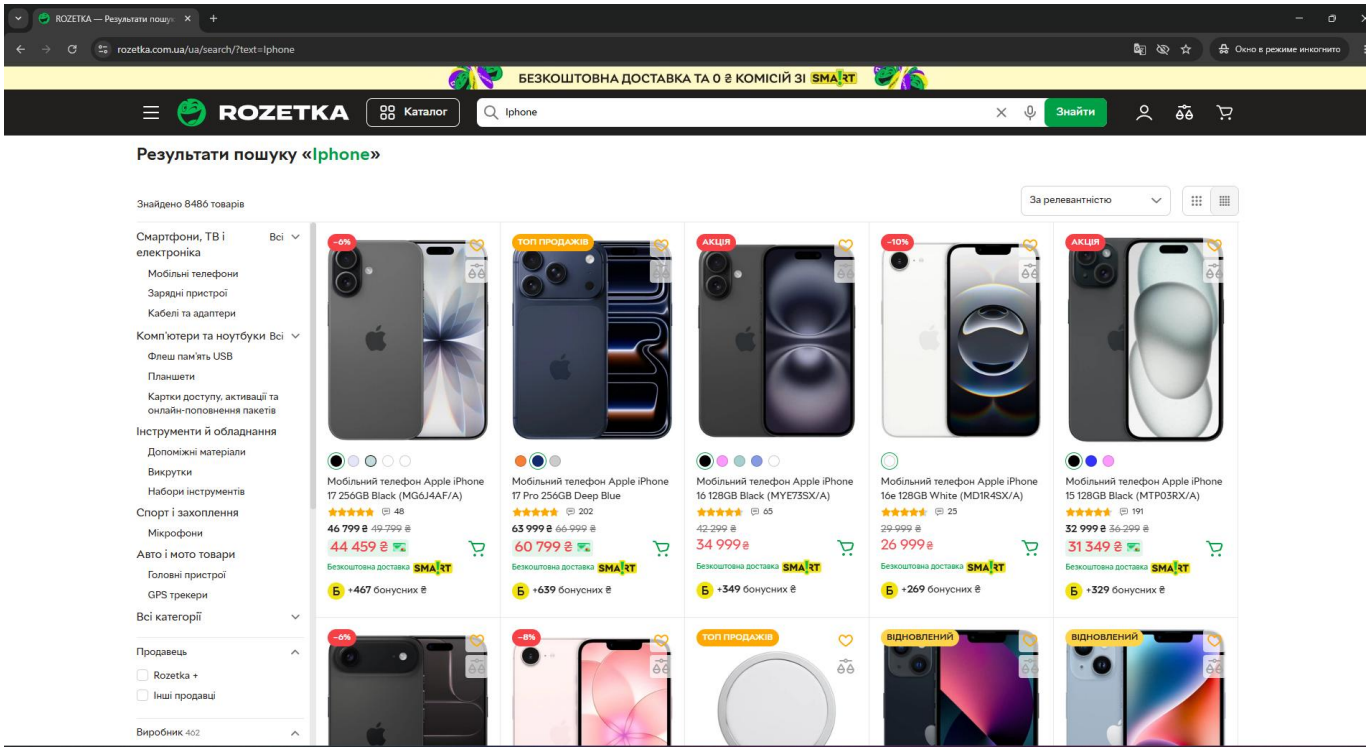


Рис. 1.3. Сторінка категорії на Rozetka

Картка товару містить галерею зображень, характеристики, опис, відгуки, блок запитань і відповідей, рекомендації, інформацію про доставку та наявність у точках продажу. Оформлення замовлення реалізовано як послідовний сценарій із можливістю повернення до попередніх етапів.

До недоліків такого підходу можна віднести високе навантаження сторінки на слабких пристроях або за умов повільного з’єднання. За даними Google, збільшення часу завантаження мобільної сторінки з однієї до п’яти секунд підвищує ймовірність відмови на 90 % [5]. Отже, під час розробки TechStore необхідно враховувати баланс між функціональною насиченістю інтерфейсу та швидкістю його роботи.

1.2.2. Comfy

Comfy — класична мережа електроніки і побутової техніки. Не маркетплейс, власний асортимент, упізнаваний помаранчевий колір. Головна сторінка орієнтована

на акції і кредитні пропозиції — це основний комерційний двигун мережі, і він добре видно в інтерфейсі.

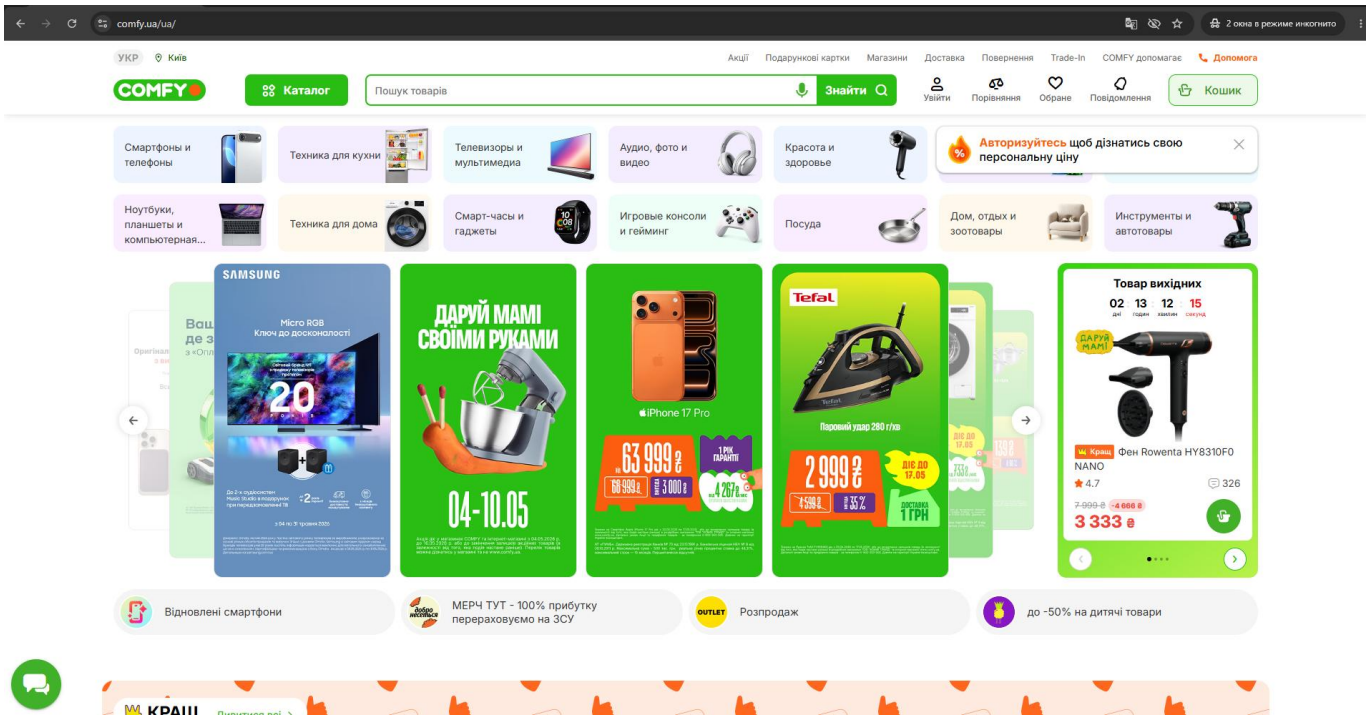


Рис. 1.4. Головна сторінка Comfy

Система фільтрів у Comfy є менш деталізованою порівняно з Rozetka, однак інтерфейс каталогу завдяки цьому виглядає менш перевантаженим. Картка товару акцентує увагу на кредитних пропозиціях і бонусах, що відповідає комерційній моделі мережі та сценаріям покупки техніки у розстрочку.

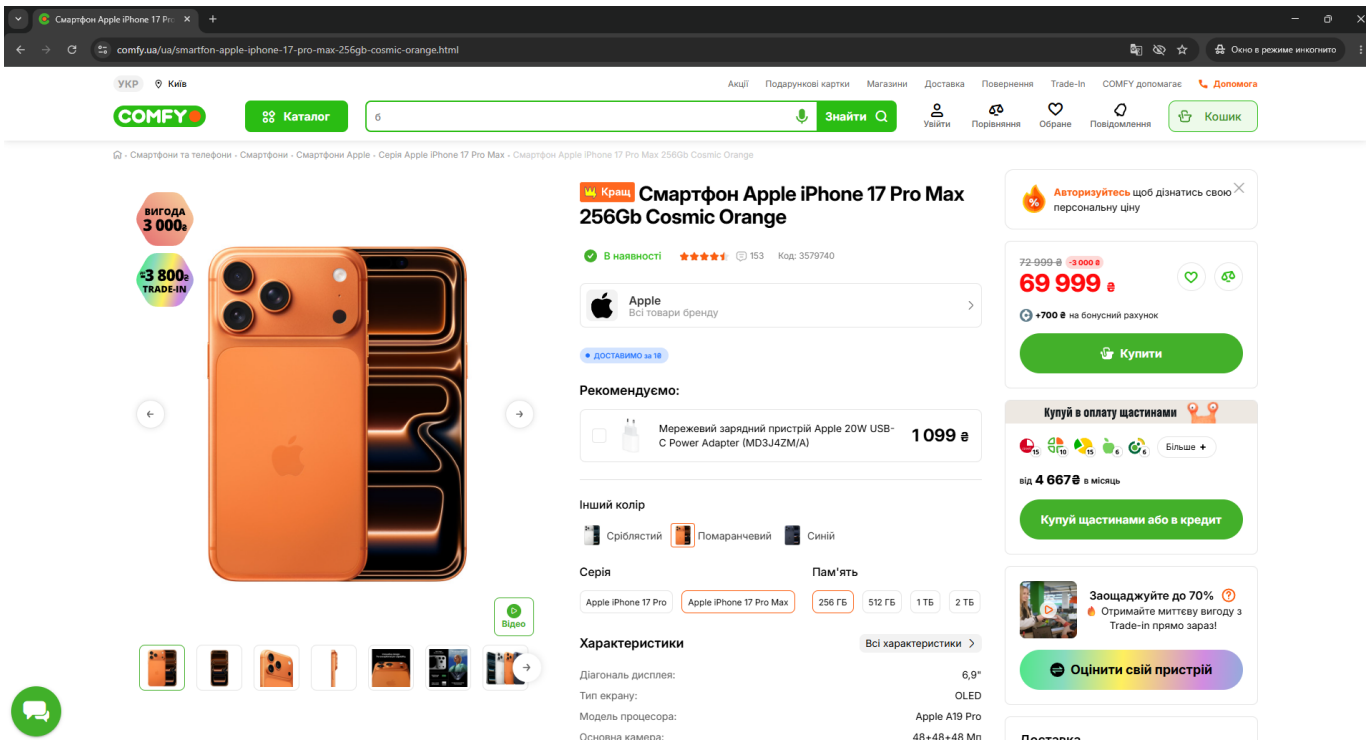


Рис. 1.5. Картка товару Comfy

Сильною стороною Comfy є мобільна версія інтерфейсу. Нижня навігаційна панель забезпечує швидкий доступ до головної сторінки, каталогу, кошика, кабінету та пошуку. Такий підхід доцільно врахувати під час розробки TechStore, оскільки він скорочує кількість дій для користувача на смартфоні.

До недоліків інтерфейсу Comfy можна віднести помітну кількість рекламних спливаючих повідомлень: пропозиції підписки, кредитні акції та повторні повідомлення в межах однієї сесії. За дослідженнями з юзабіліті [6], надмірне використання таких елементів може знижувати лояльність користувача, тому в TechStore доцільно застосовувати сповіщення помірно й лише у функціонально виправданих випадках.

1.2.3. Foxtrot

Foxtrot будував свою онлайн-присутність довго і помітно. Головна сторінка відмовляється від постійної бічної колонки категорій на користь широкої сітки — більш сучасний вигляд, але трохи глибша навігація до потрібного розділу.

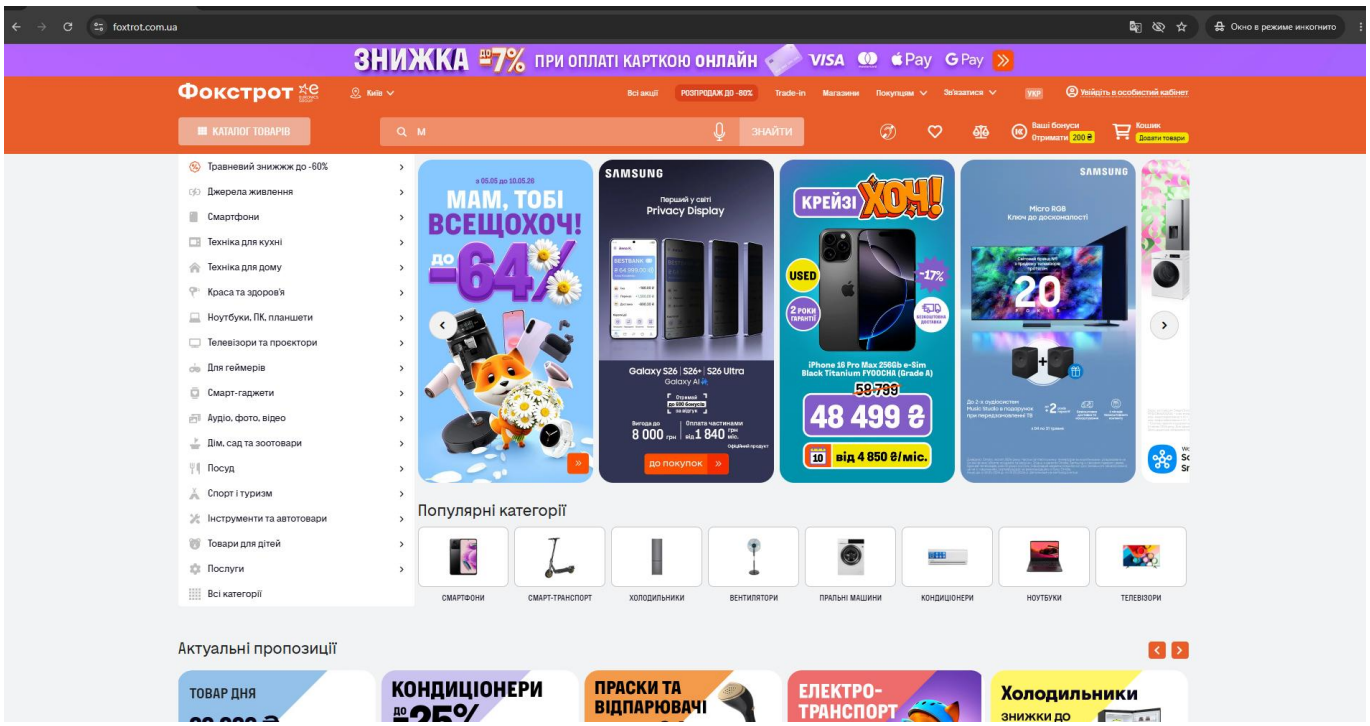


Рис. 1.6. Головна сторінка Foxtrot

Окремої уваги заслуговує реалізація мобільних фільтрів у Foxtrot. На десктопній версії фільтри подано у вигляді бічної панелі, тоді як на мобільних пристроях вони відкриваються в окремому блоці. Це дозволяє адаптувати один і той самий функціональний елемент до різних типів пристроїв без втрати зручності.

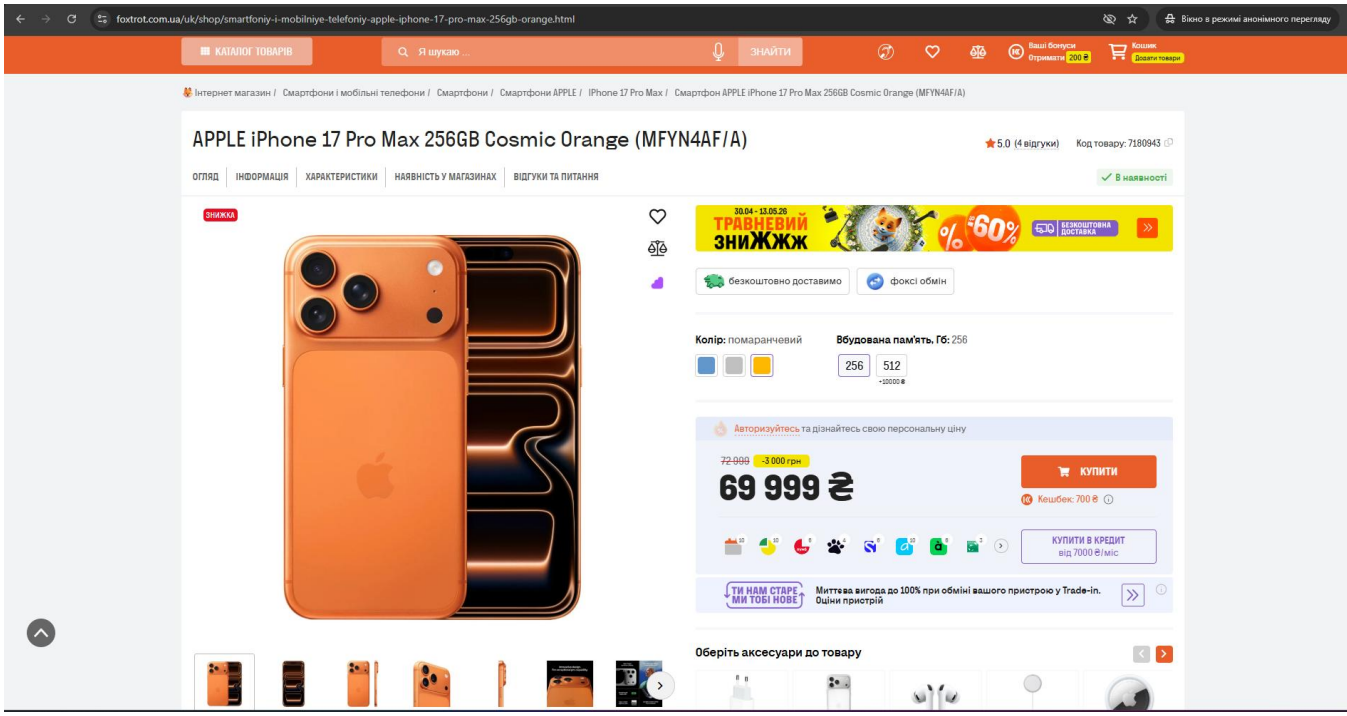


Рис. 1.7. Картка товару Foxtrot

До недоліків інтерфейсу Foxtrot можна віднести навігацію між вкладками всередині картки товару. Для перемикання між розділами опису, характеристик і відгуків користувачеві потрібно повертатися до верхньої частини сторінки. Такий сценарій збільшує кількість зайвих дій і демонструє важливість юзабіліті-тестування навіть для базових елементів інтерфейсу.

1.2.4. Узагальнення результатів аналізу аналогів

Для узагальнення результатів огляду аналогів сформовано порівняльну таблицю за ключовими критеріями, які впливають на користувацький досвід і можуть бути враховані під час розробки TechStore.

Таблиця 1.1 — Порівняльний аналіз інтерфейсних рішень інтернет-магазинів-аналогів

Критерій	**Rozetka**	**Comfy**	**Foxtrot**
Тип каталогу	Маркетплейс	Власний асортимент	Власний асортимент
Бічна категоризація	Так (постійна)	Лише в меню	Лише в меню
Інтеграція фільтрів з URL	Так	Часткова	Так
Лічильник результатів у фільтрі	Так	Ні	Так
Швидкий випадний кошик	Так	Так	Так
Покрокове оформлення замовлення	Так	Так	Так
Список бажань	Так	Так	Так
Порівняння товарів	Так	Так	Так
BottomNav на мобільному	Частково	Так	Ні
Темна тема	Ні	Ні	Ні
Видимі попапи (агресивність)	Помірна	Висока	Помірна

Критерій	Rozetka	Comfy	Foxtrot
Швидкість завантаження (суб’єктивно)	Середня	Висока	Висока
Якість мобільної версії	Добра	Дуже добра	Добра

Аналіз таблиці 1.1 показує, що всі розглянуті інтернет-магазини мають базовий набір функцій, необхідний для електронної комерції: швидкий кошик, список бажань, порівняння товарів та покрокове оформлення замовлення. Водночас між ними є помітні відмінності у способах організації каталогу, фільтрації та мобільної навігації. Rozetka має найрозвиненішу систему фільтрів і постійну бічну категоризацію, що є доцільним для маркетплейсу з великою кількістю товарів. Comfy вирізняється якісною мобільною навігацією, зокрема наявністю BottomNav, що спрощує доступ до основних розділів на смартфоні. Foxtrot демонструє зручну реалізацію фільтрів, однак не використовує нижню мобільну навігацію. Спільним недоліком трьох аналогів є відсутність темної теми. Це створює можливість для TechStore використати темний режим як додаткову перевагу з погляду зручності користування.

За результатами порівняння аналогів визначено інтерфейсні рішення, які доцільно використати у TechStore. До таких рішень належать збереження параметрів фільтрації в URL, відображення кількості товарів для окремих фільтрів, нижня навігаційна панель у мобільній версії, адаптивна панель фільтрів і покрокове оформлення замовлення.

Водночас у розглянутих аналогах виявлено рішення, яких доцільно уникати. До них належать надмірна кількість рекламних попапів, перевантаження картки товару маркетинговими блоками, ускладнена навігація між вкладками товару та недостатня оптимізація важких графічних елементів на мобільних пристроях. Для TechStore пріоритетними мають бути зрозуміла структура, швидке завантаження, мінімальна кількість зайвих дій і стабільна робота основних сценаріїв.

1.3. Постановка задачі

На основі аналізу ринку і огляду аналогів сформульовано вимоги до TechStore. Вимоги розбиті на дві групи: функціональні (що саме система повинна робити) і нефункціональні (яку якість виконання вона має забезпечити).

1.3.1. Функціональні вимоги

Таблиця 1.2 — Основні функціональні вимоги до TechStore

Група вимог	Зміст вимоги	Очікуваний результат
Каталог	Перегляд категорій, фільтрація, сортування, пагінація	Користувач швидко знаходить потрібний товар
Пошук	Швидкий пошук із підказками під час введення	Скорочення часу переходу до товару
Картка товару	Галерея, характеристики, опис, відгуки, рекомендації	Користувач отримує повну інформацію перед покупкою
Кошик	Додавання, видалення, зміна кількості, збереження між сесіями	Склад кошика не втрачається під час повторного входу
Оформлення	Односторінкове оформлення замовлення з вибором доставки, способу оплати і застосуванням промокоду	Замовлення оформлюється без зайвих дій
Кабінет	Профіль користувача, зміна пароля, перегляд замовлень, список бажань та бонусна інформація	Користувач керує власними даними та переглядає історію взаємодії
Адміністрування	Керування товарами, категоріями, замовленнями, користувачами	Адміністратор підтримує актуальність каталогу

Каталог:

- відображення головної сторінки з банерами, добірками популярних категорій та виокремленими товарами;
- перегляд каталогу за категоріями та підкатегоріями з пагінацією;
- фільтрація товарів за брендом, ціною, наявністю;
- сортування за ціною, популярністю, новизною;

- збереження стану фільтрів і сортування в URL;
- швидкий пошук з підказками при введенні.

Картка товару:

- галерея зображень з можливістю збільшення;
- характеристики, опис, відгуки;
- додавання у кошик, список бажань або порівняння;
- рекомендовані супутні та аналогічні товари.

Кошик і замовлення:

- редагування складу кошика — кількість, видалення;
- односторінкове оформлення замовлення з вибором доставки через Нову пошту або кур'єрську доставку, а також оплатою при отриманні або картою;
- підтримка промокодів;
- збереження кошика між сесіями.

Авторизація:

- вхід і реєстрація за email/паролем;
- вхід через Google;
- відновлення пароля.

Особистий кабінет:

- список замовлень з деталями і статусами;
- список бажань;
- редагування профілю, адреси доставки.

Адміністрування:

- дашборд зі статистикою;
- керування товарами, категоріями, замовленнями, користувачами, купонами.

1.3.2. Нефункціональні вимоги

Таблиця 1.3 — Основні нефункціональні вимоги до TechStore

Категорія	Вимога	Критерій перевірки
Продуктивність	LCP до 2,5 с, INP до 200 мс, CLS до 0,1	Lighthouse, PageSpeed Insights
Розмір клієнтського коду	JavaScript-бандл головної сторінки до 250 КБ після стиснення	Аналізатор збірки
Адаптивність	Коректна робота на ширині екрана від 360 до 1920 px	Ручне тестування, DevTools
SEO	Індексація сторінок товарів і категорій, schema.org, title, description, canonical URL	Перевірка HTML-розмітки
Безпека	HTTPS, захищені заголовки, відсутність секретів у NEXT_PUBLIC-змінних	Перевірка конфігурації
Доступність	Контрастність 4,5:1 для основного тексту, тач-цілі не менше 44 × 44 px	axe DevTools, ручна перевірка
Сумісність	Останні дві версії Chrome, Firefox, Safari, Edge	Кросбраузерне тестування

Продуктивність. Метрики Core Web Vitals: LCP не більше 2,5 с, INP не більше 200 мс, CLS не більше 0,1 на сторінці каталогу при широкосмуговому з’єднанні [4]. Саме ці метрики використовуються як основні орієнтири для оцінювання продуктивності та стабільності інтерфейсу. Вага JavaScript-бандла головної сторінки — до 250 КБ після стиснення.

Адаптивність. Коректна робота від 360 до 1920 пікселів. Всі ключові дії — пошук, кошик, оформлення — доступні без переходу на окрему «мобільну» версію.

SEO. Сторінки каталогу і товарів відкриті для індексації. Структуровані дані schema.org. Унікальні title, description, canonical URL, Open Graph на кожній сторінці.

Безпека. HTTPS обов'язково. Заголовки X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Permissions-Policy. Токени — через захищені заголовки, без збереження секретів у NEXT_PUBLIC_-змінних.

Юзабіліті. Відповідність WCAG 2.1 рівня A та базовим вимогам доступності. Для основного тексту інтерфейсу має забезпечуватися контрастність не менше 4,5:1 [10]. Розміри тач-цілей на мобільному — не менше 44 × 44 CSS-пікселів [11].

Решта: підтримка останніх двох версій Chrome, Firefox, Safari, Edge; інтерфейс українською з технічною можливістю розширення; цільовий uptime 99,5 %.

1.3.3. Загальна постановка задачі

Спроєктувати та реалізувати клієнтську частину TechStore, яка забезпечує: швидкий доступ до каталогу товарів з гнучкою фільтрацією і сортуванням (архітектура передбачає масштабування каталогу до понад тисячі позицій; на момент здачі роботи продакшн-каталог містить 51 товар); повноцінні картки товару з усією потрібною інформацією і функціями взаємодії; односторінкове оформлення замовлення з підтримкою актуальних для України способів доставки і вибором способу оплати; безпечну авторизацію через email/пароль і Google; профіль користувача, список замовлень, wishlist, порівняння товарів та адміністративний інтерфейс. Серверна частина, база даних і зовнішні інтеграції розглядаються як допоміжні компоненти, необхідні для демонстрації роботи фронтенду.

1.3.4. Критерії успішності виконання роботи

Для об’єктивного оцінювання результату роботи заздалегідь сформульовано критерії успішності, які охоплюють функціональну повноту, відповідність нефункціональним вимогам, стабільність, придатність до розвитку та задокументованість.

Перший критерій — функціональна повнота. Функціональні вимоги з підрозділу 1.3.1 мають бути реалізовані й перевірені у типових користувацьких сценаріях. Контрольний сценарій охоплює пошук товару, перегляд картки, додавання в кошик, оформлення замовлення, перегляд підтвердження та перевірку статусу замовлення в особистому кабінеті.

Другий — відповідність нефункціональним вимогам. LCP до 2,5 с, INP до 200 мс, CLS до 0,1, бандл до 250 КБ, контрастність 4,5:1, тач-цілі 44 × 44 пікселя. Перевіряється через Lighthouse і axe DevTools.

Третій критерій — стабільність. Помилка мережі, помилка сервера або завершення терміну дії сесії мають оброблятися без втрати вже введених користувачем даних і без критичного порушення роботи інтерфейсу.

Четвертий критерій — придатність до розвитку. Структура проєкту має дозволяти додавати нові категорії товарів, поля форм і маршрути без суттєвого переписування наявних модулів.

П'ятий критерій — задокументованість і відтворюваність. Керівництво користувача покриває базові сценарії; вихідний код містить README та npm-скрипти для запуску, тестування, lint-перевірки і продакшн-збірки. Розгортання фронтенду організовано через Vercel, бекенду — через Railway.

Висновки до розділу 1

У першому розділі сформовано основу для подальшого проєктування системи. Проаналізовано предметне середовище: частка онлайн-продажів перевищила 10 %, сегмент електроніки — серед трьох найбільших за оборотом, більше двох третин трафіку — мобільні пристрої, 80 % доставок — «Нова пошта». Виявлено специфіку 2022–2024 років: регіональна міграція покупців і нестабільне електропостачання як нові умови експлуатації.

Розглянуто три магазини — Rozetka, Comfy і Foxtrot. Відібрано практики для запозичення: URL-інтеграція фільтрів з лічильниками результатів, BottomNav на мобільному, окрема мобільна панель фільтрів знизу та зручне оформлення замовлення. Жоден із трьох не підтримує темну тему — пряма конкурентна можливість для TechStore. Визначено антипатерни, від яких доцільно утримуватись: агресивні попапи, перевантажені картки, незручна вкладкова навігація.

Сформульовано вимоги: 6 груп функціональних і 8 категорій нефункціональних з конкретними цільовими значеннями. LCP 2,5 с, INP 200 мс, CLS 0,1, бандл 250 КБ, контрастність 4,5:1, тач-цілі 44 × 44 пікселя, uptime 99,5 %. Сформульовано 5 критеріїв успішності виконання роботи.

Усі ці вимоги визначають образ TechStore: магазин середнього масштабу, орієнтований на умови українського ринку, з акцентом на мобільну зручність, швидкість і функціональну повноту. Наступним етапом є проєктування системи та опис її алгоритмічного забезпечення.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

Проєктування клієнтської частини TechStore потребує попереднього визначення даних, з якими працює інтерфейс: які дані надходять від API, які формуються користувачем, які тимчасово зберігаються в браузері та які передаються назад на сервер.

У вузькому сенсі фронтенд інтернет-магазину — це програма-споживач API. Вона отримує дані від бекенду, відображає їх користувачу, збирає введення, формує запити на зміну стану і відправляє їх назад. Уся бізнес-логіка (наприклад, перевірка наявності, розрахунок остаточної ціни з усіма знижками і податками, створення транзакції) перебуває на бекенді. Фронтенд робить копію частини інформації для відображення, кешує її в локальному стані, але не є джерелом істини щодо більшості сутностей.

Виняток становлять три типи даних, які природно живуть на клієнті:

- гостьовий кошик — поки користувач не авторизувався, склад кошика не відомий бекенду; зберігається у локальному сховищі браузера й переноситься на бекенд після входу в обліковий запис;
- список бажань реалізовано через серверний API для авторизованих користувачів; для неавторизованого користувача дія додавання до wishlist вимагає входу в систему;
- історія нещодавно переглянутих товарів — клієнтський механізм, який зберігає повні об’єкти Product у localStorage через Zustand persist з лімітом 12 останніх товарів.

Для всіх інших сутностей (товари, замовлення, відгуки, користувацький профіль) фронтенд є тонким клієнтом — звертається до API за актуальними даними і відображає їх.

2.1.1. Основні сутності системи

У TechStore фігурують такі основні сутності.

Product (товар) — центральна сутність каталогу. Поля: ідентифікатор, slug для людиночитного URL, назва, опис, перелік зображень, ціна (поточна, стара), бренд, категорія, наявність, кількість на складі, набір технічних характеристик у форматі ключ-значення, рейтинг, кількість відгуків, набір бейджів («хіт», «новинка», «акція»). Технічні характеристики мають специфічну для кожної категорії структуру: для смартфонів це один набір полів, для холодильників — зовсім інший. Для уніфікованого зберігання використано підхід EAV (Entity-Attribute-Value), коли характеристики зберігаються як набір пар, а не як фіксована схема.

Category (категорія) — дерево категорій. Поля: ідентифікатор, slug, назва, посилання на батьківську категорію (для побудови ієрархії), список доступних фільтрів (які характеристики мають бути показані як фільтри в каталозі цієї категорії), мета-теги для SEO. Категорії можуть бути будь-якого рівня вкладеності: «Електроніка → Смартфони → Apple → iPhone 15 серія».

User (користувач) — обліковий запис покупця або адміністратора. Поля: ідентифікатор, email, ім'я, прізвище, телефон, роль (user / admin), дата реєстрації, налаштування сповіщень. Паролі зберігаються виключно на бекенді у захешованому вигляді, фронтенду вони ніколи не передаються.

Cart (кошик) — поточний склад кошика користувача. Поля: список елементів {product, quantity}, стан завантаження та ознака авторизації. На клієнті кошик зберігається через Zustand persist під ключем cart-storage. Для авторизованого користувача кошик синхронізується з серверним API.

Order (замовлення) — оформлене замовлення. Поля: ідентифікатор, номер, дата створення, статус замовлення (очікує, в обробці, відправлений, доставлений, скасований) та окремо статус оплати (очікує, оплачено, відхилено, повернено), список позицій (товар, кількість, ціна на момент замовлення), спосіб і адреса доставки, спосіб оплати, загальна сума, опційний промокод, коментар покупця. На

відміну від кошика, після створення замовлення цей запис стає незмінним для користувача — він може лише переглядати статус і деталі.

Review (відгук) — відгук користувача про товар. Поля: ідентифікатор, посилання на товар, посилання на користувача, оцінка (1-5), текст, дата, статус модерації. Відгуки відображаються на сторінці товару після проходження модерації, причому залишити відгук може лише користувач, який раніше придбав цей товар, — це підвищує довіру до оцінок і ускладнює накрутку.

Coupon (купон) — промокод знижки. Поля: код, тип знижки (відсоток або фіксована сума), значення знижки, дата дії від і до, опційний мінімальний обсяг замовлення, опційне обмеження за категоріями.

Окремо існують клієнтські колекції, що доповнюють роботу основних сутностей: RecentlyViewedItem зберігає повні об’єкти Product у localStorage з лімітом 12 позицій; ComparisonItem може зберігатися локально для гостя і синхронізуватися через API для авторизованого користувача; WishlistItem працює через API та потребує авторизації.

Для роботи з основними сутностями на стороні клієнта використовуються TypeScript-інтерфейси, розташовані в каталозі src/types. Інтерфейс Product містить поля _id, name, slug, description, shortDescription, price, comparePrice, category, brand, sku, images, stock, specifications, features, rating, isActive, isFeatured, isOnSale, warranty, views, createdAt та updatedAt. Інтерфейс Order описує номер замовлення, статус, список позицій, спосіб доставки, спосіб оплати, контактні дані користувача та підсумкову суму. Аналогічні інтерфейси визначено для User, Cart, Review, Coupon і Category. Такий підхід дає змогу узгодити структуру даних між REST API та компонентами клієнтської частини.

2.1.2. Вхідні і вихідні дані

Класифікація даних з точки зору фронтенду:

Вхідні дані з API. Це найбільший потік. Каталог товарів — GET /api/products зі знаходженням за категорією, фільтрами, сортуванням, пагінацією. Деталі товару — GET /api/products/:id (маршрут приймає як ObjectId, так і slug). Список категорій з

ієрархією — GET /api/categories. Профіль авторизованого користувача — GET /api/auth/profile. Список замовлень — GET /api/orders/my-orders. Деталі замовлення — GET /api/orders/:id. Відгуки про товар — GET /api/reviews/product/:productId. У відповідь сервер повертає JSON з даними, що відповідають описаним вище сутностям.

Вхідні дані від користувача. Це запити користувацького введення. Реєстрація і вхід — POST /api/auth/register, POST /api/auth/login. Створення замовлення — POST /api/orders. Залишення відгуку — POST /api/reviews. Зміна профілю — PUT /api/auth/profile. Адміністративні операції виконуються на звичайних ресурсних маршрутах із перевіркою прав (verifyToken + requireAdmin) — наприклад, POST /api/products, PUT /api/orders/:id/status, GET /api/orders/all; окремого простору /api/admin/* немає. Усі дані з форм валідуються на клієнті за схемами Zod (детальніше — у розділі 3) і додатково перевіряються на бекенді.

Вихідні дані. У тому, що стосується власне фронтенду, вихідні дані — це HTML-розмітка, що рендериться у браузері користувача. На рівні мережі — це HTTP-запити до бекенду з даними, описаними вище. На рівні браузера — записи в localStorage (кошик, тема, мова, список бажань, переглянуті).

2.1.3. Архітектура взаємодії клієнт-сервер

TechStore побудований за трирівневою архітектурою. Клієнтська частина — це Next.js-застосунок, розгорнутий на платформі Vercel. Серверна частина реалізована на Node.js з використанням Express. Для роботи з базою даних MongoDB використовується ODM-бібліотека Mongoose. Бекенд розгортається на Railway, а база даних може використовуватися як локально, так і через MongoDB Atlas. Зовнішні сервіси включають Firebase Authentication для Google OAuth та API Нової пошти для довідника міст і відділень. Повноцінна інтеграція з платіжним шлюзом у поточній версії не реалізована; у системі передбачено вибір способу оплати картою або при отриманні.

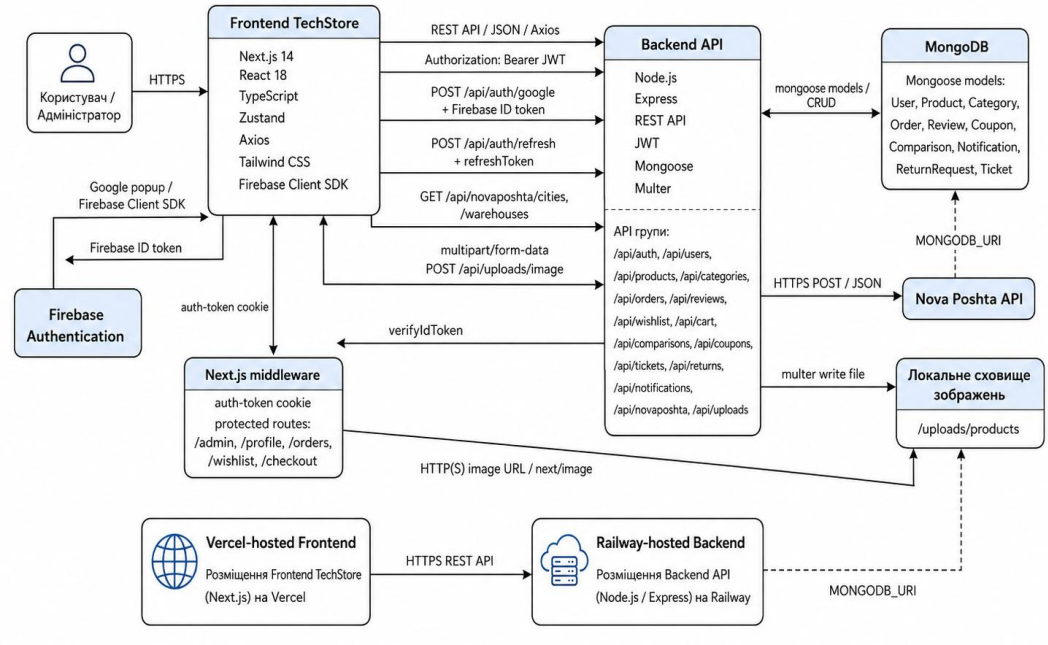


Рис. 2.1. Архітектурна діаграма системи TechStore

Усі звернення фронтенду до бекенду виконуються через HTTPS-канал. Для приватних маршрутів у заголовок Authorization додається JWT access-токен. Запити до Нової пошти виконуються через проксі-маршрути власного бекенду /api/novaposhta, щоб не відкривати службові ключі на клієнті.

2.2. Проєктування системи

2.2.1. Діаграма використання (Use Case)

Загальна діаграма використання TechStore вже наведена в розділі 1 (рисунок 1.1) для опису предметного середовища. Тут же доцільно деталізувати конкретні варіанти використання найбільш складних сценаріїв, що потребують багатоетапної взаємодії з системою.

Сценарій оформлення замовлення (Place Order). Передумова: у кошику є щонайменше один товар. Послідовність: користувач відкриває кошик; перевіряє склад і кількість; вирішує оформити замовлення; обирає спосіб доставки; вводить адресу; обирає спосіб оплати; за потреби вводить промокод; підтверджує замовлення; система валідує дані, перевіряє наявність товарів, відправляє запит на бекенд, отримує підтвердження і перенаправляє на сторінку успіху. Альтернативні потоки: товар став недоступним — користувач отримує попередження і можливість видалити

позицію; промокод недійсний — повідомлення про помилку, можливість продовжити без промокоду; оформлення для неавторизованого користувача — переадресація на сторінку входу /login з поверненням до оформлення після успішної авторизації.

Сценарій додавання товару адміністратором (Add Product). Передумова: користувач увійшов з роллю admin. Послідовність: відкриває розділ /admin/products; натискає «Додати товар»; заповнює форму (назва, категорія, бренд, опис, ціна, характеристики, зображення); зберігає; система валідує дані за схемою Zod, відправляє запит POST /api/products, отримує створений товар, перенаправляє на список товарів. Альтернативи: помилка валідації — підсвічення проблемних полів; конфлікт slug-у — пропозиція скоригувати; помилка завантаження зображень — окреме повідомлення з можливістю повторити.

2.2.2. Концептуальна модель даних MongoDB/Mongoose

Концептуальна модель даних, що використовується в TechStore, представлена у вигляді ER-подібної діаграми документної моделі на рисунку 2.2. Оскільки в проєкті використовується MongoDB, така діаграма відображає не класичні реляційні таблиці, а основні документи, вкладені структури та зв’язки між ними на рівні Mongoose-моделей.

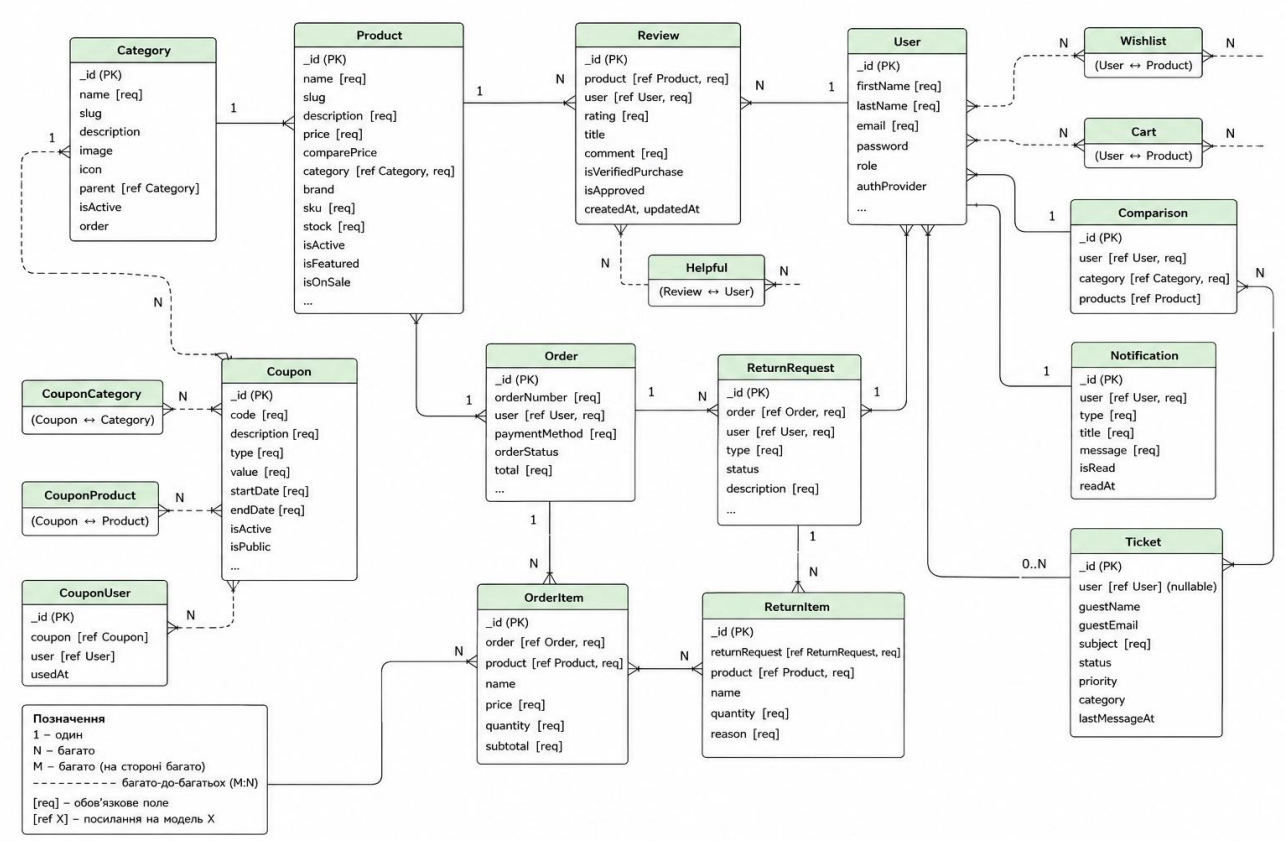


Рис. 2.2. ER-подібна діаграма документної моделі TechStore

Зв’язки між основними документами реалізуються через ідентифікатори та вкладені структури. Користувач може мати багато замовлень; замовлення містить масив позицій OrderItem із посиланнями на товари та фіксацією ціни на момент покупки; товар належить до категорії та має масив specifications; відгук пов’язується з користувачем і товаром; купон може застосовуватися під час створення замовлення. Такий підхід відповідає документній моделі MongoDB і дозволяє зберігати частину пов’язаних даних у вкладеному вигляді.

Хоча збереження даних виконується на серверній частині, фронтенд оперує цими сутностями у вигляді TypeScript-інтерфейсів, що відповідають структурі JSON-відповідей REST API. Завдяки строгій типізації помилки у назвах полів або очікуваних типах виявляються ще на етапі розробки.

2.2.3. Діаграма класів сторів Zustand

Управління станом на клієнті побудовано на бібліотеці Zustand. Стан розбито на сім незалежних сторів, кожен з яких відповідає за свою предметну область. Загальну структуру цих сторів подано на рисунку 2.3.

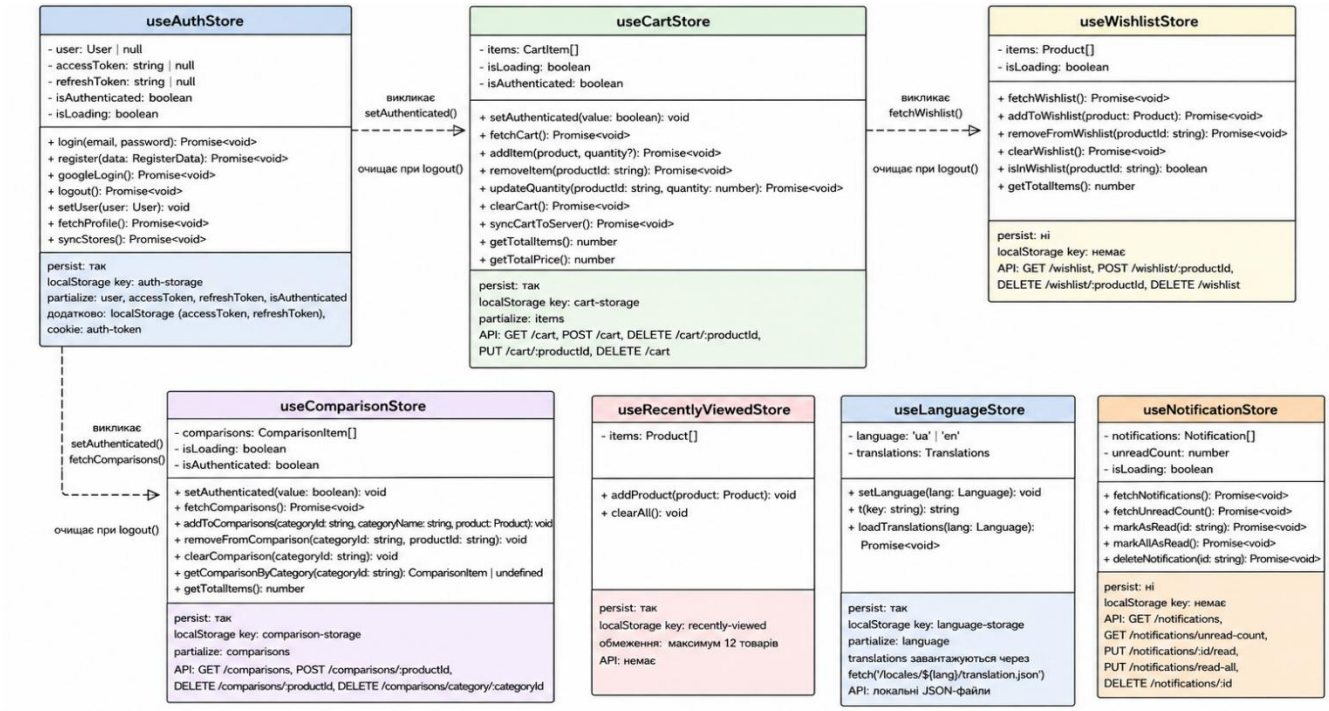


Рис. 2.3. Діаграма класів сторів управління станом

Кожен стор реалізує два набори елементів: поля стану (state) і методи зміни стану (actions). Наприклад, cartStore має поля items, isLoading, isAuthenticated і методи setAuthenticated, fetchCart, addItem, removeItem, updateQuantity, clearCart, syncCartToServer, getTotalItems, getTotalPrice. WishlistStore містить items, isLoading і методи fetchWishlist, addToWishlist, removeFromWishlist, clearWishlist, isInWishlist, getTotalItems.

Між сторами здебільшого немає прямих посилань, що мінімізує зв'язаність. Якщо одній дії потрібно вплинути на кілька сторів (наприклад, при logout треба очистити кошик), використовується звернення до статичного методу getState іншого стору з тіла action: useCartStore.getState().clearCart(). Це уникає циклічних залежностей між модулями.

2.2.4. Архітектура клієнтської частини

Клієнтська частина TechStore побудована як Next.js-застосунок з App Router. Маршрутизація реалізована через файлову структуру каталогу src/app. Частина сторінок використовує клієнтську інтерактивність із директивою 'use client', зокрема каталог і картка товару, оскільки вони активно працюють зі станом, ефектами, API-запитами з браузера та інтерактивними елементами.

- Шар маршрутизації (App Router) — структура папки src/app визначає набір URL-маршрутів і їхні layout'и.
- Шар сторінок і layout'ів — компоненти, що рендеряться на маршрутах; інтерактивні сторінки реалізуються як клієнтські компоненти, а глобальний layout відповідає за провайдери, тему, Header, Footer, BottomNav і toast-сповіщення.
- Шар повторно використовуваних компонентів — каталог src/components, що містить компоненти, не прив'язані до конкретного маршруту (Header, Footer, ProductCard, ReviewSection тощо).
- Шар управління станом — каталог src/store з сімома Zustand-сторами.
- Шар сервісного коду — каталог src/lib з налаштуваннями Axios, Firebase, утилітами.
- Шар типів — каталог src/types з TypeScript-інтерфейсами основних сутностей.
- Шар статичних ресурсів — каталог public з зображеннями, іконками, статичними файлами.

Кожен шар розвивається незалежно: новий маршрут не вимагає змін у компонентах, новий компонент не зачіпає рівень даних, новий стор не змінює маршрути. Деталізована структура коду розглядається в розділі 3.

2.2.5. Діаграма послідовностей для сценарію оформлення замовлення

Динаміку взаємодії компонентів доцільно подати через діаграму послідовностей для сценарію оформлення замовлення. У цьому сценарії беруть

участь користувач, сторінка checkout, cartStore, authStore, API-клієнт Axios, серверна частина та зовнішній сервіс Нової пошти.

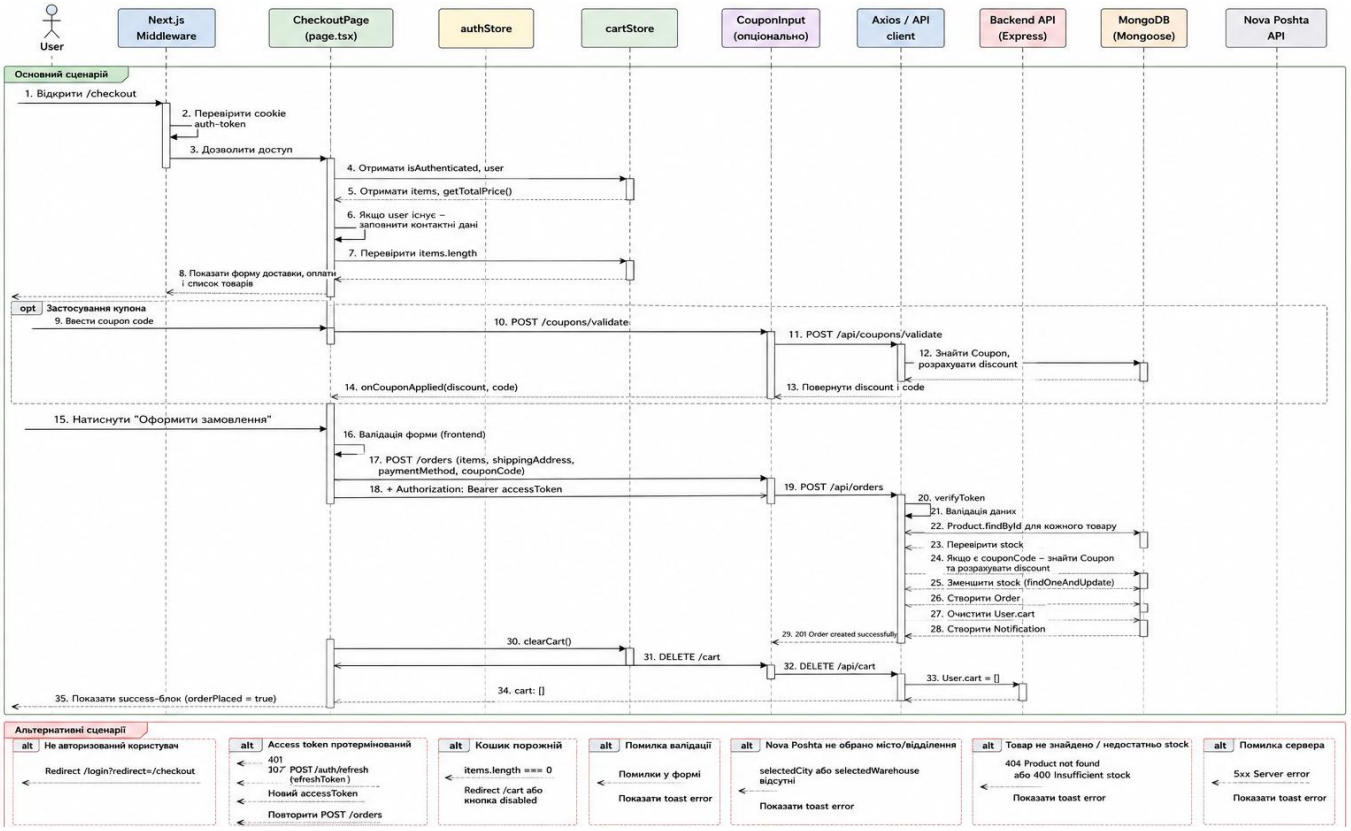


Рис. 2.4. Діаграма послідовностей: оформлення замовлення

Послідовність повідомлень. Користувач відкриває /checkout — Next.js монтує компонент CheckoutPage. У componentDidMount (через useEffect) компонент звертається до cartStore і authStore для отримання вмісту кошика і даних авторизованого користувача (якщо такий є). Якщо кошик порожній — користувач перенаправляється на /cart. Якщо є — компонент рендерить форму, заповнюючи попередньо ім'я, email, телефон з профілю користувача.

Користувач починає вводити місто доставки — компонент відправляє запит GET /api/novaposhta/cities?q=<text> через Axios. Axios через інтерсептор додає JWT-токен, відправляє запит до бекенду. Бекенд звертається до API Нової пошти, отримує список варіантів, повертає його клієнту. Компонент відображає підказки під полем. Користувач обирає місто, далі аналогічно вибирає відділення.

Користувач вибирає спосіб доставки, вводить адресу, вибирає спосіб оплати, опційно вводить промокод. На введення промокоду компонент відправляє запит

POST /api/coupons/validate з кодом і поточним обсягом замовлення. Бекенд перевіряє валідність коду, термін дії, обмеження за обсягом, повертає або помилку (з відповідним повідомленням), або підтвердження зі знижкою. Знижка миттєво відображається у підсумковому блоку.

Користувач натискає «Підтвердити замовлення». Компонент викликає handleSubmit від React Hook Form. Спрацьовує валідація Zod — перевіряються всі поля. Якщо є помилки — submit-обробник не викликається, форма скролиться до першого помилкового поля. Якщо помилок немає — відправляється запит POST /api/orders з повним об'єктом замовлення (склад кошика, дані доставки, оплата). Бекенд створює замовлення в базі, генерує номер, повертає об'єкт Order з ID і номером. Списання залишків товарів виконується атомарно: для кожної позиції застосовується операція findOneAndUpdate з умовою stock ≥ потрібної кількості, тому за одночасних замовлень того самого товару система не допускає продажу більше, ніж є на складі, а в разі недостатнього залишку замовлення відхиляється з відповідним повідомленням. Компонент очищає кошик через cartStore.clearCart(), показує toast «Замовлення створено», перенаправляє на /orders/[id] з підтвердженням.

Альтернативні гілки: помилка мережі під час будь-якого запиту — toast з повідомленням про повтор спроби; помилка 401 — інтерсептор Axios виконує refresh-token-flow, а за невдалого оновлення токенів сесія завершується; недоступний товар у кошику — повідомлення з можливістю видалити позицію; помилка валідації на бекенді (наприклад, ціна змінилася за час оформлення) — повідомлення з пропозицією переглянути.

Отже, сценарій оформлення замовлення охоплює взаємодію кількох частин клієнтської системи: сторінки checkout, глобального стану кошика, стану авторизації, API-клієнта, серверної частини та зовнішніх сервісів доставки. Найважливішими вимогами до цього сценарію є збереження введених даних, обробка помилок мережі, перевірка актуальності товарів і зрозуміле повідомлення користувача про результат кожної дії. Така організація зменшує ризик втрати замовлення та підвищує надійність користувацького сценарію.

2.3. Математичне та алгоритмічне забезпечення

У цьому підрозділі описано алгоритми ключових компонентів клієнтської частини TechStore. Хоча фронтенд інтернет-магазину не потребує складного математичного апарату, у ньому використано низку алгоритмічних рішень, які впливають на функціональність і продуктивність.

2.3.1. Алгоритм роботи кошика (cartStore)

Кошик у TechStore реалізовано через cartStore, який зберігає список позицій items, стан завантаження isLoading та ознаку авторизації користувача isAuthenticated. Для неавторизованого користувача стан кошика зберігається у localStorage через Zustand persist під ключем cart-storage, а для авторизованого користувача додатково синхронізується із серверним API.

Базова операція додавання товару виконується методом addItem. Алгоритм передбачає отримання поточного списку items, перевірку наявності товару з відповідним ідентифікатором, додавання нової позиції або оновлення кількості вже наявної позиції, після чого стан кошика оновлюється через механізм Zustand. Компоненти, підписані на відповідні селектори, автоматично отримують актуальні дані та перерендерюються.

Зміна кількості товару виконується методом updateQuantity. Якщо нове значення quantity більше нуля, у відповідній позиції змінюється кількість. Якщо quantity менше або дорівнює нулю, товар видаляється з кошика, що запобігає відображенню позицій із нульовою кількістю. Для прямого видалення позиції використовується метод removeItem, а для повного очищення кошика — clearCart.

Для авторизованого користувача застосовується метод syncCartToServer, який узгоджує локальний стан кошика із серверною частиною. Це дає змогу зберегти склад кошика між різними сесіями та пристроями. Для розрахунку підсумкових показників використовуються методи getTotalItems і getTotalPrice, які повертають загальну кількість товарів та сумарну вартість позицій у кошику.

Часова складність пошуку позиції в масиві items становить O(n), де n — кількість товарів у кошику. Для типового сценарію інтернет-магазину така складність є прийнятною, оскільки кількість позицій у кошику зазвичай невелика. Використання словникової структури могло б зменшити складність пошуку до O(1), однак у межах цього проєкту масив забезпечує достатню простоту реалізації та зрозумілу структуру стану.

2.3.2. Алгоритм фільтрації каталогу

Сторінка каталогу TechStore підтримує фільтрацію за брендом, ціною та наявністю. Власне фільтрація виконується на бекенді — фронтенд лише формує параметри запиту і відображає результат. Однак сам алгоритм формування URL і його синхронізації з UI заслуговує опису.

Стан фільтрів зберігається у URL у вигляді query-параметрів: /products/laptops?brand=asus,acer&price_from=15000&price_to=40000&page=2. Це дозволяє ділитися посиланням, відкривати в новій вкладці, повертатися до попереднього стану через історію браузера. Алгоритм оновлення фільтра при кліку користувача:

1) Зчитати поточні параметри URL через useSearchParams з Next.js.
2) Створити нову копію через new URLSearchParams(currentParams).
3) Додати/видалити/оновити відповідний параметр.
4) Якщо змінювався фільтр (а не пагінація), скинути page=1.
5) Викликати router.push(`${pathname}?${newParams.toString()}`).
6) Каталог оновлює клієнтський стан і повторно виконує API-запит до бекенду з новими параметрами.

Для діапазону ціни використовується пара полів «від / до» разом із повзунком. На відміну від інших фільтрів, зміна ціни не оновлює URL миттєво — застосування відбувається за окремою дією користувача:

1) Користувач виставляє нижню і верхню межі ціни полями введення або перетягуванням повзунка.

2) Введені значення зберігаються лише в локальному стані компонента і поки що не впливають на запит до бекенду.

3) Натискання кнопки «Застосувати» викликає метод applyPriceFilter.

4) Метод формує оновлені query-параметри price_from і price_to та оновлює URL сторінки каталогу.

5) Зміна URL запускає новий запит до каталогу вже з урахуванням вибраного цінового діапазону.

Такий підхід дає змогу виставити обидві межі та оновити результат одним кліком, без проміжних запитів на кожен крок повзунка, і робить поведінку фільтра передбачуваною. Користувач бачить миттєву реакцію повзунка, а оновлення списку відбувається після того, як він зупиниться на остаточному значенні.

2.3.3. Алгоритм нещодавно переглянутих товарів

Список нещодавно переглянутих товарів реалізовано як клієнтський механізм у recentlyViewedStore. На відміну від серверних сутностей, цей список не потребує обов’язкової синхронізації з бекендом, оскільки використовується для покращення навігації користувача в межах поточної клієнтської сесії.

Під час відкриття сторінки товару до recentlyViewedStore передається повний об’єкт Product. Якщо цей товар уже є у списку, попередній запис видаляється, а актуальний об’єкт переноситься на початок списку. Якщо товар відсутній, він додається першим елементом. Після оновлення список обмежується 12 останніми товарами.

Оновлений список зберігається в localStorage через Zustand persist. Завдяки цьому блок «Нещодавно переглянуті» може відображатися без додаткового запиту до API, оскільки всі необхідні дані про товари вже містяться у локальному стані. Такий підхід зменшує кількість мережевих запитів і прискорює повторне відображення блоку на головній сторінці або в інших частинах інтерфейсу.

Алгоритм також запобігає дублюванню товарів у списку: повторний перегляд тієї самої позиції не створює нового елемента, а лише переміщує товар

на початок. Це забезпечує коректний порядок відображення останніх переглядів і підтримує компактний обсяг локально збережених даних.

2.3.4. Алгоритм пошуку з підказками

Пошук у TechStore має два основні сценарії: швидкий пошук у шапці сайту та пошук у каталозі разом із фільтрацією. У Header використовується live-search із debounce-затримкою приблизно 280 мс і запитом до endpoint /products/autocomplete. Це зменшує кількість запитів під час введення та водночас забезпечує швидке відображення підказок.

Алгоритм роботи швидкого пошуку передбачає оновлення локального стану searchQuery після кожної зміни введення, запуск таймера debounce, виконання запиту до /products/autocomplete?q=... після завершення затримки та відображення результатів у випадаючому списку. Підказки містять основні дані товару: мініатюру, назву та ціну. Натискання на підказку відкриває сторінку товару, а натискання Enter переводить користувача на сторінку каталогу з параметром пошуку /products?search=....

Для сторінки каталогу використовується інша логіка. Під час зміни фільтрів, сортування, сторінки пагінації або пошукового параметра формується новий API-запит до каталогу. Якщо попередній запит ще не завершився, він скасовується через AbortController. Це запобігає ситуації, коли застаріла відповідь API перезаписує актуальний стан каталогу.

Поєднання debounce у шапці сайту та AbortController у каталозі дає змогу зменшити навантаження на API, уникнути конфліктів асинхронних відповідей і підтримувати передбачувану поведінку інтерфейсу під час швидкої взаємодії користувача з пошуком і фільтрами.

Тип збігу, який застосовується на рівні бекенду, — префіксний пошук за полем name з використанням регулярного виразу MongoDB (оператор $regex з опцією $options: 'i' для нечутливості до регістру). Це означає, що запит «sam» поверне «Samsung Galaxy» і «Samsung Tab», але не «Asus Zenfone», — рішення свідомо прийнято на користь точності, оскільки користувач інтернет-магазину

зазвичай знає початок назви або бренду. Кількість результатів обмежена до восьми позицій через limit(8) на рівні запиту, що запобігає перевантаженню випадаючого списку і зменшує обсяг переданих даних.

На рівні клієнта результати окремо не кешуються між сесіями — кожен новий рядок пошуку спричиняє новий запит до сервера. Водночас у межах однієї сесії повторні виклики для незмінного введення не відбуваються завдяки debounce: якщо користувач зупинив набір, а потім видалив і знову ввів той самий текст, таймер debounce спрацьовує лише один раз. Скасування попереднього запиту через AbortController гарантує, що у випадаючому списку відображаються підказки саме для поточного рядка, а не для проміжного стану введення.

2.3.5. Алгоритм оновлення JWT-токенів

Авторизація в TechStore побудована на парі JWT-токенів: access-токен зі строком життя 15 хвилин і refresh-токен зі строком 7 днів. Access-токен передається в заголовку Authorization кожного авторизованого запиту. Коли він протерміновується, бекенд повертає 401, і клієнтська частина має непомітно для користувача оновити токен і повторити запит.

Алгоритм реалізований у вигляді response-interceptor Axios:

1) При отриманні відповіді з кодом 401 перевіряється поле _retry на оригінальній конфігурації запиту.

2) Якщо _retry=true (тобто це вже повторна спроба) — виконується logout користувача, тобто очищення токенів; саме перенаправлення на /login виконує middleware при наступній спробі відкрити захищений маршрут.

3) Якщо _retry відсутній — встановлюється _retry=true; виконується запит на POST /api/auth/refresh з refreshToken.

4) Якщо refresh успішний — оновлюється accessToken у authStore; оригінальний запит повторюється з новим токеном; результат повертається з функції-перехоплювача.

5) Якщо refresh неуспішний — виконується logout (очищення токенів у localStorage).

Додатково обробляється ситуація, коли кілька запитів майже одночасно отримують відповідь 401. Щоб уникнути дублювання refresh-запитів, використовується спільний promise refreshPromise. Перший запит запускає оновлення токена, а інші очікують завершення цієї операції. Після успішного оновлення токена всі відкладені запити повторюються з актуальним access-токеном.

2.3.6. Алгоритм валідації форм

Для більшості форм авторизації та адміністративних форм у TechStore використано React Hook Form + Zod. Сторінка checkout реалізована окремо через локальний стан React, оскільки має складну умовну логіку доставки, вибору міста/відділення Нової пошти та способу оплати.

1) У файлі поряд з компонентом форми оголошується схема Zod з усіма полями і їхніми обмеженнями.

2) З цієї схеми автоматично виводиться TypeScript-тип через z.infer<typeof schema>.

3) У компоненті форми викликається useForm з resolver: zodResolver(schema).

4) Кожне поле прив'язується через {...register("fieldName")}.

5) При submit викликається handleSubmit, який спочатку валідує дані за схемою.

6) Якщо є помилки — у formState.errors з'являються відповідні записи; submit-обробник не викликається.

7) Якщо помилок немає — викликається переданий submit-обробник з типобезпечно валідованими даними.

Такий підхід зменшує ризик розбіжностей між типами даних, правилами валідації та повідомленнями про помилки. Якщо змінюється схема форми, TypeScript допомагає виявити компоненти, які також потребують оновлення.

2.3.7. Алгоритм lazy-loading зображень і оптимізації next/image

Зображення товарів є одним із ключових чинників, що впливають на вагу сторінок інтернет-магазину. На сторінці каталогу може відображатися кілька десятків карток товарів із мініатюрами, а на головній сторінці — банери та добірки товарів. Без оптимізації завантаження зображень сторінка може мати надмірний обсяг і повільно відкриватися на мобільних пристроях або за умов нестабільного з’єднання.

Для розв'язання цієї задачі у TechStore використовується компонент next/image, що автоматизує кілька оптимізацій одночасно. Алгоритм його роботи:

1) Серверна частина Next.js при першому запиті зображення генерує його варіанти у потрібних розмірах через бібліотеку Sharp.

2) Next.js кешує оптимізовані варіанти зображень і повторно використовує їх для наступних запитів, не виконуючи зайву обробку.

3) У згенерованій HTML-розмітці зображення оформлено з атрибутами srcset (різні варіанти для різних щільностей екрана) і sizes (підказка браузеру, який варіант обрати залежно від ширини контейнера).

4) За замовчуванням всі зображення, що знаходяться за межами першого екрана (below the fold), вантажаться через loading="lazy" — браузер вантажить їх лише при наближенні до видимості.

5) Для зображень першого екрана (наприклад, hero-банер головної сторінки) свідомо встановлюється priority={true}, щоб браузер почав вантажити їх раніше за інший контент.

6) Поки зображення вантажиться, на його місці показується placeholder — або blurDataURL (розмитий мініатюрний варіант, що генерується при build-time), або сірий контейнер з фіксованим розміром, щоб уникнути зміщення макета (CLS).

Сукупний ефект — типова сторінка каталогу важить 800–1200 КБ замість 5–10 МБ; LCP покращується у 3–5 разів; CLS близький до нуля. Для досягнення цього вимагається лише замінити <img> на <Image> з Next.js і вказати реалістичні значення width/height. Решта — автоматично.

2.3.8. Алгоритм оновлення списку бажань і порівняння

Список бажань (wishlist) і список порівняння оновлюються за схемою «запит — підтвердження — оновлення стану»: інтерфейс змінюється вже після того, як сервер підтвердив операцію. Такий підхід дещо повільніший за оптимістичне оновлення, проте усуває ризик розсинхронізації між клієнтом і базою даних у випадку, коли запит завершується помилкою.

Для авторизованого користувача список бажань зберігається на сервері, тому кожна зміна супроводжується мережевим запитом. Розглянемо алгоритм на прикладі додавання товару у список бажань.

1) Користувач натискає на іконку серця поряд з товаром.
2) Компонент викликає метод стору — addToWishlist(productId) для додавання або removeFromWishlist(productId) для видалення.
3) Метод відправляє відповідний запит на бекенд: POST /api/wishlist/:productId для додавання або DELETE /api/wishlist/:productId для видалення.
4) Після успішної відповіді сервера стор оновлює список бажань отриманими даними, і лише тоді іконка серця змінює стан (порожнє → заповнене).
5) Якщо запит завершується помилкою (наприклад, 401 — сесія протерміновано після refresh) — стан списку бажань не змінюється, а користувачу показується toast «Не вдалося оновити список бажань». Оскільки інтерфейс ще не відобразив зміну, відкочувати нічого не потрібно.

Перевага такого підходу — узгодженість стану: інтерфейс завжди показує саме той результат, який реально збережено на сервері, тому розсинхронізація між різними пристроями виключена. Платою за це є невелика затримка між дією користувача і візуальним відгуком, оскільки зміна чекає на відповідь сервера. Як напрям подальшого розвитку можливе впровадження оптимістичного оновлення (optimistic UI) для некритичних дій — додавання у список бажань чи порівняння, — коли зміна

відображалася б миттєво, а у разі помилки відкочувалася назад; для критичних операцій, зокрема створення замовлення, такий підхід застосовувати недоцільно.

Висновки до розділу 2

У другому розділі виконано проєктування інформаційної моделі системи TechStore. Описано дані, з якими взаємодіє клієнтська частина, проаналізовано зв’язки між основними сутностями та подано алгоритмічне забезпечення ключових сценаріїв.

Основні сутності — Product, Category, User, Cart, Order, OrderItem, Review, Coupon — описані з точки зору їх структури і зв'язків, що відображено на ER-діаграмі. Повністю на клієнті зберігається лише список нещодавно переглянутих товарів (RecentlyViewedItem). Список бажань зберігається на сервері в профілі користувача, а порівняння реалізоване гібридно — локально для неавторизованого відвідувача і на сервері для авторизованого. Потоки даних між браузером і сервером класифіковано: запити до API, введення від користувача, записи в localStorage.

Описані алгоритми визначають стабільність ключових сценаріїв клієнтської частини. До таких сценаріїв належать обробка асинхронних запитів у каталозі, уникнення race condition під час пошуку, централізоване оновлення JWT-токенів через спільний refreshPromise та підтримка коректного списку нещодавно переглянутих товарів. Попереднє опрацювання цих механізмів зменшує кількість помилок під час роботи застосунку та спрощує супровід коду.

Описані моделі та алгоритми реалізовані у вихідному коді, що детально розглядається у розділі 3.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Вибір технологій для фронтенд-частини TechStore здійснено з урахуванням характеру задачі: інтернет-магазин потребує якісної індексації сторінок, швидкого завантаження, адаптивної верстки, стабільної роботи форм і зручного керування клієнтським станом. Тому стек оцінювався не лише за популярністю, а й за практичною придатністю до реалізації каталогу, кошика, оформлення замовлення, особистого кабінету та адміністративної панелі.

Основними критеріями вибору були підтримка компонентної архітектури, типобезпечність, зрілість екосистеми, зручність реалізації адаптивного інтерфейсу, наявність інструментів оптимізації, підтримка темної теми, інтеграція з REST API та можливість подальшого розвитку проєкту.

Таблиця 3.1 — Основні технології фронтенд-частини TechStore

Технологія	Версія	Призначення
Node.js	≥18.x	Середовище виконання JavaScript, npm, build-процес
React	18.2.0	Побудова компонентного інтерфейсу
TypeScript	5.3.3	Статична типізація та підвищення надійності коду
Next.js	14.1.0	Маршрутизація, SSR/SSG, оптимізація зображень
Tailwind CSS	3.4.1	Адаптивна стилізація інтерфейсу
Next-Themes	0.2.1	Перемикання світлої та темної теми
Zustand	4.5.0	Глобальний стан кошика, фільтрів і налаштувань
React Hook Form	7.49.3	Робота з формами
Zod	3.22.4	Валідація схем введення
Axios	1.6.5	HTTP-запити до серверної частини
Firebase	12.9.0	Аутентифікація та допоміжні хмарні сервіси
Jest	30.3.0	Unit- та integration-тестування
Playwright	1.58.2	End-to-end тестування
CSS Modules	вбудовано	Локальні стилі компонентів через *.module.css
Express	4.x	REST API для демонстрації роботи клієнтської частини
MongoDB	актуальна	Документна БД для товарів, користувачів і замовлень
Mongoose	актуальна	ODM-бібліотека для роботи з MongoDB
JWT	-	Access/refresh-токени для авторизації
Helmet	-	Налаштування HTTP security headers на бекенді
express-validator	-	Валідація даних у серверних API-маршрутах
multer	-	Завантаження зображень товарів у каталог /uploads

Технологічний стек фронтенд-частини сформовано з урахуванням вимог до продуктивності, типобезпечності, адаптивності та подальшого розвитку системи. Базовим фреймворком обрано Next.js 14, оскільки він підтримує файлову маршрутизацію, App Router, metadata API, оптимізацію зображень і зручну підготовку до розгортання на Vercel. React 18 використовується як основна бібліотека

для побудови компонентів інтерфейсу, а TypeScript забезпечує контроль типів на етапі розробки.

3.1.1. Next.js — базовий фреймворк

Вибір базового фреймворку є важливим архітектурним рішенням, оскільки його зміна на пізніх етапах розробки потребує значних витрат часу. У межах React-екосистеми розглянуто Create React App, Vite, Remix, Gatsby та Next.js. Для порівняння також враховано альтернативні підходи на основі Nuxt, SvelteKit та Angular.

Create React App (CRA) раніше був поширеним інструментом для створення React-застосунків, однак у сучасній розробці поступово втратив актуальність. Відсутність вбудованого SSR і обмежені можливості SEO-оптимізації роблять його менш придатним для інтернет-магазину, де важливими є індексація сторінок каталогу та карток товарів.

Vite. Сучасний і швидкий інструмент для розробки React-застосунків, однак у межах цього проєкту перевагу надано Next.js через готову файлову маршрутизацію, metadata API, next/image, зручну структуру App Router і просте розгортання на Vercel.

Remix. Фреймворк активно розвивається після придбання компанією Shopify. Його основні принципи — повний SSR, форми як базовий механізм взаємодії, прогресивне покращення — є концептуально вдалими. Однак на момент реалізації роботи Remix мав меншу за Next.js екосистему, меншу кількість готових e-commerce-прикладів і слабшу інтеграцію з Vercel, тому для поточного проєкту він є передчасним вибором.

Gatsby. Зрілий фреймворк із розвиненою екосистемою, орієнтований на статичну генерацію сайтів. Для магазину з великим каталогом і цінами, що часто змінюються, така модель не підходить, оскільки кожне оновлення даних потребуватиме повної перебудови проєкту.

Next.js 14 обрано за сукупністю можливостей, важливих для фронтенд-частини TechStore: файлову маршрутизацію через App Router, metadata API, компонент next/image для оптимізації зображень, зручну структуру layout-ів і просте розгортання

на Vercel. У цьому проєкті сторінки каталогу та картки товару реалізовано як клієнтські компоненти, оскільки вони активно використовують стан, ефекти, фільтри, wishlist, comparison і API-запити з браузера.

3.1.2. React — бібліотека UI

Вибір React у межах цього проєкту зумовлений використанням Next.js, який побудовано навколо React-компонентної моделі. Додатковими аргументами є розвинена екосистема, наявність готових бібліотек для форм, стану, анімацій, тестування та широка підтримка e-commerce-патернів.

Vue має якісну документацію та популярний серед багатьох розробників, однак у контексті цього проєкту його використання вимагало б переходу на інший стек і заміни значної частини бібліотек. Angular є потужним фреймворком для великих корпоративних систем, але для фронтенд-частини цього проєкту він створює надмірну складність. Svelte, Solid і Qwik є перспективними технологіями, проте мають меншу кількість готових рішень для типових сценаріїв електронної комерції.

React 18 обрано: повна сумісність з Next.js, найбільша спільнота, найбільш затребуваний на ринку праці, компонентний підхід добре лягає на реальність магазину з десятками повторюваних елементів.

3.1.3. TypeScript

JavaScript — динамічно типізована мова. У невеликих проєктах динамічна типізація є зручною, однак у проєктах із десятками типів сутностей (Product, Order, User, Cart) одна помилка в назві поля може спричинити дефект, що виявиться лише у браузері кінцевого користувача.

TypeScript вирішує це на рівні редактора і збірки. Типи Product, Order, User описуються один раз, після чого при будь-якому зверненні до поля редактор підказує доступні варіанти і миттєво підкреслює помилку. Рефакторинг стає безпечнішим: змінив форму інтерфейсу — TypeScript покаже всі місця, де треба адаптувати.

У TechStore налаштований strict-режим і path-alias @/* — короткі читабельні імпорти замість відносних шляхів через три рівні вгору. Альтернативи TypeScript

(Flow, ReScript) або покинуті, або занадто нішеві. «Чистий JS» — свідома відмова від страхівки, яку немає сенсу відкидати.

3.1.4. Стилізація: Tailwind CSS, CSS Modules, next-themes

Для стилізації інтерфейсу TechStore використано Tailwind CSS у поєднанні з CSS Modules. Такий підхід забезпечує повторне використання дизайн-токенів, контроль локальної області дії стилів, підтримку адаптивної верстки та сумісність із клієнтськими інтерактивними сторінками.

Tailwind CSS є utility-first фреймворком, у якому стилізація виконується через набір готових класів у JSX. Такий підхід зменшує потребу у створенні великої кількості окремих CSS-класів, підтримує єдину систему відступів, розмірів і кольорів, а також спрощує адаптивну верстку через префікси sm, md і lg. Під час продакшн-збірки до бандла потрапляють лише використані класи, що зменшує розмір фінальних CSS-файлів.

Tailwind CSS не покриває всіх потреб стилізації. Для складних анімацій та нестандартних сіткових композицій utility-класи стають надмірно громіздкими, тому в цих випадках доцільно використовувати CSS Modules: звичайний CSS у файлі *.module.css, де класи хешуються автоматично при збірці — без конфліктів між компонентами. Кожна сторінка має власний page.module.css, складні компоненти — свої *.module.css.

CSS-in-JS (styled-components, Emotion) міг би бути альтернативою, однак для цього проєкту обрано Tailwind CSS і CSS Modules, оскільки така комбінація простіша для підтримки, добре працює з темізацією через CSS-змінні та не ускладнює структуру компонентів.

Бібліотека next-themes використовується для реалізації світлої та темної теми. Вона розв’язує проблему короткочасного відображення неправильної теми під час першого рендерингу сторінки, оскільки застосовує відповідний клас ще до повної гідратації React. Також бібліотека підтримує системні налаштування користувача, синхронізацію між вкладками та зручні хуки для перемикання теми.

3.1.5. HTTP-клієнт: Axios

Fetch — вбудований браузерний API, здавалося б, достатній. Але в магазині, де всі запити потребують авторизаційного заголовка, де треба обробляти 401 і виконувати refresh-token-flow, де GET-запити мають автоматично повторюватись при тимчасовому збої — fetch потребує обгортки, яку треба написати самостійно. Axios дає цю обгортку готовою.

У межах реалізації API-клієнта використано єдиний формат отримання відповіді через response.data. Interceptors виконують роль проміжного шару до та після кожного запиту: JWT-токен підставляється автоматично, відповіді з кодом 401 обробляються централізовано, а тимчасові мережеві помилки можуть повторно оброблятися без дублювання логіки в бізнес-компонентах. Це зменшує кількість повторюваного коду та уніфікує обробку HTTP-помилок.

ky, react-query/TanStack Query — хороші альтернативи, особливо останній з кешуванням і дедуплікацією. Але вони мають свою філософію і вимагають окремого вивчення. Для цього проєкту простота Axios виявилась доречнішою.

3.1.6. Форми: React Hook Form + Zod

Форм у магазині багато: реєстрація, вхід, відновлення пароля, оформлення замовлення, редагування профілю, додавання товару в адмінці, форма відгуку. І всі вони мають валідуватись, показувати помилки, блокуватись під час відправки.

Formik раніше був стандартом, але зараз поступається React Hook Form за продуктивністю. RHF будується навколо неконтрольованих компонентів і ref-ів — мінімум перерендерів при введенні. У формі з 15 полями це відчутно. API чистий: useForm, register, handleSubmit. Інтеграція зі сторонніми валідаторами — через @hookform/resolvers.

Zod — типобезпечна схема-валідатор. Описав схему один раз: z.object({ email: z.string().email(), password: z.string().min(8) }) — і отримав одночасно TypeScript-тип і набір правил перевірки. Немає ситуації, коли тип і валідація розходяться: джерело одне. Повідомлення про помилки пишуться одразу українською: «Некоректний

email», «Мінімум 8 символів». Yup раніше вирішував ту ж задачу, але Zod виграє в TypeScript-інтеграції.

3.1.7. Авторизація: Firebase Authentication

Firebase Authentication використовується для Google-аутентифікації на стороні фронтенду. Після успішного входу Firebase ID token передається на бекенд, після чого бекенд видає власні JWT-токени TechStore для подальшої роботи із захищеними API-маршрутами.

Надалі фронтенд працює не з токенами Firebase, а з власними access і refresh JWT-токенами TechStore. Access-токен додається до API-запитів у заголовку Authorization, а також записується у cookie auth-token для роботи middleware із приватними маршрутами.

Конфіденційні параметри конфігурації Firebase (API ключ, auth domain тощо) мають NEXT_PUBLIC_-префікс, оскільки за дизайном вони публічні — це ідентифікатори клієнта для Firebase Auth, а не секрети. Реальні секрети залишаються виключно на бекенді у Firebase Admin SDK. Auth0 і Clerk — хороші альтернативи, але платні на масштабах; NextAuth.js — безкоштовний, але вимагає більше налаштувань. Firebase тут — найшвидший старт і нульова вартість.

3.1.8. Управління станом: Zustand

Стан інтернет-магазину — це кошик, авторизаційна сесія, список бажань, нещодавно переглянуті товари, вибір мови, тема, активні фільтри. Все це треба десь тримати і синхронізувати між компонентами.

Redux є зрілим і добре документованим рішенням для керування станом, однак навіть із Redux Toolkit потребує більшої кількості службового коду, ніж потрібно для цього проєкту. Для TechStore доцільнішим є легший підхід, оскільки основні стани системи логічно поділяються на окремі доменні стори: кошик, авторизація, список бажань, порівняння, тема та сповіщення.

MobX забезпечує реактивну модель стану та зменшує кількість шаблонного коду, але його механізм автоматичного відстеження залежностей може бути менш

прозорим під час налагодження. У навчальному проєкті важливо, щоб логіка зміни стану легко простежувалася у коді.

Recoil офіційно не вийшов зі статусу Release Candidate, його розвиток уповільнився, що робить його непридатним для виробничого проєкту.

Jotai. Атомарний підхід, хороша TypeScript-підтримка, стабільний. Концепція «атомів, що комбінуються» трохи нестандартна для предметної області типу кошик/список бажань, де природніше думати про цілісний об'єкт-стор.

Context API + useReducer. Вбудовано в React, без залежностей. Але будь-яка зміна контексту перерендерить всіх споживачів — у великому застосунку це проблема продуктивності.

Zustand є компактною бібліотекою для керування станом у React-застосунках. Стор створюється через функцію create, у якій описуються поля стану та методи їх зміни. Компонент підписується лише на потрібний фрагмент стану, тому перерендер відбувається не при кожній зміні стору, а лише при зміні вибраного значення. Middleware persist дозволяє зберігати частину стану в localStorage, що важливо для кошика, теми та списку бажань.

Отже, Zustand обрано через мінімальну кількість службового коду, зрозумілу модель оновлення стану, підтримку TypeScript, наявність persist-механізму та можливість розділити глобальний стан на незалежні предметні модулі.

Таблиця 3.2 — Порівняння засобів керування станом

Засіб	**Переваги**	**Недоліки**	**Оцінка для TechStore**
Redux Toolkit	Зрілість, DevTools, передбачуваність	Більше службового коду	Доцільний для більших команд
MobX	Реактивність, менше шаблонів	Менша прозорість залежностей	Можливий, але менш контрольований
Recoil	Атомарна модель стану	Повільніший розвиток екосистеми	Не обрано
Jotai	Легкість, TypeScript, атоми	Менш природний для кошика як цілісного стору	Придатний, але не оптимальний

Засіб	Переваги	Недоліки	Оцінка для TechStore
Context API	Вбудований у React	Ризик зайвих перерендерів	Підходить лише для простого стану
Zustand	Мінімум коду, persist, селектори, TypeScript	Потребує дисципліни у структурі сторів	Обрано для TechStore

3.1.9. Допоміжні бібліотеки

Framer Motion — анімації. Декларативний API, природна інтеграція з React. AnimatePresence для анімації зникнення елементів (наприклад, видалення з кошика). Альтернативи — react-spring (точніша фізика) і чисті CSS-анімації (для простих випадків). Framer Motion обрано через баланс між можливостями і простотою API.

Lucide React — SVG-іконки. Форк Feather Icons, більше тисячі іконок, однорідний стиль, повне tree-shaking. У бандл потраплять лише ті, що реально використовуються.

Sonner — toast-сповіщення. «Товар додано у кошик», «Замовлення створено», «Не вдалося завантажити список товарів». Мінімальний API, хороша анімація за замовчуванням.

Recharts — графіки в адмінці. Найпростіший API серед React-сумісних рішень. Chart.js — потужніший, але з'являється зайва складність для наших задач. D3 — взагалі занадто низькорівневий.

date-fns — дати. Модульний, tree-shakeable. Moment.js давно застарів і важить 70 КБ. date-fns дає тільки те, що імпортується.

clsx + tailwind-merge — утиліти для умовних класів. clsx дозволяє писати умовні класи об'єктом; tailwind-merge коректно вирішує конфлікти між Tailwind-класами (якщо є і px-2, і px-4 — перемагає останній). Разом економлять купу дрібних if-ів у кожному компоненті.

3.1.10. Тестування та якість коду

Три рівні тестування. Jest — юніт-тести для чистої логіки: функції форматування, методи сторів, валідаційні схеми Zod. Налаштовано з jsdom і alias @/* — тести пишуться з тими самими шляхами, що й основний код.

React Testing Library — компонентні тести. Знаходимо елементи через getByRole, getByLabelText, а не за CSS-класами — тести не ламаються при рефакторингу внутрішньої структури. Перевіряємо поведінку: чи з'являється помилка при невалідному email, чи викликається addItem при натисканні кнопки.

Playwright — e2e. Повний сценарій у реальному браузері: від пошуку товару до підтвердження замовлення, від реєстрації до перевірки профілю. Налаштований на Chromium з українською локаллю, скріншотами при невдачі.

ESLint з eslint-config-next — статичний аналіз коду. Ловить неправильне використання next/image, потенційні проблеми з хуками, порушення правил стилю.

3.2. Вимоги до технічного та програмного забезпечення

3.2.1. Вимоги до технічного забезпечення серверної інфраструктури

Хоча об'єктом цієї роботи є саме клієнтська частина TechStore, для коректного функціонування потрібна низка серверних компонентів. Фронтенд розгортається на платформі Vercel — вона надає виділені ресурси на основі Edge Network з близько ста точок присутності у світі. Базові вимоги Vercel виконуються автоматично, оскільки платформа спеціально оптимізована під Next.js. На безкоштовному тарифному плані ліміти включають 100 ГБ вихідного трафіку на місяць і обмеження на тривалість виконання серверних функцій до 10 секунд, чого для дипломного проєкту достатньо з запасом.

Бекенд (Node.js + Express + MongoDB/Mongoose) розгортається на платформі Railway. Мінімальна конфігурація для навчально-практичного проєкту: одне ядро CPU, 512 МБ оперативної пам’яті для застосунку Node.js і підключення до MongoDB / MongoDB Atlas. Така конфігурація є достатньою для демонстрації роботи фронтенду, API, авторизації, кошика, замовлень і адміністративних сторінок.

Завантаження зображень товарів реалізовано через multer на бекенді з розміщенням файлів у каталозі /uploads. У конфігурації Next.js також дозволені зовнішні джерела зображень, зокрема Cloudinary, Railway-бекенд, images.unsplash.com, randomuser.me та localhost, що дає змогу працювати як із локальними, так і з віддаленими ресурсами.

3.2.2. Вимоги до клієнтського устаткування

Кінцевий користувач взаємодіє з TechStore через веб-браузер на персональному комп'ютері або мобільному пристрої. Мінімальні вимоги:

- операційна система: Windows 10 або новіша, macOS 10.15+, Ubuntu 20.04+, iOS 14+, Android 9+;
- браузер: Chrome 110+, Firefox 110+, Safari 16+, Edge 110+ (тобто останні дві мажорні версії на момент написання роботи);
- оперативна пам'ять: 4 ГБ для стабільної роботи з браузером;
- роздільна здатність екрана: від 360×640 пікселів (мобільні пристрої) до 4K (десктопи);
- швидкість інтернет-з'єднання: від 1 Мбіт/с (припустима робота на 3G), 10 Мбіт/с і вище — для оптимального досвіду.

Сайт також тестувався на повільних з'єднаннях через інструменти DevTools (емуляція 3G), і базова функціональність зберігається навіть на 400 кбіт/с, хоча картинки вантажаться повільніше.

3.2.3. Вимоги до програмного забезпечення для розробки

Для розробки і локального запуску проєкту потрібне таке програмне забезпечення:

- Node.js версії 18 або новіше — середовище виконання JavaScript на сервері; Next.js 14 потребує саме цієї версії;
- npm 9+ або yarn 1.22+ — менеджер пакетів для встановлення залежностей;

- Git — система контролю версій, потрібна для роботи з GitHub-репозиторієм;
- VS Code (рекомендовано) або інший редактор з підтримкою TypeScript — для редагування коду; для VS Code корисно встановити розширення ESLint, Prettier, Tailwind CSS IntelliSense, MongoDB for VS Code (для роботи з базою даних);
- сучасний браузер на основі Chromium для відлагодження клієнтської частини, з відкритими DevTools.

Для повноцінної розробки потрібен доступ до облікових записів сервісів, які використовує застосунок: Vercel для деплою фронтенду, Railway для бекенду, Firebase для Google-аутентифікації та GitHub для контролю версій. Для зображень використовується завантаження через multer на бекенді, а конфігурація Next.js також дозволяє роботу із зовнішніми джерелами зображень.

3.3. Опис програмної реалізації

У підрозділі описано, як саме обрані технології застосовано для побудови клієнтської частини TechStore. Розглянуто структуру каталогів проєкту, конфігураційні файли, реалізацію основних компонентів і сторів, інтеграцію з API, налаштування розгортання.

3.3.1. Структура проєкту

Уся клієнтська частина TechStore розташована у каталозі frontend GitHub-репозиторію. Саме цей каталог Vercel розглядає як корінь проєкту під час збірки. У корені каталогу знаходяться конфігураційні файли (package.json, tsconfig.json, next.config.js, tailwind.config.js, postcss.config.js, jest.config.ts, playwright.config.ts, .eslintrc.json, .env.example, .gitignore) і папка src з основним кодом.

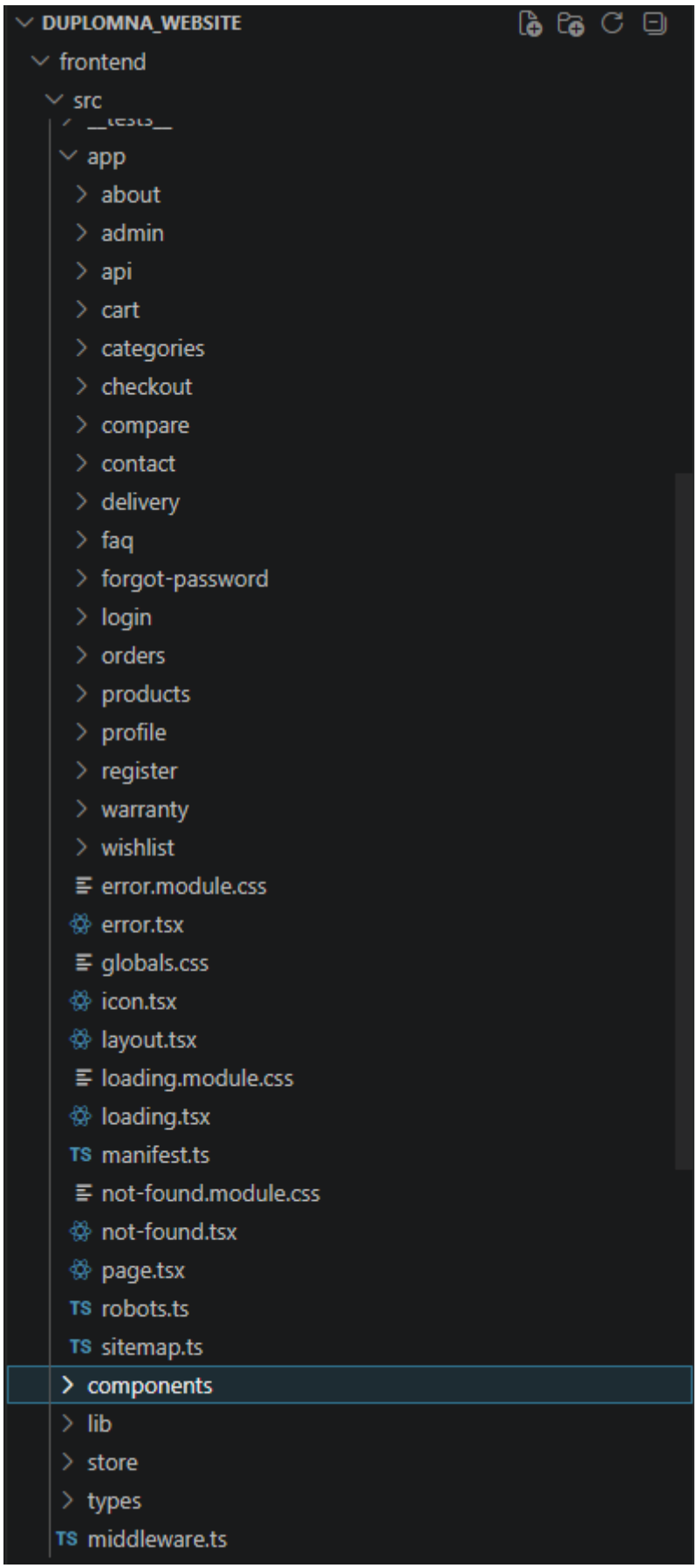


Рис. 3.1. Структура каталогу frontend

Усередині src проєкт розділений на кілька логічних частин: app — маршрутизація і сторінки; components — компоненти повторного використання; store — Zustand-стори; lib — допоміжні файли інфраструктури; types — TypeScript-інтерфейси; а також файл middleware.ts для захисту маршрутів. Глобальні стилі винесено у src/app/globals.css, а файли перекладів — у public/locales. Окремо у корені каталогу frontend знаходиться папка public для статичних ресурсів, доступних за URL.

Така організація структури проєкту забезпечує передбачуване розміщення файлів: сторінки зберігаються в app, повторно використовувані UI-компоненти — у components, бізнес-стан — у store, а інфраструктурний код для роботи з API та допоміжними сервісами — у lib.

3.3.2. Конфігураційні файли

Файл next.config.js — головна точка налаштувань Next.js. Серед важливих рішень: увімкнено strict-режим React (reactStrictMode: true), що під час розробки виявляє потенційно небезпечні патерни; вимкнено заголовок X-Powered-By (poweredByHeader: false), щоб не повідомляти зайвої інформації про використовувані технології; налаштовано images.remotePatterns для коректної роботи next/image із зображеннями з нашого бекенду і Cloudinary.

Найбільш насичений блок next.config.js — security headers. Він повертає масив правил із заголовками X-Frame-Options: SAMEORIGIN, X-Content-Type-Options: nosniff, Referrer-Policy: origin-when-cross-origin, Permissions-Policy: camera=(), microphone=(), geolocation=(), X-XSS-Protection: 1; mode=block і Cross-Origin-Opener-Policy: same-origin-allow-popups. Також у конфігурації вимкнено X-Powered-By і дозволено remotePatterns для зовнішніх джерел зображень.

```
const nextConfig = {
  async headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'X-DNS-Prefetch-Control',
            value: 'on'
          },
          {
            key: 'X-Frame-Options',
            value: 'SAMEORIGIN'
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff'
          },
          {
            key: 'Referrer-Policy',
            value: 'origin-when-cross-origin'
          },
          {
            key: 'Permissions-Policy',
            value: 'camera=(), microphone=(), geolocation=()'
          },
          {
            key: 'X-XSS-Protection',
            value: '1; mode=block'
          },
          {
            key: 'Cross-Origin-Opener-Policy',
            value: 'same-origin-allow-popups'
          },
        ],
      },
      {
        source: '/sitemap.xml',
        headers: [
          {
            key: 'Cache-Control',
            value: 'public, max-age=3600, s-maxage=3600, stale-while-revalidate=86400'
          },
        ],
      },
      {
        source: '/robots.txt',
        headers: [
```

Рис. 3.2. Налаштування заголовків безпеки

Файл tsconfig.json відповідає за поведінку компілятора TypeScript. Налаштовано strict: true, що активує всі найсуворіші перевірки одночасно:

noImplicitAny, strictNullChecks, strictFunctionTypes, strictPropertyInitialization, noImplicitThis. Окремо налаштовано path-mapping: paths: { "@/*": ["./src/*"] }, завдяки чому імпорти з будь-якого місця виглядають коротко: @/lib/api замість ../../../lib/api. Додатково встановлено incremental: true для прискорення повторних збірок.

Файл tailwind.config.js містить розширену конфігурацію Tailwind CSS. У theme.extend.colors визначено власну палітру через CSS-змінні: --background, --foreground, --primary, --primary-foreground, --secondary, --muted, --accent, --border, --input, --ring, --card. Це дає можливість єдиним рядком перемкнути всю палітру у темну або світлу тему. У content вказано шляхи src/**/*.{ts,tsx}.

3.3.3. Реалізація маршрутизації

Маршрутизація у TechStore побудована на App Router, що з'явився у Next.js 13. Принцип простий: кожна папка всередині src/app — це маршрут, кожен файл page.tsx у такій папці — сторінка. Динамічні параметри записуються через квадратні дужки в назві папки: [id], [slug]. Файл layout.tsx у кожній папці — обгортка, що зберігається при переходах між дочірніми маршрутами без перерендерингу.

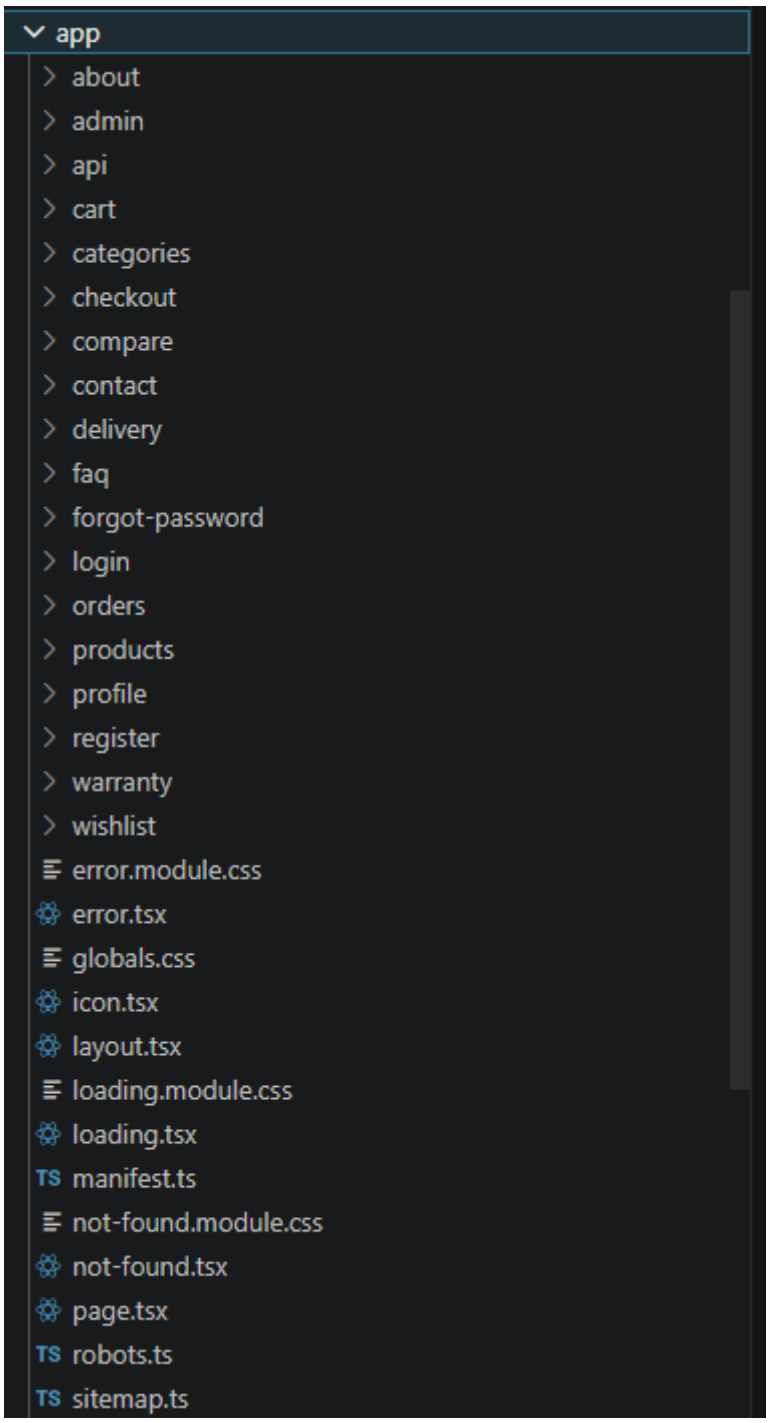


Рис. 3.3. Структура маршрутів у App Router

Структура маршрутів TechStore охоплює сторінки покупця та адміністратора: / — головна сторінка; /products — каталог; /products/[slug] — картка товару; /categories — категорії; /cart — кошик; /checkout — оформлення замовлення; /login, /register, /forgot-password — авторизаційні сторінки; /profile — профіль користувача; /orders і /orders/[id] — замовлення; /wishlist — список бажань; /compare — порівняння; /admin,

/admin/products, /admin/orders, /admin/users, /admin/categories, /admin/coupons, /admin/tickets — адміністративні сторінки.

У Next.js App Router сторінки можуть бути серверними або клієнтськими. У TechStore сторінки /products і /products/[slug] реалізовані як клієнтські компоненти з директивою 'use client', оскільки активно використовують useEffect, Zustand, інтерактивні фільтри, API-запити з браузера, wishlist, comparison і кошик.

3.3.4. Глобальний layout, метадані, SEO-файли

Файл src/app/layout.tsx — кореневий layout, що обгортає весь застосунок. Тут підключаються глобальні провайдери, налаштовується html-тег з lang="uk", додається інлайн-скрипт встановлення теми до основного рендеру. У body послідовно розташовані ThemeProvider, Header, тег main з основним вмістом, Footer і Toaster.

```tsx
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="uk" suppressHydrationWarning>
      <head>
        <script
          type="application/ld+json"
          dangerouslySetInnerHTML={{ __html: JSON.stringify(jsonLd) }}
        />
        <script
          dangerouslySetInnerHTML={{
            __html: `(function(){try{var t=localStorage.getItem('theme');if(t==='dark'||(t!=='light'&&matchMedia('(prefers-color-scheme:dark)').matches)){document.documentElement.classList.
            add('dark')}}catch(e){}})()`,
          }}
        />
        <script
          dangerouslySetInnerHTML={{
            __html: `(function(){var d=document,h=d.documentElement;h.setAttribute('data-fouc','loading');function reveal(){h.setAttribute('data-fouc','ready')}function check(){var l=d.
            querySelectorAll('link[rel="stylesheet"]'),p=[],i;for(i=0;i<l.length;i++){(function(x){if(x.sheet)return;p.push(new Promise(function(r){var done=function(){r()};x.
            addEventListener('load',done,{once:true});x.addEventListener('error',done,{once:true})})})(l[i])}if(!p.length){reveal();return}Promise.all(p).then(reveal)}if(d.
            readyState==='loading'){d.addEventListener('DOMContentLoaded',check)}else{check()}setTimeout(reveal,1500)})()`,
          }}
        />
      </head>
      <body className={inter.className} suppressHydrationWarning>
        <ErrorBoundary>
          <ThemeProvider
            attribute="class"
            defaultTheme="light"
            enableSystem
            disableTransitionOnChange
          >
            <Main>
              <App>
                <NavigationFOUCGuard>{children}</NavigationFOUCGuard>
              </App>
            </Main>
          </ThemeProvider>
        </ErrorBoundary>
      </body>
    </html>
  )
}
```

Рис. 3.4. Кореневий layout проєкту

Об'єкт metadata, експортований з layout.tsx, централізовано задає мета-теги: title.template ("%s | TechStore"), title.default, description, keywords, canonical URL, ім'я автора, інформацію про сайт. Окремо налаштовано блок openGraph (title, description,

type, locale, siteName, image-зображення) і блок twitter (для коректного відображення посилань у різних соціальних мережах).

У корені src/app розташовані спеціальні файли, що генерують технічні файли SEO і PWA: sitemap.ts, robots.ts і manifest.ts. Sitemap формує URL основних сторінок і товарів, robots.ts задає правила індексації, а manifest.ts містить базові параметри PWA: name, short_name, start_url, display: standalone, background_color, theme_color, orientation, categories і lang. Повноцінний офлайн-режим через service worker у поточній версії не реалізовано.

На рівні кореневого layout додано JSON-LD для Organization та WebSite, а також SearchAction для пошуку. На сторінках товарів реалізовано мікророзмітку schema.org Product у форматі JSON-LD з полями Brand, Offer і AggregateRating. Для подальшого SEO-розвитку залишається додати динамічні per-product metadata (title, description, Open Graph), оскільки в поточній реалізації заголовок сторінки товару задається на рівні клієнта.

3.3.5. Реалізація основних сторінок

Головна сторінка побудована з кількох логічно відокремлених секцій: hero-блок, категорії, добірка товарів, промо-блок і секція нещодавно переглянутих товарів. Для цього використано компоненти з каталогу src/components/home: HeroSection, Categories, FeaturedProducts, PromoSection і RecentlyViewed.

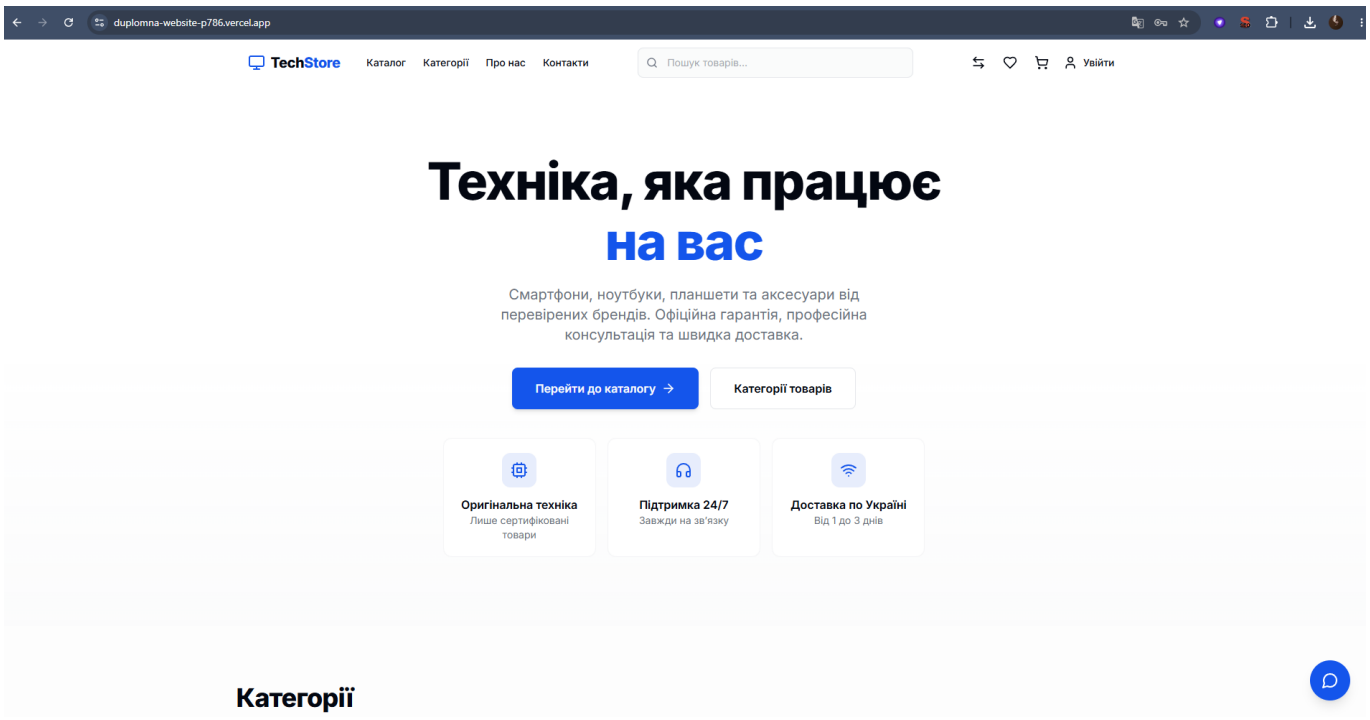


Рис. 3.5. Головна сторінка TechStore

Сторінка каталогу /products реалізована як клієнтська сторінка з URL-синхронізацією фільтрів, пагінацією, сортуванням і запитами до API. Вона читає параметри з URL, підтримує пошук, категорії, бренди, ціновий діапазон, наявність, сортування та пагінацію. На десктопі використовується бічна панель фільтрів, а на мобільних пристроях — висувна панель.

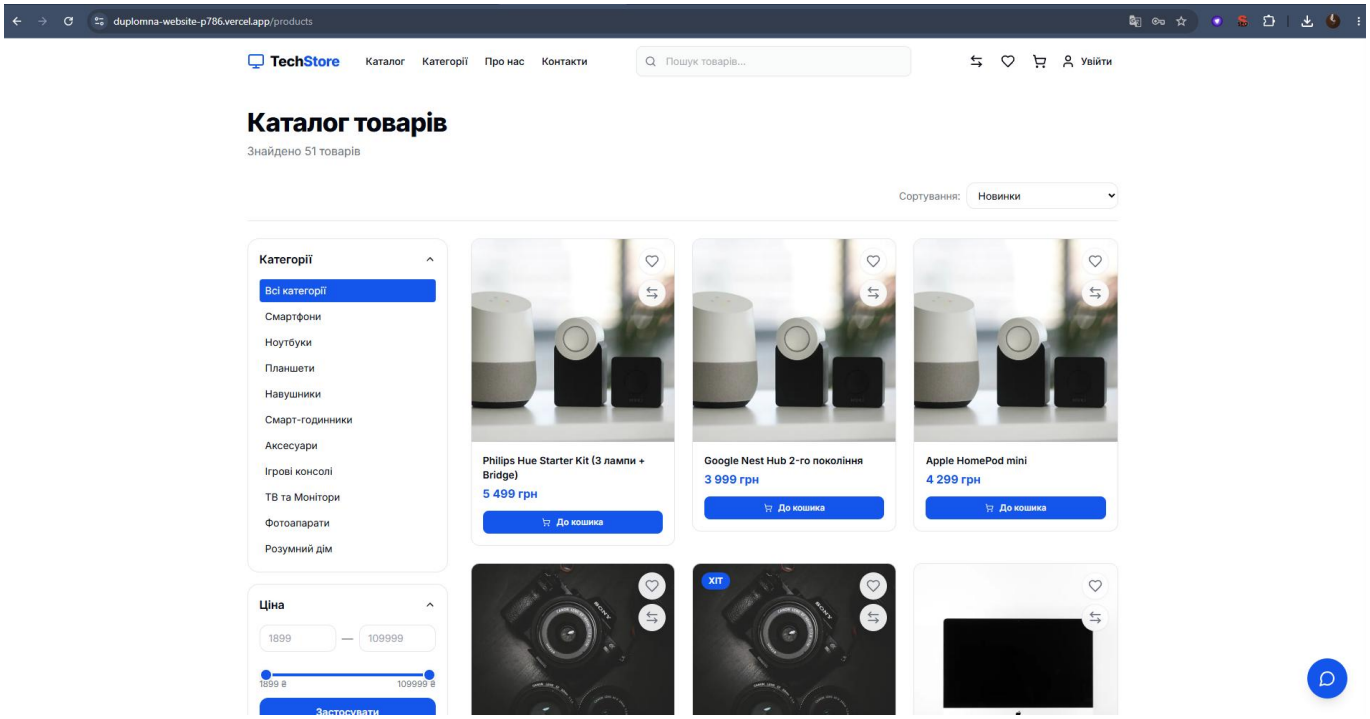


Рис. 3.6. Сторінка каталогу TechStore

Картка товару /products/[slug] реалізована як клієнтська сторінка, яка отримує товар за slug через API, кешує дані у sessionStorage, змінює document.title і додає товар у список нещодавно переглянутих. Сторінка показує галерею, мініатюри, назву, категорію, рейтинг, ціну, comparePrice, stock status, кнопки кошика/бажань/порівняння, блок доставки/гарантії, опис, features, specifications і відгуки.

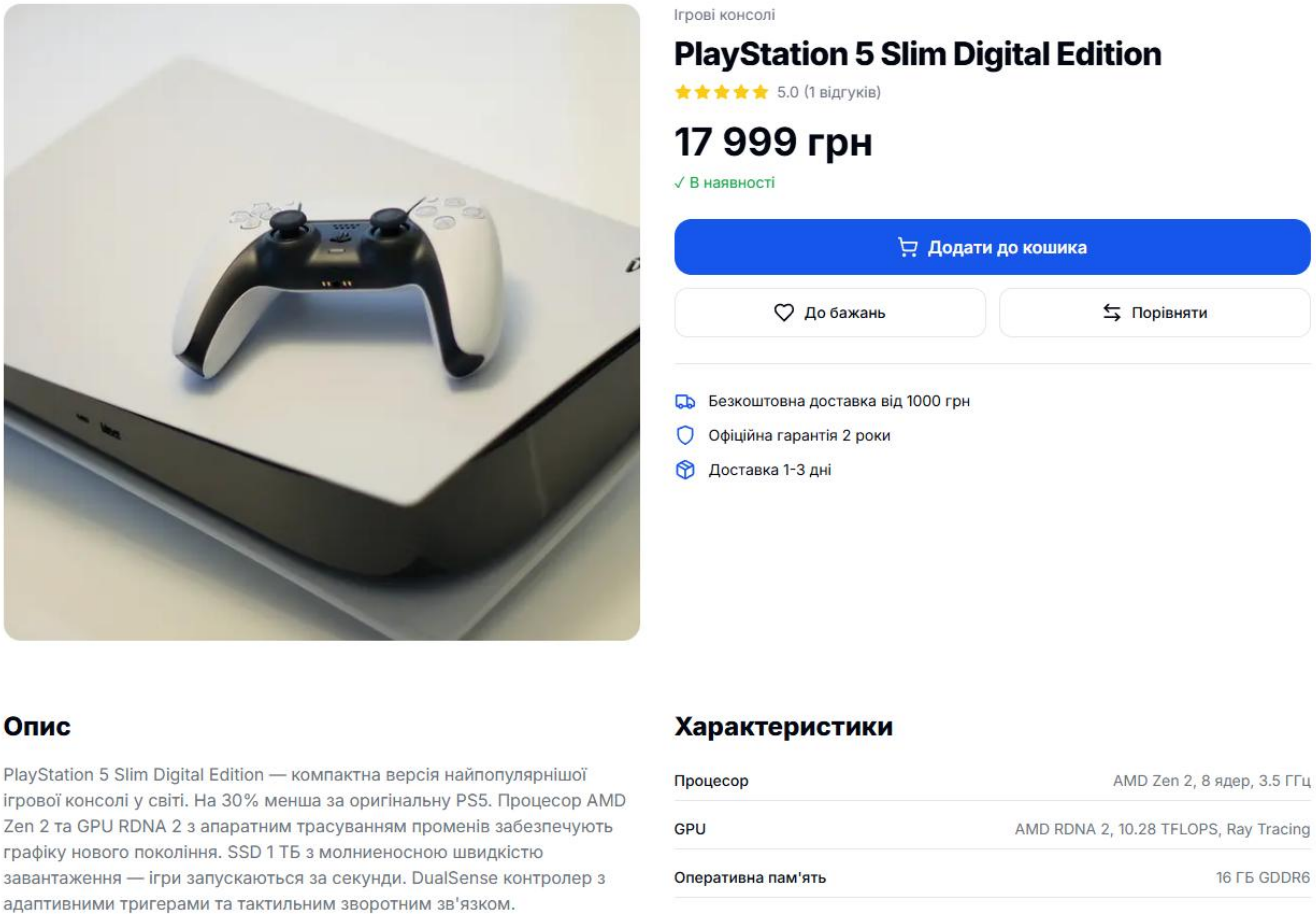


Рис. 3.7. Сторінка товару TechStore

Кошик реалізовано як повноцінна сторінка /cart, а в шапці сайту розміщено іконку кошика з лічильником доданих товарів. Натискання на іконку веде на сторінку /cart, де доступні зміна кількості, видалення позицій, поле для промокоду, блок розрахунку доставки та рекомендації супутніх товарів. Стан кошика тримається в єдиному cartStore, тому лічильник у шапці й вміст сторінки завжди узгоджені. Підсумковий блок на десктопній версії фіксується відносно області перегляду під час прокручування сторінки.

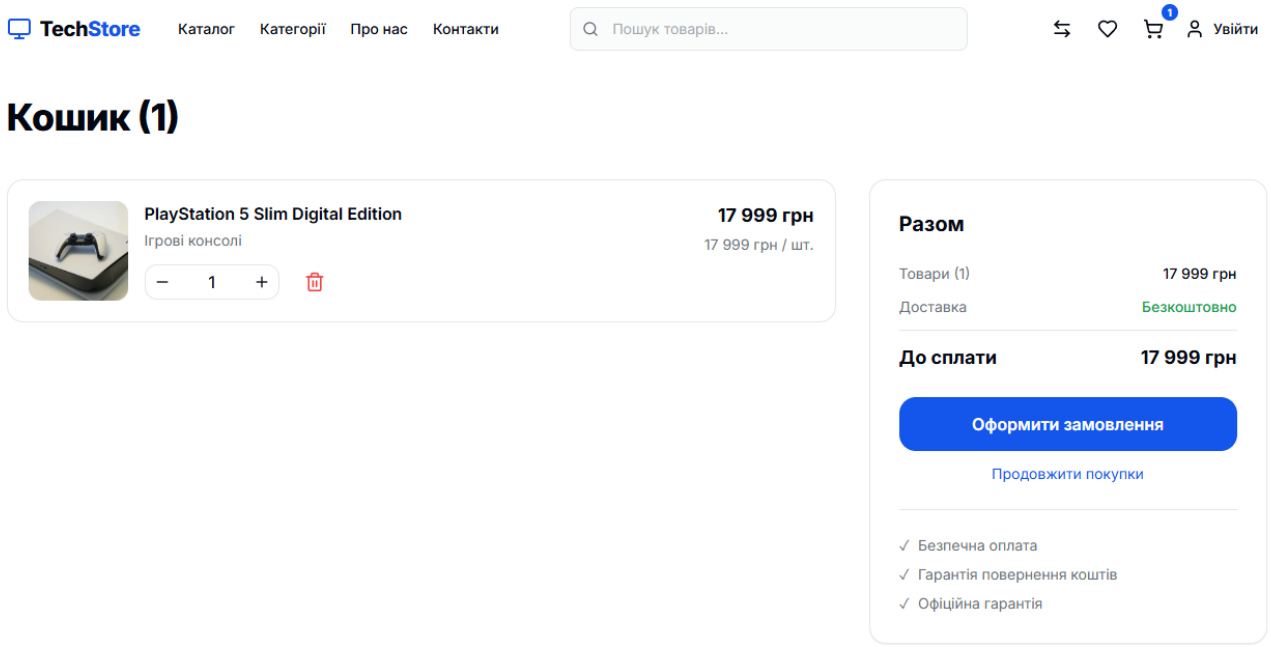


Рис. 3.8. Сторінка кошика

Сторінка оформлення замовлення (checkout) реалізована як односторінковий сценарій. Якщо користувач не авторизований, виконується перехід на /login?redirect=/checkout; якщо кошик порожній — на /cart. Форма checkout реалізована через React state з ручною перевіркою обов’язкових полів, формату телефону та вибору міста/відділення Нової пошти. Підтримуються доставка через Нову пошту або кур’єрська доставка, а також оплата при отриманні або вибір оплати картою.

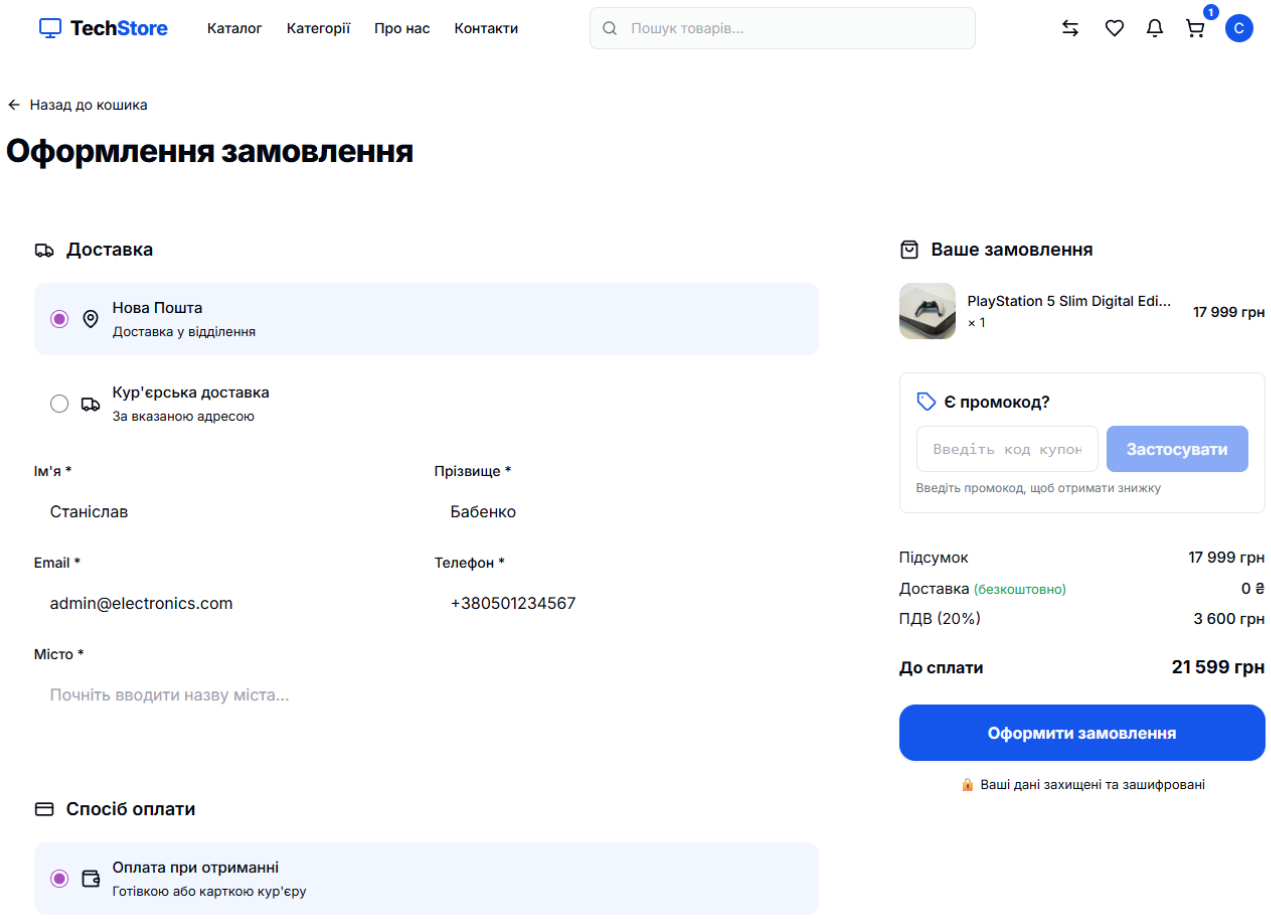


Рис. 3.9. Оформлення замовлення TechStore

Особистий кабінет реалізовано за маршрутом /profile. Сторінка доступна лише авторизованому користувачу та дозволяє редагувати ім’я, прізвище, телефон, переглядати email у недоступному для редагування полі, змінювати пароль і бачити підготовлений блок loyalty/bonus points. Замовлення винесені в окремі маршрути /orders і /orders/[id], а список бажань — у маршрут /wishlist.

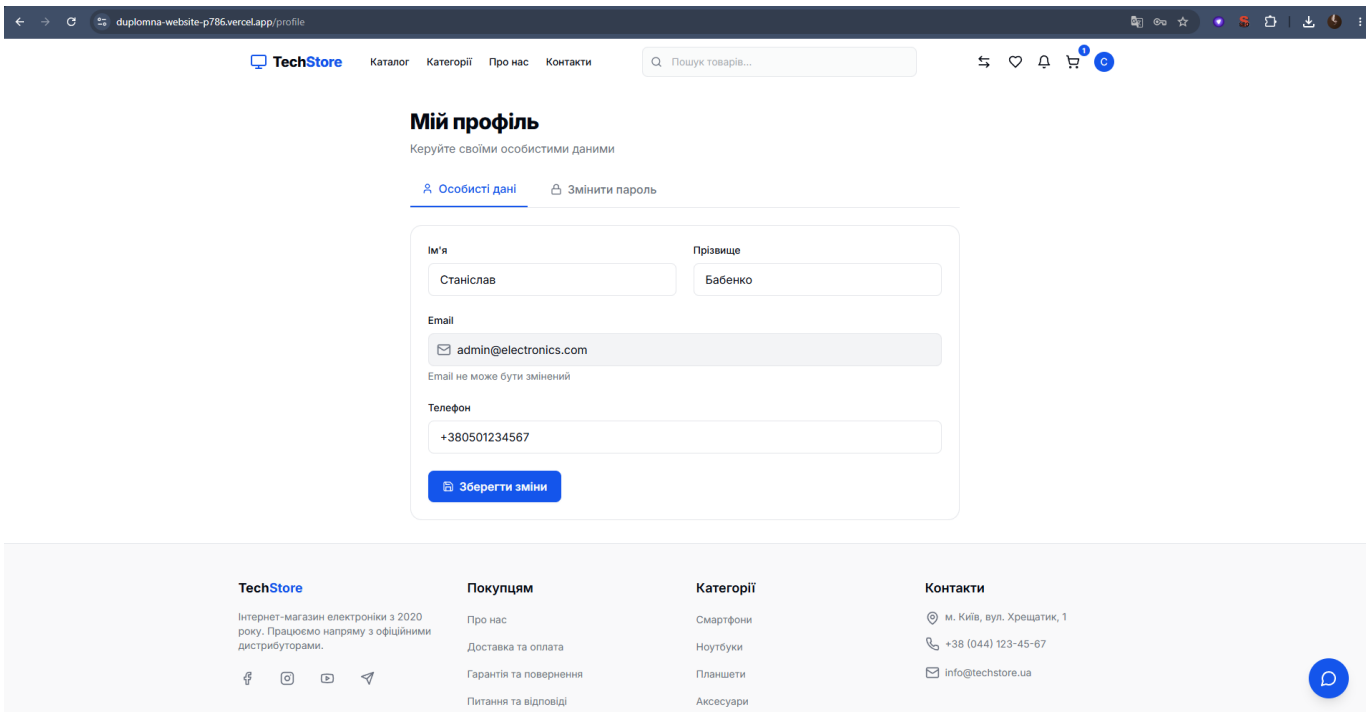


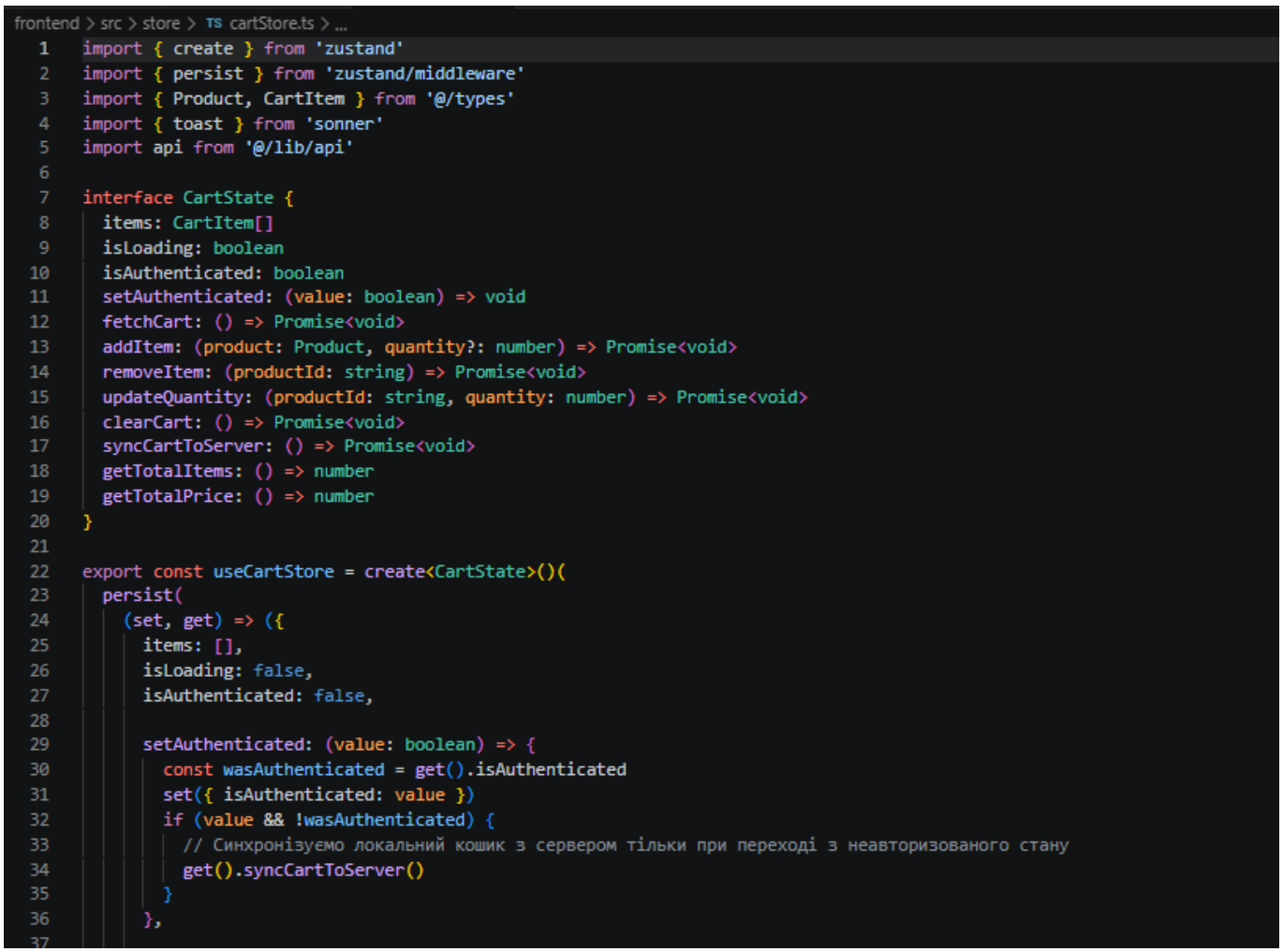
Рис. 3.10. Особистий кабінет користувача

Адміністративна панель є частиною інтерфейсу за маршрутом /admin. Вона містить dashboard, керування товарами, створення і редагування товарів, замовленнями, користувачами, категоріями, купонами та тікетами підтримки. Dashboard використовує Recharts для побудови графіків. Middleware перевіряє наявність auth-token для приватних маршрутів, а права адміністратора додатково перевіряються на рівні клієнтських компонентів і захищених API-маршрутів бекенду.

Рис. 3.11. Адміністративна панель TechStore

3.3.6. Реалізація сторів Zustand

Усі сім сторів розташовані в каталозі src/store і використовують єдиний шаблон. Кожен експортує функцію-хук, створену через create від Zustand. Усередині оголошується стан і методи його зміни через set і get. Для сторів з персистентністю додатково застосовується middleware persist з налаштуваннями name (ключ у localStorage) і partialize (вибір тільки тих полів, що мають зберігатися).



```
frontend > src > store > TS cartStore.ts > ...
import { create } from 'zustand'
import { persist } from 'zustand/middleware'
import { Product, CartItem } from '@/types'
import { toast } from 'sonner'
import api from '@/lib/api'

interface CartState {
  items: CartItem[]
  isLoading: boolean
  isAuthenticated: boolean
  setAuthenticated: (value: boolean) => void
  fetchCart: () => Promise<void>
  addItem: (product: Product, quantity?: number) => Promise<void>
  removeItem: (productId: string) => Promise<void>
  updateQuantity: (productId: string, quantity: number) => Promise<void>
  clearCart: () => Promise<void>
  syncCartToServer: () => Promise<void>
  getTotalItems: () => number
  getTotalPrice: () => number
}

export const useCartStore = create<CartState>()(
  persist(
    (set, get) => ({
      items: [],
      isLoading: false,
      isAuthenticated: false,

      setAuthenticated: (value: boolean) => {
        const wasAuthenticated = get().isAuthenticated
        set({ isAuthenticated: value })
        if (value && !wasAuthenticated) {
          // Синхронізуємо локальний кошик з сервером тільки при переході з неавторизованого стану
          get().syncCartToServer()
        }
      },
```

Рис. 3.12. Реалізація cartStore

Стор cartStore описує стан кошика з полями items, isLoading, isAuthenticated і методами setAuthenticated, fetchCart, addItem, removeItem, updateQuantity, clearCart, syncCartToServer, getTotalItems, getTotalPrice. Для гостя кошик зберігається локально через persist key cart-storage, а для авторизованого користувача синхронізується із серверним API.

Підписка на стор у компонентах виглядає просто: const items = useCartStore(state => state.items). Це селектор, що каже Zustand підписатися лише на конкретне поле. Якщо натомість написати const cart = useCartStore() — компонент перерендерюється на будь-яку зміну стору. Тому селектори — обов'язкова практика, що істотно впливає на продуктивність.

Аналогічно реалізовані інші стори: authStore (user, accessToken, refreshToken, login, register, googleLogin, logout, fetchProfile, syncStores); wishlistStore (items, fetchWishlist, addToWishlist, removeFromWishlist, clearWishlist, isInWishlist);

comparisonStore (comparisons, fetchComparisons, addToComparison, removeFromComparison, clearComparison, getComparisonByCategory); languageStore; notificationStore; recentlyViewedStore, який зберігає повні Product-об’єкти з лімітом 12 позицій.

3.3.7. API-клієнт і обробка помилок

Усі взаємодії з бекендом проходять через єдиний Axios-екземпляр, налаштований у src/lib/api.ts. Він має baseURL з NEXT_PUBLIC_API_URL із fallback на http://localhost:5000/api, JSON-заголовок і timeout. Токен авторизації читається з localStorage і додається до запитів у заголовок Authorization.

```
const API_BASE = ENV_API_URL || 'http://localhost:5000/api'

const api = axios.create({
  baseURL: API_BASE,
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: isProduction ? 30000 : 15000,
})

// Retry helper for GET requests that fail due to network/server startup issues
const MAX_RETRIES = 2
const RETRY_DELAY = 500 // ms

const sleep = (ms: number) => new Promise(resolve => setTimeout(resolve, ms))

const isRetryableError = (error: AxiosError) => {
  // Retry on network errors (backend not started yet) or 5xx server errors
  if (!error.response) return true // Network error / ECONNREFUSED
  const status = error.response.status
  return status >= 500 || status === 0
}

// Override the default get method to add retry logic for resilience
const originalGet = api.get.bind(api)
api.get = async function retryGet(...args: Parameters<typeof originalGet>) {
  let lastError: any
  for (let attempt = 0; attempt <= MAX_RETRIES; attempt++) {
    try {
      return await originalGet(...args)
    } catch (error: any) {
      lastError = error
      if (attempt < MAX_RETRIES && isRetryableError(error)) {
        await sleep(RETRY_DELAY * (attempt + 1))
      } else {
        throw error
      }
    }
  }
  throw lastError
} as typeof originalGet

api.interceptors.request.use(
  (config: InternalAxiosRequestConfig) => {
    if (typeof window !== 'undefined') {
      const token = localStorage.getItem('accessToken')
      if (token && config.headers) {
        config.headers.Authorization = `Bearer ${token}`
      }
    }
    return config
  },
  (error) => Promise.reject(error)
)
```

Рис. 3.13. API-клієнт з interceptors

Request interceptor додає Authorization: Bearer <token>. Response interceptor централізовано обробляє 401, запускає refresh-token-flow зі спільним refreshPromise для паралельних запитів і повторює оригінальний запит після оновлення токенів. Для GET-запитів реалізовано до двох повторних спроб при тимчасових мережевих збоях або 5xx-відповідях. Повідомлення користувачу здебільшого показуються на рівні сторів і компонентів через Sonner.

Усі звернення до API проходять через цей єдиний Axios-екземпляр безпосередньо: компоненти й стори викликають його методи там, де потрібні дані, — наприклад, api.get('/products') або api.post('/orders', data). Окремого доменного шару сервісів-обгорток у поточній версії немає, а типізація параметрів і відповідей забезпечується спільними TypeScript-інтерфейсами з каталогу types.

3.3.8. Реалізація форм

Форми у TechStore написані за єдиним патерном. Першим кроком оголошується схема Zod: const loginSchema = z.object({ email: z.string().email("Некоректний email"), password: z.string().min(8, "Мінімум 8 символів") }). Усі повідомлення про помилки одразу пишуться українською. Далі: type LoginInput = z.infer<typeof loginSchema> — це дає тип, який автоматично оновлюється при зміні схеми.

У компоненті: const { register, handleSubmit, formState: { errors, isSubmitting } } = useForm<LoginInput>({ resolver: zodResolver(loginSchema) }). Поле виглядає так: <input {...register("email")} />. Помилка: {errors.email && {errors.email.message}}. Submit-обробник: const onSubmit = handleSubmit(async (data) => { const res = await loginApi(data); ... }).

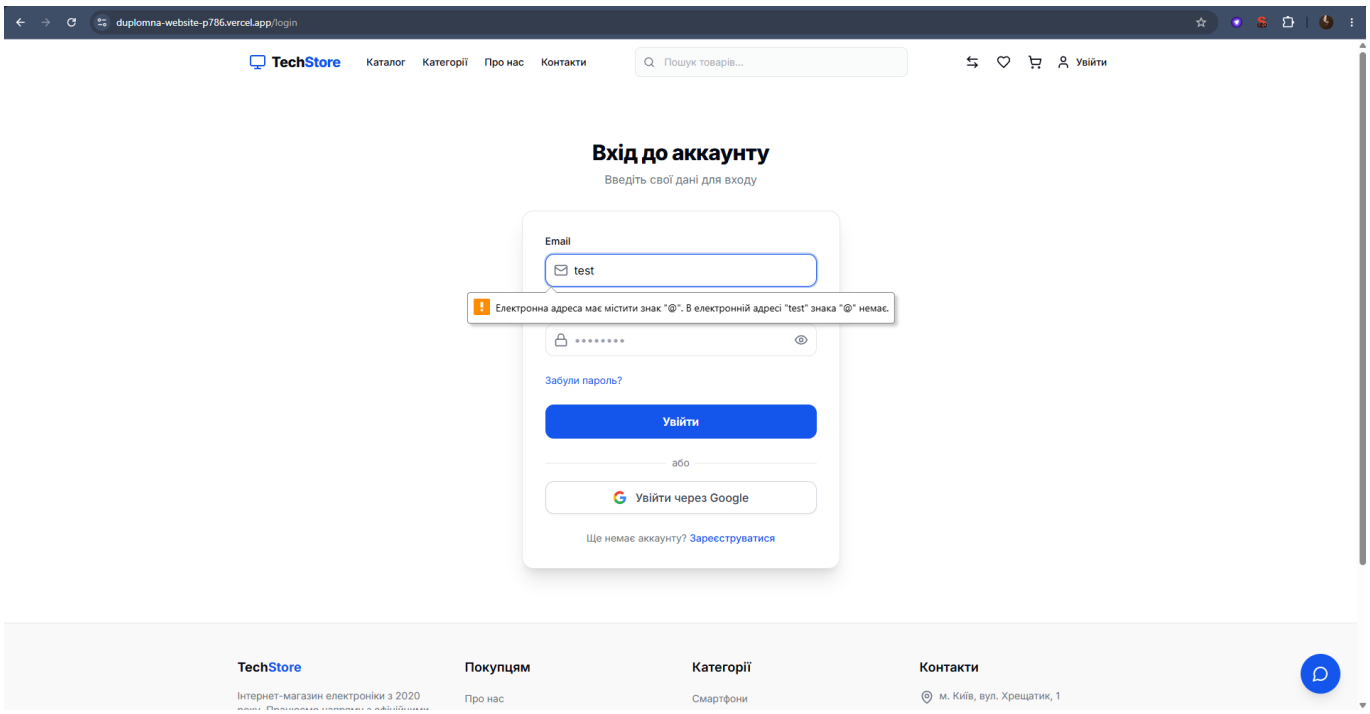


Рис. 3.14. Форма входу TechStore

Аналогічна структура використовується для форм входу, реєстрації, відновлення пароля та адміністративних форм створення або редагування товару. Checkout є винятком: він реалізований через React state з ручною перевіркою обов'язкових полів, формату телефону та вибору міста/відділення Нової пошти.

3.3.9. Тестування і розгортання

Юніт-тести покривають окремі функції з src/lib/utils.ts (форматування цін, переклад статусів, обчислення знижок), методи Zustand-сторів і валідаційні схеми Zod. Кожен тест-файл називається *.test.ts і знаходиться поряд з тестованим. Запуск виконується через npm run test, watch-режим — через npm run test:watch.

За результатами прогону тестів для клієнтської частини TechStore виконано 53 успішних Jest-тести, для серверної частини — 27 успішних тестів; загалом 80 юніт-тестів проходять без помилок. У сукупності з 44 Playwright e2e-сценаріями загальна кількість автоматизованих тестів становить 124. Покриття коду за рядками (lines) становить: фронтенд — 19,10 %, бекенд — 19,25 %, загальне — 19,18 %. Тести охоплюють ключову бізнес-логіку обох частин застосунку і забезпечують рівень покриття, достатній для бакалаврської кваліфікаційної роботи. Подальше нарощення покриття залишається пріоритетом при розвитку проєкту.

Розгортання організовано через Vercel. У панелі Vercel під'єднано GitHub-репозиторій, обрано папку frontend як корінь проєкту та прописано змінні середовища. Процес роботи побудовано так: після надсилання змін у гілку Vercel автоматично створює прев'ю-розгортання за окремим URL; цю адресу можна відкрити і протестувати функціональність до об'єднання змін з main. Після об'єднання Vercel розгортає основну версію застосунку.

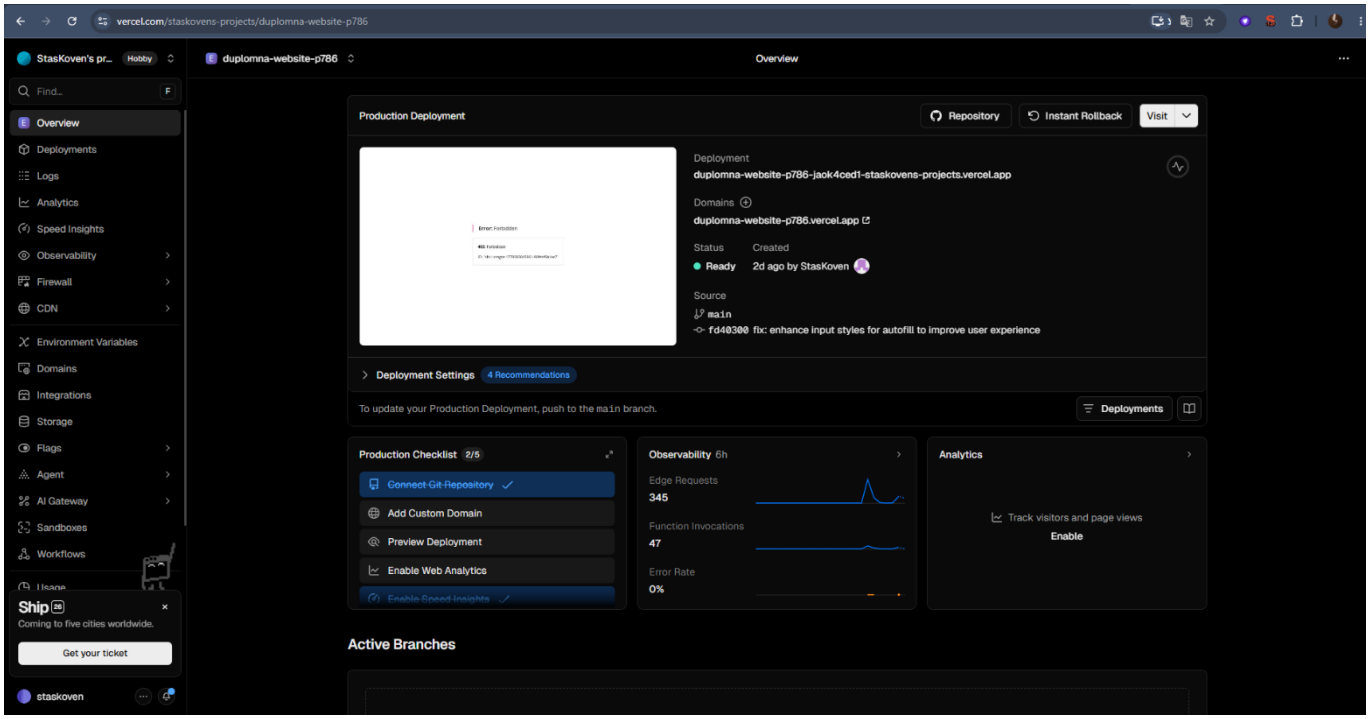


Рис. 3.15. TechStore у продакшні

Для проєкту налаштовано npm-скрипти для запуску тестів, збірки та lint-перевірки: npm run test, npm run build, npm run lint. Розгортання фронтенду виконується через Vercel, бекенду — через Railway.

3.3.10. Оптимізація продуктивності

Продуктивність — окрема тема, що варта спеціального розгляду, оскільки саме її значення безпосередньо корелюють з конверсією. У TechStore застосовано низку оптимізаційних технік на різних рівнях.

На рівні стратегії рендерингу важливо враховувати фактичну реалізацію: сторінки каталогу і картки товару є клієнтськими компонентами, оскільки активно

використовують стан, ефекти, взаємодію зі сторами та API-запити з браузера. Оптимізація досягається через code splitting, lazy-loading другорядних компонентів, кешування частини даних на клієнті та оптимізацію зображень через next/image.

На рівні бандла. Next.js автоматично розбиває JavaScript-код на менші частини (code splitting) — кожна сторінка отримує тільки той код, який їй реально потрібен. Code splitting працює і всередині застосунку: компоненти, обгорнуті у dynamic import з опцією ssr: false, виносяться в окремий бандл і не блокують серверний рендеринг сторінки. У TechStore це застосовано до ChatWidget і BottomNav, які не критичні для першого рендеру. Важка бібліотека Recharts, що використовується в адмінці, потрапляє в окремий chunk завдяки розбиттю коду на рівні маршрутів (route-level splitting).

На рівні зображень. Як описано в підрозділі 2.3.7, через next/image автоматично формуються адаптивні версії, налаштовано lazy-loading нижче першого екрана і priority для критичних зображень.

На рівні шрифтів. TechStore використовує шрифт Inter, підключений через next/font/google, що автоматично робить self-hosting — шрифти зберігаються у /_next/static і не вимагають зовнішніх запитів до Google Fonts. Це усуває не тільки повільну ланку, але й окремий ризик трекінгу.

На рівні мережевої взаємодії застосовується централізований API-клієнт із retry для GET-запитів, кешування окремих клієнтських даних у sessionStorage/localStorage та стандартне кешування статичних ресурсів Next.js. Динамічні API-відповіді залишаються під контролем бекенду, тому остаточні показники продуктивності мають фіксуватися після Lighthouse-аудиту продакшн-версії.

За результатами Lighthouse-аудиту продакшн-версії (https://duplomna-website-p786.vercel.app/) головна сторінка має такі показники. Mobile: Performance 97, Accessibility 96, Best Practices 96, SEO 100; FCP 1,1 с, LCP 2,6 с, TBT 40 мс, CLS 0,001. Desktop: Performance 100, Accessibility 96, Best Practices 96, SEO 100; FCP 0,3 с, LCP 0,6 с, TBT 0 мс, CLS 0,001. Десктопні Core Web Vitals (LCP 0,6 с, TBT 0 мс, CLS 0,001) відповідають категорії Good; мобільне LCP 2,6 с незначно перевищує поріг Good (2,5

с) — Lighthouse зафіксував попередження про дані IndexedDB, які могли вплинути на цю метрику.

3.3.11. Безпека на рівні фронтенду

Хоча основна лінія оборони знаходиться на бекенді, фронтенд також несе власну частку відповідальності за безпеку. У TechStore передбачено кілька рівнів захисту.

Security headers, налаштовані у next.config.js і описані у 3.3.2, — базова лінія захисту від типових атак. X-Frame-Options блокує вбудовування сайту в iframe (clickjacking), X-Content-Type-Options відключає MIME-sniffing, Referrer-Policy контролює, що потрапляє у заголовок Referer при переході. Самі по собі вони не замінюють серверних перевірок, але закривають значну частину поверхні атак.

Обережна робота з користувацьким вводом — ще один важливий шар. React за замовчуванням екранує всі JSX-змінні, тому небажаний HTML не виконується у браузері. Опис товару виводиться як звичайний текст у JSX, тому React автоматично екранує потенційно небезпечний HTML без додаткової ручної санітизації в поточній реалізації.

Усі чутливі параметри (API-ключі, секрети) — у файлі .env.local, що не комітиться у git. Файл .env.example містить лише імена змінних без значень. Змінні з префіксом NEXT_PUBLIC_ доступні з клієнтського коду; усі інші — лише з серверного. У NEXT_PUBLIC_ ніколи не повинно бути справжніх секретів — вони потраплять у бандл і стануть видимими будь-якому користувачу. Firebase-ключі мають NEXT_PUBLIC_-префікс, бо за дизайном вони і так публічні (ідентифікатори клієнта) — реальні секрети залишаються виключно у Firebase Admin SDK на бекенді.

Захист маршрутів через middleware працює до завантаження сторінки. src/middleware.ts перевіряє cookie auth-token і виконує redirect на /login для приватних маршрутів (/admin, /profile, /orders, /wishlist, /checkout), якщо токена немає. Middleware перевіряє факт авторизації, а права адміністратора додатково контролюються у фронтенд-компонентах і на рівні захищених API-маршрутів бекенду.

П'ятий рівень — обмеження частоти дій. Кнопки submit блокуються на час виконання запиту через прапорець isSubmitting у React Hook Form, що запобігає випадковим повторним натисканням і дублюванню замовлень. Дії з високою частотою виклику, зокрема голосування за відгук, обмежуються блокуванням відповідного елемента керування на час виконання запиту. Серверна частина також використовує rate-limit-механізми, а клієнтські обмеження зменшують кількість зайвих мережевих запитів.

3.3.12. Тестування продуктивності та доступності

Окрім юніт- і e2e-тестів, для контролю якості клієнтської частини TechStore використано інструменти, орієнтовані на продуктивність і доступність. Це Google Lighthouse — для аудиту швидкодії, доступності, найкращих практик і SEO; axe DevTools — для виявлення базових порушень accessibility-вимог; вбудовані інструменти Chrome DevTools — для аналізу мережевих запитів, рендеру і обсягу JavaScript-бандла.

Lighthouse-аудит виконано для продакшн-версії за адресою https://duplomna-website-p786.vercel.app/. На мобільному профілі отримано такі оцінки: Performance — 97, Accessibility — 96, Best Practices — 96, SEO — 100. На десктопному профілі: Performance — 100, Accessibility — 96, Best Practices — 96, SEO — 100. Ключові метрики Core Web Vitals для мобільного: FCP 1,1 с, LCP 2,6 с, TBT 40 мс, CLS 0,001; для десктопу: FCP 0,3 с, LCP 0,6 с, TBT 0 мс, CLS 0,001. Десктопні метрики повністю відповідають категорії Good; мобільне LCP 2,6 с незначно перевищує поріг Good (2,5 с) — під час аудиту Lighthouse зафіксував попередження про кешовані дані IndexedDB, які могли завищити цю метрику.

Окремо для перевірки користувацьких сценаріїв виконано прогін Playwright e2e-тестів: 44 із 44 сценаріїв успішно пройдені, тривалість прогону — приблизно 42 секунди. Сценарії охоплюють відкриття головної сторінки, навігацію між розділами, роботу каталогу, пошук товарів, перехід на сторінку товару, реєстрацію, вхід і валідацію форм, додавання в кошик і wishlist у режимі без авторизації, сторінку порівняння, контактну форму, технічні файли SEO (robots.txt, sitemap.xml, manifest),

базові security headers, адаптивність на мобільному вигляді та базові перевірки продуктивності і доступності.

Аудит доступності через axe DevTools показав, що основні інтерактивні елементи мають достатню кольорову контрастність, поля форм містять відповідні label, а кнопки доступні з клавіатури. Незначне зниження оцінки Accessibility пов’язане з кількома місцями, де контрастність другорядного тексту і деяких бейджів акцій трохи нижча за рекомендовану — це залишено для подальшого доопрацювання.

3.4. Керівництво користувача

Підрозділ описує основні сценарії використання TechStore з точки зору кінцевого користувача. Розглядаються типові дії як для незареєстрованого відвідувача, так і для авторизованого користувача та адміністратора.

3.4.1. Перший вхід на сайт і навігація

При першому відкритті сайту користувач потрапляє на головну сторінку, яка автоматично адаптується під розмір екрана. На десктопі видно повну шапку з логотипом, основним меню категорій, полем пошуку і набором іконок дій (вхід, кошик, обране); нижче — головний банер, сітка популярних категорій, добірки товарів. На мобільному пристрої замість горизонтального меню відкривається бургер-іконка, а внизу екрана з'являється фіксований BottomNav з ключовими розділами.

Для перемикання між світлою і темною темою у шапці є відповідна іконка (місяць або сонце). Зміна теми відбувається миттєво, без перезавантаження сторінки, і запам'ятовується для наступних відвідувань.

3.4.2. Пошук і перегляд товарів

Для пошуку товару можна скористатися полем пошуку у шапці. По мірі введення тексту під полем з'являються підказки з мініатюрами, назвами та цінами найбільш релевантних товарів. Натискання на підказку відкриває картку товару; натискання Enter — повну сторінку результатів пошуку.

Альтернативний шлях — через каталог. У головному меню обирається категорія, далі — підкатегорія. На сторінці категорії можна застосувати фільтри ліворуч (бренд, ціна, наявність), обрати спосіб сортування зверху (за ціною, популярністю, новизною), переходити між сторінками результатів. Стан фільтрів зберігається у URL — посиланням можна поділитися або зберегти у закладки.

Картка товару містить галерею (з можливістю збільшення), повну інформацію про модель, перелік характеристик, відгуки попередніх покупців, рекомендації супутніх товарів. Внизу картки — блок схожих товарів. Натискання на «Додати у кошик» додає товар у кошик з миттєвим toast-сповіщенням; іконка серця додає у список бажань; іконка ваги — у порівняння.

3.4.3. Реєстрація та вхід

Для реєстрації нового облікового запису потрібно натиснути іконку користувача у шапці, обрати «Зареєструватися», ввести email, пароль (мінімум 8 символів), повторити пароль. Після успішної реєстрації користувач одразу авторизований і може користуватися всіма функціями.

Альтернативний шлях — вхід через Google. На сторінці входу є кнопка «Увійти через Google», що відкриває попап Firebase Authentication; обирається обліковий запис; після підтвердження користувач авторизується. Жодного додаткового заповнення форм не потрібно — ім'я та email беруться з Google-профілю.

Для відновлення пароля використовується посилання «Забули пароль?» на сторінці входу. Користувач вводить email, отримує лист з посиланням, переходить за ним, задає новий пароль.

3.4.4. Оформлення замовлення

Після того як товари додані у кошик, можна перейти до оформлення замовлення. Іконка кошика у шапці показує лічильник доданих товарів; натискання на неї відкриває сторінку кошика. Там можна змінити кількість або видалити позицію, ввести промокод, побачити розрахунок доставки і перейти до оформлення замовлення.

На сторінці checkout заповнюються контактні дані, обирається спосіб доставки — Нова пошта або кур’єрська доставка, вводиться або вибирається адресна інформація, обирається спосіб оплати — оплата при отриманні або вибір оплати картою, за бажанням застосовується промокод і залишається коментар. Натискання «Підтвердити замовлення» створює замовлення, очищає кошик і показує стан успішного оформлення.

У будь-який момент можна повернутися назад і змінити склад кошика або обрані параметри. Помилки валідації (наприклад, некоректний формат телефону) показуються одразу під полями і блокують відправлення форми.

3.4.5. Особистий кабінет

У особистому кабінеті авторизований користувач може переглянути історію замовлень з номерами, датами, статусами і сумами. Кожне замовлення можна відкрити для деталізації: повний склад товарів, спосіб доставки і оплати, статус кожної позиції, а також можливість оформити заявку на повернення товару.

Список бажань відображає всі товари, додані у wishlist. Тут є кнопки «додати у кошик» і «видалити». Список синхронізується з бекендом для авторизованих користувачів — додавши товар на смартфоні, можна побачити його з ноутбука.

На сторінці /profile редагуються персональні дані: ім’я, прізвище і телефон; email відображається як недоступне для редагування поле. Також доступна зміна пароля і підготовлений блок бонусної/loyalty-інформації. Історія замовлень розміщена окремо у /orders, а wishlist — у /wishlist.

3.4.6. Адміністративна панель

Доступ до адміністративної панелі мають лише користувачі з роллю admin. Точка входу — /admin (посилання у меню профілю). На дашборді — графіки динаміки замовлень, виручки, розподілу за статусами, список найпопулярніших товарів.

Розділ керування товарами — таблиця з мініатюрами, назвами, категоріями, цінами, залишками, кнопками дій. Створення нового товару — окрема форма з усіма полями: назва, категорія, бренд, опис, ціна, залишок, динамічний редактор технічних

характеристик, додавання зображень за URL або завантаженням файлу з комп'ютера. Редагування існуючого — та сама форма з заповненими полями.

Розділ керування замовленнями виводить список замовлень з фільтрацією за статусом і датою. При відкритті замовлення доступне змінення статусу замовлення (очікує, в обробці, відправлений, доставлений, скасований). Зміна статусу автоматично формує сповіщення користувачу.

Аналогічна функціональність — для категорій, користувачів, промокодів. Кожна сутність має свою таблицю зі сторінкуванням, фільтрацією, сортуванням і формами редагування.

3.4.7. Робота з помилками та повідомленнями

Під час роботи інтернет-магазину можливі ситуації, пов’язані з короткочасною втратою мережевого з’єднання, помилкою сервера, завершенням терміну дії сесії або зміною доступності товару. Коректна обробка таких ситуацій підвищує передбачуваність інтерфейсу та рівень довіри користувача до системи.

Toast-сповіщення через Sonner використовуються як основний спосіб інформування користувача про результат окремих дій. Це короткі повідомлення у правому верхньому кутку при успіху або помилці окремої дії: «Товар додано у кошик», «Замовлення створено», «Не вдалося завантажити список товарів». Повідомлення автоматично зникає через кілька секунд, але користувач може закрити його раніше.

Під час заповнення форм помилки валідації відображаються безпосередньо під відповідним полем: «Вкажіть телефон у форматі +380…», «Email некоректний», «Прийміть умови оферти». Таке розміщення повідомлень допомагає користувачеві швидко визначити поле, яке потребує виправлення.

Сторінки помилок використовуються для більш критичних ситуацій. Якщо товар не знайдено, відображається сторінка 404 з посиланням на каталог; за відсутності прав доступу — сторінка 403 з поясненням; у разі помилки на рівні застосунку — загальна сторінка помилки з кнопками «оновити» і «на головну». Усі

ці сторінки оформлено в єдиному стилі магазину з логотипом і навігацією, щоб користувач залишався в межах контексту застосунку.

3.4.8. Особливості мобільного використання

Як уже зазначалось, переважна частина трафіку TechStore припадає на мобільні пристрої. У мобільному використанні є кілька особливостей, про які корисно знати.

Нижній навігаційний бар. У десктопній шапці зосереджені логотип, меню категорій, пошук, іконки дій. На мобільному пристрої частина цих елементів переноситься у фіксований нижній бар: головна, каталог, пошук, кошик, профіль. Це означає, що для повернення на головну сторінку немає потреби скролити нагору — достатньо натиснути іконку дому внизу.

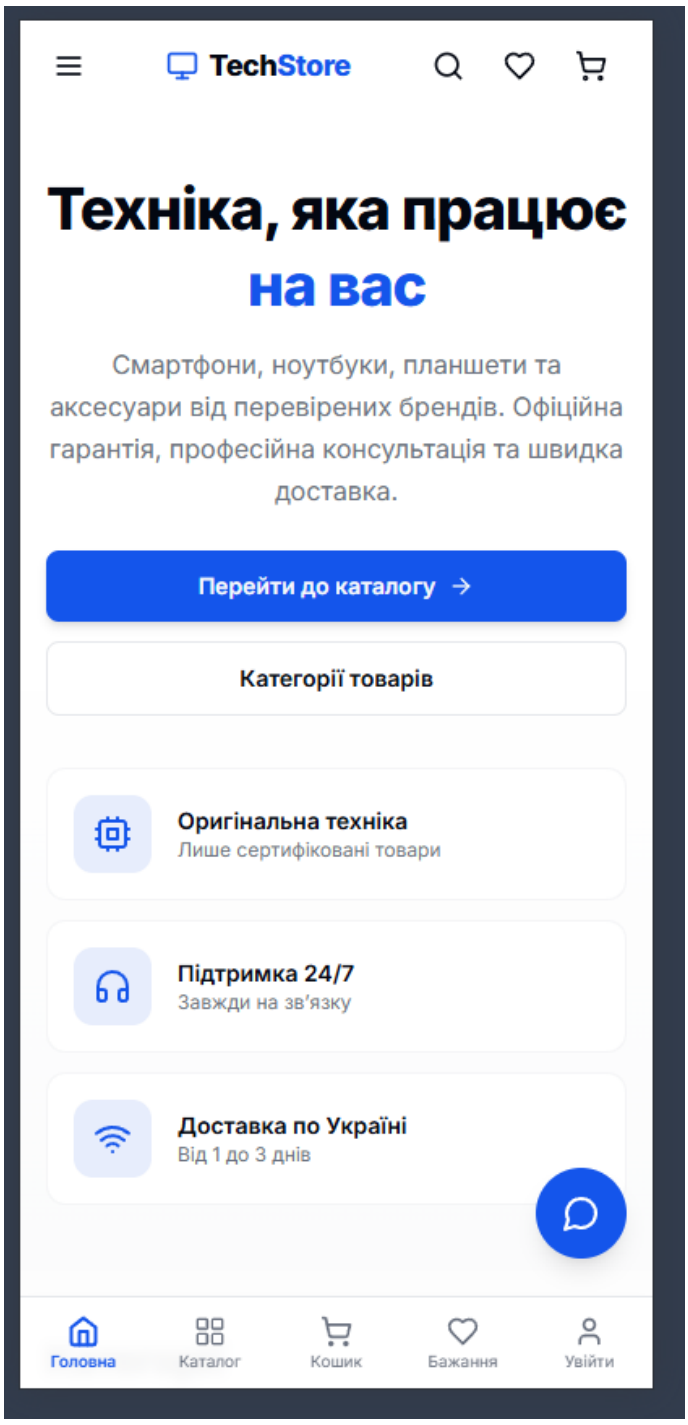


Рис. 3.16. Адаптивна мобільна версія TechStore

Жестова взаємодія. У горизонтальних блоках (хіти продажу, схожі товари, нещодавно переглянуті) працює природна прокрутка пальцем зі snap-фіксацією на кожному товарі. У галереї картки товару можна свайпати між зображеннями. У висувній мобільній панелі фільтрів — закриття свайпом донизу.

Збереження стану. Кошик і порівняння можуть зберігатися у локальному сховищі браузера через Zustand persist, а для авторизованого користувача

синхронізуються з бекендом. Wishlist працює через серверний API і потребує входу в систему. Нещодавно переглянуті товари зберігаються локально як повні Product-об’єкти з лімітом 12 позицій.

PWA. У проєкті підготовлено базовий manifest.ts із параметрами name, short_name, start_url, display, background_color, theme_color, orientation, categories і lang. Повноцінний офлайн-режим через service worker у поточній реалізації не реалізовано.

Висновки до розділу 3

У третьому розділі обґрунтовано вибір технологічного стеку, описано програмну реалізацію клієнтської частини TechStore та подано керівництво користувача, що охоплює основні сценарії взаємодії з системою.

Технологічний стек: Next.js 14 + React 18 + TypeScript 5 як базовий шар; Tailwind CSS + CSS Modules + next-themes для стилів; Axios + Firebase Auth для зв'язку з сервером і авторизації; React Hook Form + Zod для форм; Zustand з persist middleware для стану; Framer Motion, Lucide, Sonner, Recharts, date-fns для UX-деталей; Jest + React Testing Library + Playwright + ESLint для якості. Кожен вибір описано з альтернативами і конкретними причинами.

Вимоги до середовища — мінімальні. Vercel і Railway мають безкоштовні тарифи, достатні для проєкту такого масштабу. Для роботи зі сторони користувача: будь-який сучасний браузер, підключення від 1 Мбіт/с. Для розробки: Node.js 18+, Git, VS Code.

Програмна реалізація охоплює основні сторінки інтернет-магазину: головну, каталог, картку товару, кошик, checkout, профіль, замовлення, wishlist, порівняння та адміністративні розділи. Архітектура побудована по шарах — app, components, store, lib, types. Розгортання фронтенду організовано через Vercel, бекенду — через Railway. Цільові метрики продуктивності підтверджено Lighthouse-аудитом: Performance 97/100, LCP 2,6 с / 0,6 с, TBT 40 мс / 0 мс, CLS 0,001.

Керівництво користувача описує вісім основних сценаріїв — від першого відкриття сайту до роботи в адмінці. Розраховано на користувача без технічної підготовки.

ВИСНОВКИ

У кваліфікаційній роботі спроєктовано та реалізовано клієнтську частину інтернет-магазину електронних засобів TechStore. Підсумкові результати подано відповідно до поставлених у вступі завдань.

У першому розділі проаналізовано предметне середовище та три інтернет-магазини-аналоги: Rozetka, Comfy і Foxtrot. Під час порівняння було виділено рішення, корисні для TechStore: URL-синхронізацію фільтрів, нижню мобільну навігацію, адаптивні панелі фільтрації та зрозумілий сценарій оформлення замовлення. Також визначено рішення, яких у власному інтерфейсі варто уникати: надмірні рекламні попапи, перевантажені картки товарів і складну навігацію між вкладками.

На основі аналізу сформульовано функціональні та нефункціональні вимоги до системи. Вони охоплюють каталог, пошук, картку товару, кошик, checkout, особистий кабінет і адміністративний інтерфейс, а також вимоги до продуктивності, адаптивності, доступності та стабільності роботи.

У другому розділі описано дані, з якими працює клієнтська частина: товари, категорії, користувачі, кошик, замовлення, відгуки, купони, список бажань, порівняння та нещодавно переглянуті товари. Окремо подано модель взаємодії Next.js-клієнта з Express API, MongoDB/Mongoose, Firebase Authentication і сервісом Нової пошти. Алгоритмічна частина охоплює роботу кошика, синхронізацію фільтрів з URL, live-search, оновлення JWT-токенів, валідацію форм та оптимізацію зображень.

У третьому розділі описано програмну реалізацію фронтенд-частини. Стек сформовано на основі фактичного коду проєкту: Next.js 14, React 18, TypeScript, Tailwind CSS, CSS Modules, next-themes, Zustand, Axios, Firebase Authentication, Jest, React Testing Library і Playwright. Бекенд на Express, MongoDB/Mongoose та Railway розглядається як допоміжна частина, необхідна для демонстрації роботи інтерфейсу.

У практичній частині реалізовано основні сторінки інтернет-магазину: головну, каталог, картку товару, кошик, checkout, профіль, замовлення, wishlist, порівняння та

адміністративні розділи. Найбільшу увагу приділено сценаріям, у яких користувач часто змінює стан інтерфейсу: фільтрації каталогу, додаванню товарів у кошик, синхронізації кошика після авторизації, роботі з wishlist і оформленню замовлення. Розгортання фронтенду організовано через Vercel, бекенду — через Railway.

Продуктивність оцінювалася як через цільові значення Core Web Vitals, так і через застосовані механізми оптимізації: next/image, lazy-loading, адаптивну верстку, централізований API-клієнт і обмеження зайвих запитів під час пошуку та фільтрації. Lighthouse-аудит продакшн-версії підтвердив досягнення цільових показників: Performance 97 на мобільному та 100 на десктопі, LCP 2,6 с / 0,6 с, TBT 40 мс / 0 мс, CLS 0,001.

Подальший розвиток TechStore доцільно пов’язати з додаванням TanStack Query для серверного кешування, розширенням інтернаціоналізації, реалізацією service worker для офлайн-режиму, додаванням динамічних per-product metadata для сторінок товарів, завершенням платіжної інтеграції, розширенням покриття автоматизованими тестами та наповненням каталогу реальними товарами.

У результаті отримано функціональний навчально-практичний прототип інтернет-магазину. Він демонструє основні сценарії роботи клієнтської частини та може бути використаний як основа для подальшого розвитку після завершення платіжної інтеграції, розширення динамічних metadata для товарних сторінок, нарощення тестового покриття і наповнення каталогу реальними товарами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про електронну комерцію : Закон України від 03.09.2015 № 675-VIII // Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/675-19 (дата звернення: 15.01.2026).

2. EVO Marketplaces. Звіт ринку електронної комерції в Україні 2023 // EVO Marketplaces. URL: https://evo.business/ (дата звернення: 14.05.2026).

3. Promodo. Ukrainian eCommerce 2023: Market Insights & Trends // Promodo. URL: https://www.promodo.com/blog/the-state-of-ukrainian-e-commerce-in-2023-market-insights-from-promodos-research (дата звернення: 11.05.2026).

4. Google. Web Vitals: Essential metrics for a healthy site // Web.dev. URL: https://web.dev/articles/vitals (дата звернення: 11.05.2026).

5. Google / SOASTA. Milliseconds Make Millions // Think with Google. URL: https://www.thinkwithgoogle.com/_qs/documents/9757/Milliseconds_Make_Millions_report_hQYAbZJ.pdf (дата звернення: 11.05.2026).

6. Nielsen J. 10 Usability Heuristics for User Interface Design // Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 22.01.2026).

7. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley : New Riders, 2014. 216 p.

8. Шевчук О. А., Кушнір Н. О. Електронна комерція : навчальний посібник. Київ : КНЕУ, 2020. 280 с.

9. Хміль Ф. І., Хміль О. О. Електронна комерція : підручник. Львів : Магнолія, 2019. 232 с.

10. Web Content Accessibility Guidelines (WCAG) 2.1 // W3C Recommendation. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 25.01.2026).

11. W3C WAI. Understanding Success Criterion 2.5.5: Target Size // W3C. URL: https://www.w3.org/WAI/WCAG21/Understanding/target-size.html (дата звернення: 11.05.2026).

12. Next.js Documentation // Vercel Inc. URL: https://nextjs.org/docs (дата звернення: 02.02.2026).

13. React Documentation // Meta Open Source. URL: https://react.dev (дата звернення: 02.02.2026).

14. TypeScript Documentation // Microsoft Corporation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 03.02.2026).

15. Zustand Documentation // Poimandres. URL: https://zustand.docs.pmnd.rs/ (дата звернення: 04.02.2026).

16. Tailwind CSS Documentation // Tailwind Labs Inc. URL: https://tailwindcss.com/docs (дата звернення: 04.02.2026).

17. React Hook Form Documentation. URL: https://react-hook-form.com/get-started (дата звернення: 05.02.2026).

18. Zod: TypeScript-first schema validation. URL: https://zod.dev/ (дата звернення: 05.02.2026).

19. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 06.02.2026).

20. Firebase Authentication Documentation // Google. URL: https://firebase.google.com/docs/auth (дата звернення: 06.02.2026).

21. Playwright Documentation // Microsoft Corporation. URL: https://playwright.dev/docs/intro (дата звернення: 08.02.2026).

22. Jest Documentation // Meta Open Source. URL: https://jestjs.io/docs/getting-started (дата звернення: 08.02.2026).

23. Vercel Platform Documentation. URL: https://vercel.com/docs (дата звернення: 12.02.2026).

24. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 11.05.2026).

25. Express.js Documentation. URL: https://expressjs.com/ (дата звернення: 11.05.2026).

26. MongoDB Manual. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 11.05.2026).

27. Mongoose Documentation. URL: https://mongoosejs.com/docs/ (дата звернення: 11.05.2026).

ДОДАТКИ

Додаток А

Структура frontend-проєкту TechStore

```
frontend/
├── package.json                    # npm-скрипти та залежності
├── next.config.js                  # конфігурація Next.js, зображень і security headers
├── tsconfig.json                   # TypeScript-конфігурація та alias @/*
├── tailwind.config.js              # дизайн-токени та dark mode
├── jest.config.ts                  # налаштування Jest
├── playwright.config.ts            # налаштування e2e-тестів
└── src/
    ├── app/                        # App Router, сторінки, layout-и, SEO-файли
    ├── components/                 # повторно використовувані React-компоненти
    ├── lib/                        # API-клієнт, Firebase, утиліти
    ├── store/                      # Zustand-стори
    ├── types/                      # TypeScript-інтерфейси
    ├── __tests__/                  # unit/component-тести
    └── middleware.ts               # захист приватних маршрутів
```

Додаток Б

Основні маршрути клієнтської частини TechStore

Маршрут	Файл	Призначення
/	src/app/page.tsx	Головна сторінка
/products	src/app/products/page.tsx	Каталог товарів, фільтри, сортування, пагінація
/products/[slug]	src/app/products/[slug]/page.tsx	Картка товару
/cart	src/app/cart/page.tsx	Кошик користувача
/checkout	src/app/checkout/page.tsx	Оформлення замовлення
/login	src/app/login/page.tsx	Авторизація
/register	src/app/register/page.tsx	Реєстрація
/profile	src/app/profile/page.tsx	Профіль користувача та зміна пароля
/orders	src/app/orders/page.tsx	Перелік замовлень
/wishlist	src/app/wishlist/page.tsx	Список бажань
/compare	src/app/compare/page.tsx	Порівняння товарів
/admin	src/app/admin/page.tsx	Адміністративна панель
/admin/products	src/app/admin/products/page.tsx	Керування товарами
/admin/products/create	src/app/admin/products/create/page.tsx	Створення товару
/admin/orders	src/app/admin/orders/page.tsx	Керування замовленнями
/admin/users	src/app/admin/users/page.tsx	Керування користувачами
/admin/categories	src/app/admin/categories/page.tsx	Керування категоріями
/admin/coupons	src/app/admin/coupons/page.tsx	Керування купонами

Додаток В

Основні Zustand-стори клієнтської частини

Store	Основний стан	Основні дії	Призначення
authStore	user, accessToken, refreshToken, isAuthenticated, isLoading	login, register, googleLogin, logout, fetchProfile, syncStores	Стан авторизації користувача
cartStore	items, isLoading, isAuthenticated	fetchCart, addItem, removeItem, updateQuantity, clearCart, syncCartToServer, getTotalItems, getTotalPrice	Кошик та синхронізація з API
wishlistStore	items, isLoading	fetchWishlist, addToWishlist, removeFromWishlist, clearWishlist, isInWishlist	Список бажань авторизованого користувача
comparisonStore	comparisons	fetchComparisons, addToComparison, removeFromComparison, clearComparison, getComparisonByCategory	Порівняння товарів
recentlyViewedStore	items	addProduct, clearAll	Нещодавно переглянуті товари
languageStore	language	setLanguage	Мовні налаштування інтерфейсу
notificationStore	notifications, unreadCount	fetchNotifications, markAsRead, clearNotifications	Сповіщення користувача

Додаток Г

Основні npm-скрипти проєкту

Команда	Призначення
npm run dev	Запуск локального dev-сервера Next.js
npm run build	Production-збірка фронтенд-частини
npm start	Запуск зібраної продакшн-версії
npm run lint	Перевірка коду засобами ESLint
npm test	Запуск unit/component-тестів Jest
npm run test:watch	Запуск тестів у режимі спостереження
npm run test:coverage	Формування звіту про покриття тестами

Додаток Д

Ілюстративні матеріали, використані для підтвердження реалізації

Позначення	Матеріал	Місце подання
Рис. 1.1	UML-діаграма використання TechStore	Розділ 1
Рис. 1.2	Головна сторінка інтернет-магазину Rozetka	Розділ 1
Рис. 1.3	Сторінка категорії на Rozetka	Розділ 1
Рис. 1.4	Головна сторінка Comfy	Розділ 1
Рис. 1.5	Картка товару Comfy	Розділ 1
Рис. 1.6	Головна сторінка Foxtrot	Розділ 1
Рис. 1.7	Картка товару Foxtrot	Розділ 1
Рис. 2.1	Архітектурна діаграма системи TechStore	Розділ 2
Рис. 2.2	ER-подібна діаграма документної моделі MongoDB/Mongoose	Розділ 2
Рис. 2.3	Діаграма класів Zustand-сторів	Розділ 2
Рис. 2.4	Діаграма послідовностей оформлення замовлення	Розділ 2
Рис. 3.1	Структура каталогу frontend	Розділ 3
Рис. 3.2	Налаштування заголовків безпеки	Розділ 3
Рис. 3.3	Структура маршрутів у App Router	Розділ 3
Рис. 3.4	Кореневий layout проєкту	Розділ 3
Рис. 3.5	Головна сторінка TechStore	Розділ 3
Рис. 3.6	Сторінка каталогу	Розділ 3
Рис. 3.7	Сторінка товару	Розділ 3
Рис. 3.8	Сторінка кошика	Розділ 3
Рис. 3.9	Сторінка оформлення замовлення	Розділ 3
Рис. 3.10	Особистий кабінет користувача	Розділ 3
Рис. 3.11	Адміністративна панель TechStore	Розділ 3
Рис. 3.12	Реалізація cartStore	Розділ 3
Рис. 3.13	API-клієнт з interceptors	Розділ 3
Рис. 3.14	Форма входу TechStore	Розділ 3
Рис. 3.15	TechStore у продакшні	Розділ 3

Позначення	Матеріал	Місце подання
Рис. 3.16	Адаптивна мобільна версія TechStore	Розділ 3

Додаток Е

Ключові файли програмної реалізації

Файл	Функціональне призначення
src/store/cartStore.ts	методи addItem, updateQuantity, removeItem, clearCart, syncCartToServer
src/store/authStore.ts	ініціалізація користувача, збереження access-токена, logout
src/lib/api.ts	Axios-екземпляр, додавання Authorization-заголовка, обробка 401
src/middleware.ts	перевірка auth-token для приватних маршрутів
src/app/products/page.tsx	URL-синхронізація фільтрів і AbortController
src/components/layout/Header.tsx	пошук із debounce та endpoint /products/autocomplete
src/app/checkout/page.tsx	перевірка обов’язкових полів checkout та взаємодія з кошиком
backend/models/Product.model.js	Mongoose-модель товару з полями price, comparePrice, stock, specifications
backend/routes/upload.routes.js	завантаження зображень через multer

Додаток Ж

Посилання на репозиторій і продакшн-версію TechStore

Для перевірки та демонстрації результатів роботи доступні такі ресурси:

Репозиторій вихідного коду (GitHub): https://github.com/StasKoven/Duplomna_website

Продакшн-версія (Vercel): https://duplomna-website-p786.vercel.app/

Репозиторій містить два каталоги: frontend з клієнтською частиною на Next.js і backend із серверним API на Express та MongoDB/Mongoose. У README репозиторію наведено інструкції для локального запуску, опис змінних середовища і npm-скриптів. Розгортання фронтенду виконується через Vercel із preview-розгортаннями для кожного pull request; бекенд розгортається на Railway з підключенням до MongoDB Atlas.