МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»

(повне найменування вищого навчального закладу)
Відділення телекомунікацій та електронних систем
(назва відділення)
Циклова комісія комп'ютерних наук
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
фахового молодшого бакалавра

на тему: Розробка вебсайту магазину «ByteStore»

Виконав: студент 4 курсу, групи КН-423
спеціальності 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

Ратуш Андрій Петрович
(прізвище та ініціали)

Керівник Галина МАРЦІЯШ
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ратушу Андрію Петровичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка вебсайту магазину «ByteStore»

керівник роботи Марціяш Галина Ярославівна,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студенткою роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: JavaScript; фреймворки та бібліотеки: PHP, HTML, CSS, JavaScript; база даних: MySQL; зовнішні сервіси: Cloudinary, Vercel, Render., стандарти IEEE 29148-2018, IEEE 29119, ДСТУ 8302:2015.

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

1 Загальний розділ

1.1 Аналітичний огляд існуючих рішень

1.2 Технічне завдання

1.2.1 Найменування та область застосування

1.2.2 Призначення розробки

1.2.3 Вимоги до функціоналу вебзастосунку

1.2.4 Вимоги до програмної документації

1.2.5 Техніко-економічні показники

1.2.6 Стадії та етапи розробки

1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка структури сайту і вебсторінок

2.2 Створення та верстка сторінок сайту

2.3 Розробка структури бази даних сайту

2.4 Програмування сайту

2.4.1 Написання клієнтської частини

2.4.2 Написання admin частини

2.5 Тестування вебсайту

3 Спеціальний розділ

3.1 Розміщення сайту в Інтернеті

3.2 Інструкція з обслуговування та наповнення сайту

3.3 Інструкція з популяризації та підтримки сайту

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань веб-сайту магазину «ByteStore»

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості веб-сайту магазину «ByteStore»

4.7 Розрахунок ціни для веб-сайту магазину «ByteStore»

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Організація охорони праці на підприємстві

5.2 Вимоги до ручного інструменту для роботи з електрич ною проводкою.

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – вебсайту.

5. Перелік графічного матеріалу:

1. UML-діаграма варіантів використання програми
2. Структурна схема головної сторінки сайту
3. ER-діаграма бази даних вебсайту
4. Таблиця техніко-економічних показникі

6. Консультанти розділів роботи

Розділ	**Ім'я, прізвище та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	.06.2026	
12	Захист кваліфікаційної роботи	.06.2026	

7. Дата видачі завдання: 05 березня 2026 р.

Студент _____ Андрій РАТУШ
(підпис)

Керівник роботи _____ Галина МАРЦІЯШ
(підпис)

ЗМІСТ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка вебсайту магазину «ByteStore».

Метою кваліфікаційної роботи є розробка вебсайту електронної комерції для продажу ліцензійного програмного забезпечення.

Пояснювальна записка складається з п’яти основних розділів, висновків, переліку посилань та додатків.

У загальній частині описано аналітичний огляд існуючих рішень у сфері інтернет-магазинів, визначено призначення програмного продукту та сформульовано технічне завдання на розробку вебсайту.

У другому розділі представлено процес створення вебсайту: розробку структури сайту і web-сторінок, створення та верстку сторінок, проєктування структури бази даних, програмування клієнтської та адміністративної частин, а також тестування web-застосунку.

У спеціальній частині описано інструкцію з розміщення сайту в Інтернеті, порядок обслуговування та наповнення сайту, а також заходи з популяризації, резервного копіювання, моніторингу й підтримки актуальності вебсайту.

Розрахунок вартості розробки та економічної ефективності наведено в економічній частині.

Основні питання охорони праці, техніки безпеки та екологічних вимог розглянуто в п’ятому розділі.

Обсяг пояснювальної записки — 98.

До складу кваліфікаційної роботи входить графічна частина, яка складається з чотирьох аркушів формату А1: структурної схеми вебсайту, ER-діаграми бази даних, UML-діаграми використання та таблиці техніко-економічних показників.

ANNOTATION

Qualification Work Topic: Development of the «ByteStore» Online Store Website.

The purpose of the qualification work is to develop an e-commerce website for the sale of licensed software products.

The explanatory note consists of five main sections, conclusions, references, and appendices.

The general section contains an analytical review of existing solutions in the field of online stores, defines the purpose of the software product, and formulates the technical requirements for website development.

The second section presents the website development process, including the design of the website structure and web pages, creation and layout of pages, database structure design, programming of the client and administrative parts, as well as testing of the web application.

The special section describes the instructions for website deployment on the Internet, procedures for website maintenance and content management, as well as measures for promotion, backup, monitoring, and maintaining the relevance of the website.

The calculation of development costs and economic efficiency is presented in the economic section.

The main issues of occupational health and safety, workplace safety, and environmental requirements are considered in the fifth section.

The explanatory note comprises 98 pages.

The qualification work includes a graphical part consisting of four A1 sheets: a website structural diagram, a database ER diagram, a UML use case diagram, and a table of technical and economic indicators.

ВСТУП

У сучасному світі інформаційних технологій електронна комерція є одним із найдинамічніших напрямів розвитку бізнесу. Використання вебтехнологій дозволяє підприємствам розширювати ринки збуту, підвищувати доступність товарів і послуг для споживачів та автоматизувати процеси продажу.

Особливо актуальним є створення спеціалізованих інтернет-магазинів для реалізації програмного забезпечення, оскільки цифрові продукти не потребують фізичного зберігання та можуть бути доставлені користувачеві в електронному вигляді.

Актуальність теми полягає у зростанні попиту на ліцензійне програмне забезпечення та необхідності забезпечення зручного, безпечного й ефективного процесу його придбання через мережу Інтернет. Розробка сучасного вебсайту магазину дозволяє автоматизувати облік товарів, спростити процес оформлення замовлень, покращити взаємодію з клієнтами та підвищити конкурентоспроможність підприємства.

Метою кваліфікаційної роботи є розробка вебсайту магазину «ByteStore», призначеного для продажу ліцензійного програмного забезпечення.

Об’єктом дослідження є процес продажу ліцензійного програмного забезпечення через мережу Інтернет.

Предметом дослідження є методи, засоби та технології розробки вебзастосунків електронної комерції.

Практичне значення роботи полягає у створенні функціонального вебсайту магазину «ByteStore», який може бути використаний для організації продажу програмного забезпечення, управління каталогом товарів та взаємодії з клієнтами в онлайн-середовищі.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Сучасний ринок програмного забезпечення та цифрових ліцензій активно розвивається. Зростання попиту на легальне ПЗ серед фізичних осіб та підприємств зумовлює появу спеціалізованих онлайн-платформ, що надають зручний доступ до широкого каталогу продуктів. Огляд існуючих рішень дозволяє виявити їх переваги та недоліки, а також визначити напрям для розробки вебсайту магазину «ByteStore».

На сьогодні найбільш поширеними платформами для онлайн-продажу програмного забезпечення та ліцензій є такі рішення:

Rozetka.ua — великий українській маркетплейс, що продає, серед іншого, ліцензійне програмне забезпечення. Платформа надає широкий асортимент товарів, проте є універсальним маркетплейсом і не спеціалізується виключно на цифровому ПЗ. Пошук конкретного програмного продукту або ключа активації може бути ускладнений через великий обсяг несуміжних категорій. Відсутній фільтр за типом ліцензії (OEM, Retail, Volume).

SoftGroup.ua — спеціалізований інтернет-магазин ліцензійного програмного забезпечення на українському ринку. Пропонує продукти Microsoft, ESET, Kaspersky, Adobe та інших виробників. Перевагами є вузька спеціалізація та наявність корпоративних рішень. Недоліками є застарілий інтерфейс, відсутність особистого кабінету з повноцінною історією замовлень, обмежений функціонал пошуку та фільтрації.

Steam — міжнародна платформа цифрової дистрибуції ігрового ПЗ від Valve Corporation. Є зручним та добре відомим рішенням, однак орієнтована виключно на ігрову індустрію і не підходить для продажу бізнес-застосунків чи офісного ПЗ.

Microsoft Store — офіційна крамниця Microsoft, що пропонує ліцензійні продукти компанії. Платформа відзначається надійністю та інтеграцією з операційною системою Windows, проте обмежена лише власними продуктами

Microsoft і не є відкритим майданчиком для різних виробників.

Порівняльний аналіз вищеописаних платформ наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Характеристика	**Rozetka.ua**	**SoftGroup.ua**	**Steam**
Спеціалізація на ПЗ	Ні	Так	Частково
Особистий кабінет	Так	Ні	Так
Адмін-панель	Так	Обмежена	Так
Сучасний UI	Так	Ні	Так
REST API	Так	Ні	Так
JWT-автентифікація	Ні	Ні	Ні

За результатами аналізу виявлено, що існуючі рішення або є надто універсальними маркетплейсами без спеціалізації на ПЗ, або мають застарілий інтерфейс і обмежений функціонал. Розробка вебсайту «ByteStore» дозволить створити сучасну спеціалізовану платформу, яка поєднає зручний інтерфейс, JWT-автентифікацію, повноцінний особистий кабінет та адміністративну панель управління товарами.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Повне найменування програмного виробу: «ByteStore — вебсайт продажу програмного забезпечення та цифрових ліцензій». Скорочене найменування: ByteStore.

Область застосування системи охоплює два напрями використання. По-перше, система орієнтована на фізичних осіб та суб'єктів підприємницької діяльності, які здійснюють придбання ліцензійного програмного забезпечення

через мережу Інтернет. По-друге, платформа забезпечує функціонування адміністративного персоналу магазину в частині управління каталогом товарів, обробки замовлень та адміністрування облікових записів користувачів.

Об'єктом застосування є комерційна вебплатформа, що забезпечує реалізацію ліцензій та дистрибутивів програмних продуктів різних категорій, зокрема: операційних систем, офісних пакетів, антивірусного програмного забезпечення, графічних редакторів та інших програмних рішень.

1.2.2 Призначення розробки

Експлуатаційне призначення системи полягає в автоматизації процесу онлайн-продажу програмного забезпечення та цифрових ліцензій. Система, яка розробляється повинна забезпечити покупцям структурований доступ до каталогу товарів, можливість оформлення замовлень та відстеження їх поточного статусу. Адміністративний персонал отримує інструментарій для управління товарним асортиментом, контентом платформи та обліковими записами користувачів.

Функціональне призначення системи реалізується через сукупність взаємопов'язаних технологічних рішень. Серверну частину буде побудовано на основі фреймворку Spring Boot з архітектурою REST API, що забезпечує стандартизовану взаємодію між клієнтом і сервером. Автентифікацію та розмежування прав доступу буде реалізовано засобами Spring Security із застосуванням механізму JWT-токенів. Для зберігання та управління реляційними даними використовується СУБД PostgreSQL. Клієнтська частина розроблена у вигляді односторінкового застосунку (Single Page Application) на основі бібліотеки React із застосуванням Tailwind CSS та Material UI, що забезпечує формування сучасного та адаптивного інтерфейсу користувача.

1.2.3 Вимоги до функціоналу вебсайту

Вхідні дані

Вхідні дані системи наведено в таблиці 1.2.

Таблиця 1.2 – Перелік і опис вхідних даних

Назва вхідних даних	**Форма подання**	**Джерело даних**	**Опис**
Реєстраційні дані користувача	JSON (HTTP запит)	Користувач (форма реєстрації)	Ім'я, email, пароль
Дані авторизації	JSON (HTTP запит)	Користувач (форма входу)	Email та пароль для JWT-аутентифікації
Дані про товар	JSON (HTTP запит)	Адміністратор (адмін-панель)	Назва, опис, ціна, категорія, зображення, ліцензійний ключ
Параметри пошуку та фільтрації	Query-параметри URL	Користувач (каталог)	Назва товару, категорія, діапазон цін
Дані замовлення	JSON (HTTP запит)	Користувач (кошик/оформлення)	Список товарів, кількість, адреса доставки

Вихідна інформація

Вихідними даними системи є:

– каталог товарів у форматі JSON, що повертається клієнтській частині через REST API;

– сторінка товару з детальним описом, ціною та кнопкою додавання до кошика;

– підтверджене замовлення з унікальним ідентифікатором та статусом обробки;

– JWT-токен доступу, що надається після успішної автентифікації;

– особистий кабінет із переліком та деталями попередніх замовлень;

– адміністративний перегляд переліку товарів, замовлень та користувачів.

Опис функцій та обмежень

Система виконуватиме такі основні функції:

– реєстрація та автентифікація користувача — введені дані перевіряються на відповідність формату (email-валідація, мінімальна довжина пароля 8 символів); після успішного входу повертається JWT-токен із терміном дії 24 години;

– перегляд каталогу та пошук товарів — система повертає список товарів із фільтрацією за категорією та ціновим діапазоном, підтримується пагінація (20 товарів на сторінку);

– управління кошиком — користувач може додавати товари, змінювати кількість або видаляти їх; стан кошика зберігається у локальному сховищі браузера (LocalStorage);

– оформлення замовлення — формується запис у базі даних зі статусом «Нове»; після успішного оформлення користувач отримує підтвердження;

– особистий кабінет — відображається список замовлень із датою, сумою та статусом; дані доступні лише авторизованому власнику;

– адміністрування — авторизований адміністратор може додавати, редагувати та видаляти товари, а також змінювати статус замовлень.

Обмеження системи: неавторизовані користувачі мають доступ лише до каталогу та сторінок окремих товарів; оформлення замовлення та особистий кабінет доступні виключно після авторизації; адміністративний розділ доступний лише для облікових записів із роллю ADMIN.

Часові характеристики

Час відповіді на запит до REST API при стандартному навантаженні не повинен перевищувати 2 секунди. Час завантаження сторінки каталогу (з пагінацією) — не більше 3 секунд. Час обробки запиту на оформлення замовлення — не більше 3 секунд. Жорстких обмежень реального часу система не потребує, оскільки не є системою керування виробничим процесом.

Вимоги до надійності

З метою забезпечення надійного функціонування системи передбачаються такі заходи:

– ієрархічна структура серверного додатку із розподілом на рівні

Controller, Service, Repository;

– валідація вхідних даних на рівні контролерів засобами Spring Validation (@Valid, @NotBlank, @Email, @Size);

– обробка виняткових ситуацій через глобальний ExceptionHandler (@ControllerAdvice) із поверненням зрозумілих JSON-відповідей про помилки;

– захист від несанкціонованого доступу через Spring Security та JWT; захищені ендпоінти потребують валідного токена в заголовку Authorization;

– захист адміністративних ендпоінтів через перевірку ролі (@PreAuthorize("hasRole('ADMIN')"));

– збереження цілісності даних засобами транзакцій Spring Data JPA (@Transactional).

Умови експлуатації

Функціональні можливості системи:

– перегляд каталогу товарів — доступний для всіх відвідувачів;

– реєстрація та вхід — режим автентифікації на основі JWT;

– кошик та оформлення замовлення — доступно для авторизованих користувачів;

– особистий кабінет та історія замовлень — доступно для авторизованих користувачів;

– адмін-панель управління товарами та замовленнями — доступно лише для ролі ADMIN.

Режим роботи програми: цілодобовий (24/7), веброзгортання на сервері. Взаємодія користувача з системою відбувається через веббраузер у режимі «запит-відповідь» за REST-архітектурою. Клієнтська частина є SPA-застосунком, що не потребує перезавантаження сторінки при переходах між розділами завдяки react-router-dom.

Вимоги до складу технічних засобів для серверної частини: процесор із тактовою частотою не менше 1 ГГц, оперативна пам'ять не менше 2 ГБ, дисковий простір не менше 10 ГБ. Для клієнтської частини: будь-який сучасний веббраузер

із підтримкою ES6+ (Google Chrome, Mozilla Firefox, Microsoft Edge версій 2022 року і новіших).

Вимоги до програмної сумісності: серверна частина функціонує під управлінням ОС Linux (Ubuntu 22.04 або новіша) або Windows Server 2019+, з JDK 17 або вище. База даних — PostgreSQL версії 14 або вище. Клієнтська частина не залежить від ОС та виконується у веббраузері.

1.2.4 Вимоги до програмної документації

Склад програмних документів визначається відповідно до Методичних вказівок на виконання кваліфікаційної роботи [1] та включає:

- пояснювальну записку до кваліфікаційної роботи;
- технічне завдання (підрозділ 1.2 пояснювальної записки);
- опис структури бази даних (підрозділ 2.3 пояснювальної записки);
- тексти програм (у додатках до пояснювальної записки);
- інструкцію з розміщення сайту в Інтернеті (підрозділ 3.1);
- інструкцію з обслуговування та наповнення сайту (підрозділ 3.2);
- інструкцію з популяризації та підтримки сайту (підрозділ 3.3).

Вимоги до оформлення текстів програм:

- кожний програмний модуль або клас повинен мати початковий блок коментарів із зазначенням призначення, переліку вхідних та вихідних даних та умов використання;
- коментарі у коді повинні бути стислими та чіткими, написаними зрозумілою мовою;
- для REST-контролерів обов'язкова наявність Swagger/OpenAPI-анотацій для автоматичної документації API;
- кожний модуль (клас) повинен мати одну точку входу та одну точку виходу.

1.2.5 Техніко-економічні показники

Трудові витрати на розробку програмного виробу становлять орієнтовно 3 людино-місяці, що включає: проєктування архітектури та бази даних — 0,5 місяця; розробку серверної частини (REST API, Spring Boot, Spring Security) — 1 місяць; розробку клієнтської частини (React SPA) — 1 місяць; тестування, налагодження та оформлення документації — 0,5 місяця.

Витрати машинного часу на розробку та тестування становлять орієнтовно 200 годин машинного часу на персональному комп'ютері розробника.

1.2.6 Стадії та етапи розробки

Розробка вебсайту ByteStore виконується у таких стадіях та етапах:

Стадія 1. Технічне завдання та аналіз (1 тиждень).

- вивчення предметної області та аналіз існуючих рішень;
- формування вимог та складання технічного завдання;
- проєктування структури бази даних та загальної архітектури системи.

Стадія 2. Розробка серверної частини (3 тижні).

- налаштування проєкту Spring Boot, підключення залежностей;
- розробка моделей даних та JPA-сутностей;
- розробка REST API для товарів, категорій, замовлень та користувачів;
- налаштування Spring Security та реалізація JWT-автентифікації;
- налаштування PostgreSQL та написання міграцій бази даних.

Стадія 3. Розробка клієнтської частини (3 тижні).

- ініціалізація React-проєкту, налаштування react-router-dom;
- розробка компонентів каталогу, картки товару, кошика;
- розробка форм реєстрації, входу та особистого кабінету;
- розробка адмін-панелі для управління товарами та замовленнями;
- стилізація інтерфейсу засобами Tailwind CSS та Material UI.

Стадія 4. Тестування та налагодження (1 тиждень).

– функціональне тестування всіх модулів системи;

– тестування безпеки (перевірка JWT, ролей доступу);

– усунення виявлених помилок та оптимізація.

Стадія 5. Оформлення документації (1 тиждень).

– написання пояснювальної записки;

– підготовка інструкцій з розміщення та обслуговування сайту.

1.2.7 Порядок тестування та прийому

Перевірка працездатності програмного комплексу здійснюється за допомогою контрольного прикладу, що охоплює повний цикл роботи системи.

Контрольний приклад включає такі кроки:

– реєстрація нового користувача через форму реєстрації; перевіряється: збереження запису в базі даних, хешування пароля, повернення JWT-токена;

– авторизація зареєстрованого користувача; перевіряється: видача JWT-токена, коректний відгук при невірних даних;

– перегляд каталогу товарів із застосуванням фільтра за категорією «Антивіруси» та ціновим діапазоном 500–2000 грн; перевіряється: коректна вибірка з БД, правильна пагінація;

– додавання двох товарів до кошика та зміна кількості одного з них; перевіряється: коректне збереження стану кошика;

– оформлення замовлення авторизованим користувачем; перевіряється: створення запису замовлення зі статусом «Нове», відображення у особистому кабінеті;

– авторизація адміністратора та додавання нового товару через адмін-панель; перевіряється: збереження товару в БД, його відображення у каталозі;

– спроба доступу до адміністративного ендпоінту без токена або з токеном звичайного користувача; перевіряється: повернення відповіді 401 Unauthorized або 403 Forbidden.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури сайту і вебсторінок

Вебсайт ByteStore є спеціалізованою платформою електронної комерції, що забезпечує продаж ліцензійного програмного забезпечення та цифрових ліцензійних ключів. Основна цінність платформи полягає у миттєвій доставці цифрового товару: після підтвердження та оплати замовлення покупець отримує доступ до унікальних ліцензійних ключів безпосередньо в особистому кабінеті без необхідності фізичної доставки.

Призначення сайту охоплює два напрями. Перший — обслуговування покупців: фізичних осіб та суб'єктів підприємницької діяльності, які здійснюють пошук, вибір та придбання ліцензійного програмного забезпечення через мережу Інтернет. Другий — забезпечення роботи адміністративного персоналу магазину в частині управління асортиментом товарів, обробки замовлень та адміністрування облікових записів.

Основу дизайн-системи складають: глибокий синій колір (#1E4D8C) як первинний колір бренду, що асоціюється з надійністю і технологічністю; електричний блакитний (#00C2FF) для кнопок заклику до дії та акцентних елементів; світло-сірий фон (#F5F7FA) для сторінок та білий (#FFFFFF) для карток. Радіус закруглення кнопок становить 24px, карток — 8px. Типографіка базується на гарнітурах Inter (заголовки) та Roboto (основний текст).

Структура вебсайту ByteStore організована за принципом розмежування зон доступу та функціональних призначень сторінок. Усі сторінки розподілено на чотири зони: публічна зона, зона авторизованого користувача, адміністративна зона та системні сторінки зображено в таблиці 2.1.

Навігаційна ієрархія сайту побудована наступним чином: кореневим вузлом є головна сторінка (/), від якої відгалужуються розділи першого рівня — каталог товарів, сторінки авторизації, особистий кабінет та адміністративна панель. Кожен розділ першого рівня містить підсторінки другого рівня згідно зі своєю

функціональністю.

Таблиця 2.1 – Карта сторінок вебсайту ByteStore

Зона	Сторінка (компонент)	URL-маршрут	Призначення
1	2	3	4
Публічна	HomePage	/	Лендінг: hero-секція, категорії, популярні товари, блок довіри
	CatalogPage	/catalog	Каталог із sidebar-фільтрами, сортуванням, пагінацією
	ProductPage	/catalog/:id	Деталі товару: опис, ціна, специфікації, схожі товари
	CartPage	/cart	Кошик із qty-степером, підсумком та полем промокоду
	LoginPage	/login	Форма входу до системи (JWT-автентифікація)
	RegisterPage	/register	Форма реєстрації нового облікового запису
	ForgotPasswordPage	/forgot-password	Запит на відновлення пароля через email
	AboutPage	/about	Інформація про магазин
	ContactPage	/contacts	Контактна інформація та форма зворотного зв'язку
Користувач (USER)	AccountPage	/account	Хаб особистого кабінету зі sidebar-навігацією

Прдовження таблиці 2.1

1	2	3	4
	ProfilePage	/account/profile	Редагування профілю (ім'я, прізвище, телефон)
	OrdersPage	/account/orders	Таблиця замовлень з фільтром за статусом
	OrderDetailPage	/account/orders/:id	Деталі замовлення; ліцензійні ключі з blur/reveal
	CheckoutPage	/checkout	3-кроковий checkout: контакти → оплата → підтвердження
	OrderSuccessPage	/order-success/:id	Сторінка успіху після оформлення замовлення
ADMIN	AdminDashboard	/admin	KPI-картки, зведена таблиця товарів і замовлень
	AdminProductsPage / Form	/admin/products	CRUD товарів; модальна форма додавання/редагування
	AdminCategoriesPage	/admin/categories	CRUD категорій програмного забезпечення
	AdminOrdersPage / Detail	/admin/orders	Список усіх замовлень; inline зміна статусу
	AdminUsersPage	/admin/users	Перегляд та управління обліковими записами
Системні	NotFoundPage	* (будь-який)	Сторінка 404 — маршрут не знайдено
	ForbiddenPage	/403	Сторінка 403 — доступ заборонено

Проєктування інтерфейсу здійснено на основі виконаного аналізу вимог технічного завдання Нижче описано макети основних сторінок сайту.

Головна сторінка (HomePage). Сторінка виконує функцію лендінгу і складається з п'яти функціональних секцій. Перша секція — hero-блок — містить великий заголовок, підзаголовок та дві кнопки: «Переглянути каталог» (primary, акцентний колір) і «Популярні категорії» (secondary). Друга секція відображає 6

категорій товарів у вигляді іконок із підписами (Windows, Office, Antivirus, Adobe, macOS, Інше). Третя секція містить сітку з 4 популярних товарів. Четверта секція — trust-strip — відображає чотири переваги магазину: «Безпечна оплата», «Миттєва доставка ключа», «Офіційні ліцензії», «Підтримка 24/7». П'ята секція — Footer із логотипом, навігаційними посиланнями та контактами.

Сторінка каталогу (CatalogPage). Макет сторінки побудований за схемою «бічна панель + основний контент». Ліва бічна панель (sidebar) містить фільтри: чекбокси для вибору категорії, слайдер цінового діапазону, перемикач наявності. Основна зона відображає рядок управління (сортування, кількість результатів, перемикач вигляду) та тривколонну сітку карток товарів по 12 штук на сторінці з пагінацією знизу. Передбачено стан порожніх результатів (empty state) з відповідним повідомленням.

Сторінка товару (ProductPage). Макет реалізовано за двоколонним принципом. Ліва колонка містить велике зображення товару. Права колонка — назву товару, бейдж категорії, ціну (великим шрифтом акцентного кольору), короткий та розгорнутий опис, кнопку «Додати до кошика», таблицю ключових характеристик (платформа, версія, мова, тип ліцензії, спосіб доставки) та accordion-секцію «Як отримати ключ?». Нижче розміщено горизонтальний список схожих товарів.

Сторінка кошика (CartPage). Сторінка реалізована у двоколонному макеті: зліва — список товарів у кошику з мініатюрами, назвами, степером кількості та кнопкою видалення; справа — картка підсумку замовлення з сумою, полем промокоду та кнопкою «Оформити замовлення». Передбачено окремий стан порожнього кошика з ілюстрацією та кнопкою «До каталогу».

Сторінки авторизації (LoginPage, RegisterPage). Обидві сторінки побудовані на основі центрованої картки завширшки 440px. Форма входу містить поля email та пароль, посилання «Забули пароль?» та кнопку підтвердження. Форма реєстрації — поля імені, прізвища, email, пароля та підтвердження пароля. Під формою реєстрації розміщено рядок соціального підтвердження: «Вже 5 000+ покупців довіряють ByteStore».

Особистий кабінет (AccountPage). Макет кабінету реалізовано за схемою «sidebar + content area». Ліва панель містить аватар користувача, ім'я та навігаційне меню (Профіль / Мої замовлення / Вийти). Сторінка замовлень відображає таблицю з колонками: номер замовлення, дата, перелік товарів, сума, статус. Статус відображається кольоровими бейджами: жовтий (Нове), синій (В обробці), зелений (Виконано), червоний (Скасовано). Сторінка деталей замовлення містить ліцензійні ключі у розмитому вигляді (blur) з кнопкою «Показати ключ» та кнопкою «Копіювати».

Сторінка оформлення замовлення (CheckoutPage). Макет реалізовано як триетапний покроковий процес з індикатором прогресу вгорі: «Контакти → Оплата → Підтвердження». Перший крок містить поля для введення контактної інформації. Другий — вибір методу оплати та введення даних картки. Третій — підсумок замовлення та кнопка підтвердження. На всіх кроках відображаються елементи безпеки (іконка замку, значок HTTPS).

Адміністративна панель (AdminDashboard). Адмін-зона оформлена з темним sidebar-навігатором, що контрастує з основним сайтом і відразу сигналізує про перехід до службового інтерфейсу. Головна сторінка панелі відображає 4 KPI-картки: «Замовлень сьогодні», «Виручка», «Нових користувачів», «Товарів в каталозі». Нижче — зведена таблиця товарів із мініатюрами, назвами, цінами та кнопками дій. Форма додавання/редагування товару відкривається як модальне вікно шириною 560px.

Дизайн сайту спроєктований для трьох контрольних точок (breakpoints) відповідно до стандарту Tailwind CSS:

– desktop (1280px і вище) — основний цільовий розмір; трьохколонна сітка каталогу, sidebar відображається постійно;

– tablet (768px) — sidebar фільтрів каталогу переміщується у верхню панель, двоколонна сітка товарів;

– mobile (375px) — одноколонний макет, нижня навігаційна панель (BottomNav) замінює header-меню, товари відображаються у один стовпець.

Адаптивність реалізована виключно засобами утилітарних класів Tailwind CSS без медіа-запитів у CSS-файлах. Для мобільних пристроїв передбачено компонент BottomNav з іконками: каталог, пошук, кошик, профіль.

2.2 Створення та верстка сторінок вебсайту

Клієнтська частина вебсайту ByteStore розроблена як Single Page Application (SPA) з використанням бібліотеки React версії 18.3.1 та збирача Vite версії 5.4.6. Розробка здійснюється у середовищі Visual Studio Code з розширеннями ESLint та Prettier. Vite забезпечує миттєвий старт dev-сервера на порту 5173 та Hot Module Replacement (HMR) — оновлення змінених модулів у браузері без повного перезавантаження сторінки.

Стилізація інтерфейсу реалізована засобами двох взаємодоповнюючих інструментів: Tailwind CSS версії 3.4.13 та Material UI версії 5.16.7. Tailwind CSS застосовується як основна система стилізації через утилітарні класи безпосередньо в JSX-розмітці. Material UI використовується для складних інтерактивних компонентів: діалогових вікон, полів введення, таблиць, перемикачів та бічних панелей.

Управління станом HTTP-запитів здійснюється засобами TanStack React Query версії 5.56.2, яка забезпечує кешування, фонове оновлення даних та стани завантаження/помилки. Маршрутизація між сторінками реалізована через react-router-dom версії 6.26.2 без перезавантаження сторінки.

Вихідні файли клієнтської частини організовані у каталозі frontend/src/ за функціональним принципом. Кожен підкаталог відповідає за певний шар архітектури застосунку(див. таблиця 2.2).

Таблиця 2.2 – Файлова структура фронтенд-частини проєкту

Каталог / файл	Призначення
src/api/	HTTP-клієнт Axios та функції запитів до REST API бекенду (auth.js, products.js, orders.js, admin.js)
src/components/	Перевикористовувані UI-компоненти: Button, Input, Badge, Logo, ProductCard, ProductCardSkeleton, Pagination, Toaster, Header, Footer, BottomNav
src/context/	Провайдери глобального стану: AuthContext (автентифікація), CartContext (кошик), ToastContext (сповіщення), WishlistContext (список бажань)
src/pages/	Компоненти-сторінки (по одному на маршрут): публічні сторінки, підкаталог account/ для кабінету, підкаталог admin/ для панелі адміністратора
src/router/	Декларація маршрутів (AppRouter.jsx), захист маршрутів (ProtectedRoute.jsx, AdminRoute.jsx)
src/hooks/	Кастомні React-хуки: useAuth, useCart, useProducts та інші
src/utils/	Допоміжні функції: форматування цін, валідатори, константи
src/main.jsx	Точка входу застосунку: ReactDOM.createRoot, обгортки MUI Theme, QueryClient, BrowserRouter
src/index.css	Глобальні стилі: CSS-змінні дизайн-системи та директиви підключення Tailwind CSS

Усі компоненти бібліотеки розроблені як функціональні React-компоненти з використанням хуків. Кожен компонент розміщується в окремому файлі з назвою у форматі PascalCase (наприклад, ProductCard.jsx). Перед розробкою будь-якого компонента обов'язково аналізується файл frontend/design/Readme.md для

визначення точних токенів кольорів, відступів та параметрів компонента.

Компонент Button реалізований у п'яти варіантах: primary (акцентний блакитний фон), blue (темно-синій фон), secondary (прозорий з рамкою), ghost (без рамки) та disabled (неактивний стан). Радіус закруглення — 24px відповідно до дизайн-системи. Стан hover реалізований через клас Tailwind hover:opacity-90, стан натискання — через active:scale-97.

Компонент Input підтримує три стани: normal (стандартний вигляд з сірою рамкою), focus (рамка змінюється на акцентний блакитний з ефектом свічення через box-shadow) та error (червона рамка та текст помилки під полем). Реалізовано через Tailwind-класи focus:ring-2 та умовне додавання класу border-red-500.

Компонент ProductCard відображає зображення товару, бейдж категорії, назву (до двох рядків), коротке описання та ціну. Кнопка «Додати до кошика» займає повну ширину картки. Стан hover картки реалізований через Tailwind-клас hover:shadow-lg з переходом transition-shadow duration-200. Компонент ProductCardSkeleton є відповідним станом завантаження з анімованим пульсуванням (animate-pulse) без використання спінерів.

Компонент Header реалізований як sticky-елемент (position: sticky, top: 0) з z-index вище основного контенту. Містить логотип зліва, навігаційні посилання по центру, поле пошуку та іконку кошика з лічильником (Badge) та кнопку входу/профілю справа. На мобільних пристроях навігаційні посилання приховуються, а нижня панель (BottomNav) забезпечує основну навігацію.

Компонент Toaster реалізує систему сповіщень (toast notifications) через ToastContext. Сповіщення відображаються у верхньому правому куті сторінки, підтримують типи success, error та info, автоматично зникають через 3 секунди та можуть бути закриті вручну.

Компонент Pagination відображає кнопки навігації між сторінками у вигляді квадратів розміром 38×38 пікселів. Поточна сторінка виділяється акцентним кольором. Реалізований через стан сторінки у react-router-dom query-параметрах, що забезпечує збереження навігаційного стану при оновленні сторінки.

Головна сторінка (HomePage.jsx) верстається з використанням компонентів

PublicLayout (обгортка з Header та Footer), власних секцій HeroSection, CategoriesSection, PopularProductsSection та TrustBadgesSection. Hero-секція реалізована з градієнтним фоном від первинного кольору (#1E4D8C) до темнішого відтінку. Сітка популярних товарів побудована на CSS Grid з Tailwind-класами grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-6.

Сторінка каталогу (CatalogPage.jsx) реалізована з умовним відображенням sidebar для десктопу та кнопкою-перемикачем фільтрів для мобільних пристроїв. Отримання даних здійснюється через хук useProducts з TanStack React Query, який автоматично кешує результати та керує станами завантаження і помилки. Під час завантаження замість сітки товарів відображається сітка компонентів ProductCardSkeleton.

Сторінка товару (ProductPage.jsx) отримує дані товару через useQuery з ключем ['product', id]. Accordion-секція «Як отримати ключ?» реалізована через локальний useState для управління станом розкриття. Компонент схожих товарів отримує дані окремим запитом з фільтрацією за категорією поточного товару.

Сторінка кошика (CartPage.jsx) не виконує HTTP-запитів — вона зчитує стан із CartContext. Qty-степер реалізований через кнопки зі збільшенням/зменшенням кількості через функцію updateQty з контексту. Загальна сума перераховується автоматично через обчислюваний параметр totalPrice з CartContext.

Сторінка оформлення замовлення (CheckoutPage.jsx) реалізована як компонент з внутрішнім лічильником кроку (useState для step: 1 | 2 | 3). Кожен крок відображає відповідну форму. Після успішного відправлення POST /api/orders відбувається перенаправлення на /order-success/:id через useNavigate з react-router-dom.

Особистий кабінет реалізований через AccountLayout.jsx як обгортку з sidebar-навігацією. Дочірні сторінки (ProfilePage, OrdersPage, OrderDetailPage) рендеряться через <Outlet /> react-router-dom. На мобільних пристроях sidebar кабінету замінюється горизонтальним меню з прокруткою.

Адміністративна панель реалізована через AdminLayout.jsx з темним sidebar (фон #1E2A3A). Форма додавання та редагування товару (AdminProductForm.jsx)

відкривається як MUI Dialog шириною 560px, що забезпечує незмінність стану фонової сторінки. Таблиці списків замовлень та товарів реалізовані через MUI DataGrid із підтримкою сортування та пагінації на стороні клієнта.

Маршрутизація застосунку декларована в компоненті AppRouter.jsx із використанням react-router-dom v6. Всі маршрути визначені одним деревом конфігурації з використанням вкладених маршрутів (nested routes) для розділів account та admin.

Захист маршрутів реалізований через два компоненти-обгортки. Компонент ProtectedRoute перевіряє наявність автентифікованого користувача через isAuthenticated з AuthContext. У разі відсутності автентифікації виконується перенаправлення на /login з передачею поточного шляху через параметр state для повернення після входу. Компонент AdminRoute додатково перевіряє роль isAdmin та перенаправляє на /403 при недостатньому рівні доступу.

Стан автентифікації зберігається в AuthContext та ініціалізується при завантаженні застосунку зчитуванням JWT-токена з localStorage за ключем bytestore_token. Axios-інтерцептор запитів автоматично додає заголовок Authorization: Bearer <token> до кожного HTTP-запиту. Інтерцептор відповідей перехоплює статус 401, виконує logout() та перенаправляє на /login.

У процесі розробки застосовано кілька ключових методичних прийомів стилізації, що забезпечують узгодженість інтерфейсу.

Дизайн-токени визначені у файлі tailwind.config.js шляхом розширення стандартної теми Tailwind. Усі кольори, радіуси закруглень та відступи завантажуються безпосередньо з дизайн-системи handoff-пакету, що виключає розбіжності між макетом і реалізацією. Назви кастомних токенів відповідають іменуванню у файлі frontend/design/Readme.md.

CSS-змінні для ключових параметрів дизайн-системи оголошені у файлі src/index.css через :root-селектор. Це дозволяє використовувати їх як у Tailwind-класах через var(), так і в компонентах MUI через систему тематизації.

Теми Material UI налаштовані в src/main.jsx через createTheme з переписаними первинними та акцентними кольорами відповідно до дизайн-

системи ByteStore. Це забезпечує однорідний вигляд компонентів MUI (кнопки, поля введення, таблиці) у стилі, узгодженому з загальним дизайном.

Анімації та переходи реалізовані виключно через Tailwind-класи transition-all duration-200 ease-in-out для інтерактивних елементів. Стани скелетону завантаження реалізовані через animate-pulse. Це забезпечує плавність взаємодії без підключення сторонніх анімаційних бібліотек.

2.3 Розробка структури бази даних вебсайту

Для зберігання даних вебсайту ByteStore обрано реляційну систему управління базою даних PostgreSQL версії 14 і вище. Вибір обумовлений рядом технічних і практичних переваг цієї СУБД. По-перше, PostgreSQL є повноцінною ACID-сумісною реляційною СУБД з відкритим вихідним кодом, що забезпечує надійність транзакцій при обробці замовлень та призначенні ліцензійних ключів. По-друге, нативна підтримка типів NUMERIC(12,2) та TIMESTAMPTZ забезпечує точне зберігання грошових сум і часових міток із урахуванням часового поясу. По-третє, PostgreSQL має розвинені механізми індексування (B-tree, GIN), які дозволяють ефективно виконувати пошук та фільтрацію товарів у каталозі.

Взаємодія між додатком і базою даних організована через Spring Data JPA з реалізацією Hibernate ORM. Це дозволяє описувати структуру таблиць у вигляді Java-сутностей (entity-класів) та генерувати SQL-запити автоматично. Міграції схеми бази даних виконуються засобами Flyway версії 10.x, що гарантує відтворюваність стану БД у будь-якому середовищі розгортання.

База даних вебсайту ByteStore містить шість таблиць, що відображають предметну область інтернет-магазину цифрових ліцензій. Перелік таблиць із коротким описом призначення кожної наведено у таблиці 2.3.

Таблиця 2.3 – Перелік таблиць бази даних ByteStore

№	Назва таблиці	Призначення
1	users	Облікові записи покупців та адміністраторів магазину
2	categories	Категорії програмного забезпечення (операційні системи, антивіруси тощо)
3	products	Товари каталогу — програмні продукти з описом, ціною та залишком на складі
4	orders	Замовлення покупців із поточним статусом та загальною сумою
5	order_items	Позиції замовлення — конкретні товари у складі кожного замовлення
6	license_keys	Ліцензійні ключі активації — прив'язані до товару та (після продажу) до замовлення

Нижче наведено детальний опис структури кожної таблиці бази даних із зазначенням типів даних, обмежень та призначення полів (див. таблиці 2.4-2.9).

Індекси таблиці products: idx_products_category (за полем category_id) — прискорює вибірку товарів за категорією; idx_products_price (за полем price) — прискорює сортування та фільтрацію за ціновим діапазоном.

Таблиця 2.4 – Структура таблиці users

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	email	VARCHAR(255)	String	Електронна адреса; унікальна (UNIQUE), NOT NULL; використовується як логін
3	password_hash	VARCHAR(255)	String	Хеш пароля (BCrypt, strength 12); NOT NULL; відкритий текст пароля не зберігається
4	first_name	VARCHAR(100)	String	Ім'я користувача; може бути порожнім
5	last_name	VARCHAR(100)	String	Прізвище користувача; може бути порожнім
6	phone_number	VARCHAR(20)	String	Номер телефону; nullable; додано міграцією V2
7	role	VARCHAR(20)	Role (enum)	Роль: USER або ADMIN; DEFAULT USER'
8	created_at	TIMESTAMPTZ	OffsetDateTime	Дата і час реєстрації; DEFAULT OW();

Таблиця 2.5 – Структура таблиці products

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	name	VARCHAR(255)	String	Назва товару (наприклад, «Windows 11 Pro»); NOT NULL
3	description	TEXT	String	Повний опис товару; nullable
4	price	NUMERIC(12,2)	BigDecimal	Ціна в гривнях; NOT NULL; 2 знаки після коми
5	image_url	VARCHAR(500)	String	URL-адреса зображення товару; nullable

Таблиця 2.6 – Структура таблиці categories

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	name	VARCHAR(100)	String	Відображувана назва категорії (наприклад, «Операційні системи»); NOT NULL
3	slug	VARCHAR(100)	String	URL-сумісний ідентифікатор категорії (наприклад, «os»); UNIQUE, NOT NULL

Таблиця 2.7 – Структура таблиці orders

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	user_id	BIGINT	User (FK)	Зовнішній ключ → users.id; NOT NULL; замовлення завжди прив'язане до користувача
3	status	VARCHAR(20)	OrderStatus (enum)	Статус замовлення: NEW | PROCESSING | COMPLETED | CANCELLED; DEFAULT 'NEW'
4	total	NUMERIC(12,2)	BigDecimal	Загальна сума замовлення; NOT NULL; фіксується на момент оформлення
5	created_at	TIMESTAMPTZ	OffsetDateTime	Дата і час оформлення замовлення; DEFAULT NOW()

Індекси таблиці orders: idx_orders_user (за полем user_id) — прискорює

вибірку замовлень конкретного користувача; idx_orders_status (за полем status) — прискорює фільтрацію за статусом в адмін-панелі.

Таблиця 2.8 – Структура таблиці order_items

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	order_id	BIGINT	Order (FK)	Зовнішній ключ → orders.id; ON DELETE CASCADE — позиції видаляються разом із замовленням
3	product_id	BIGINT	Product (FK)	Зовнішній ключ → products.id; NOT NULL
4	quantity	INTEGER	Integer	Кількість одиниць товару у позиції; NOT NULL
5	price_at_purchase	NUMERIC(12,2)	BigDecimal	Ціна товару на момент оформлення замовлення; NOT NULL; фіксує historical-вартість незалежно від майбутніх змін ціни

Індекс таблиці order_items: idx_order_items_order (за полем order_id) — прискорює завантаження всіх позицій конкретного замовлення.

Таблиця 2.9 – Структура таблиці license_keys

№	Назва поля	Тип SQL	Тип Java	Обмеження та примітки
1	id	BIGSERIAL	Long	Первинний ключ, автоінкремент
2	product_id	BIGINT	Product (FK)	Зовнішній ключ → products.id; NOT NULL; ключ належить конкретному продукту
3	key_value	VARCHAR(500)	String	Рядок ліцензійного ключа активації; UNIQUE, NOT NULL; унікальність забезпечує неможливість подвійного продажу
4	order_id	BIGINT	Order (FK)	Зовнішній ключ → orders.id; ON DELETE SET NULL; nullable — NULL означає, що ключ ще не продано
5	assigned_at	TIMESTAMPTZ	OffsetDateTime	Дата і час призначення ключа замовленню; nullable — заповнюється при обробці замовлення

Між таблицями бази даних встановлено шість зв'язків через зовнішні ключі. Опис усіх зв'язків наведено у таблиці 2.10.

Таблиця 2.10 – Зв'язки між таблицями бази даних

Таблиця (від)	Таблиця (до)	Тип зв'язку	Деталі та поведінка при видаленні
products	categories	M:1	Кожен товар належить до однієї категорії. При видаленні категорії поле category_id у товарах обнуляється (ON DELETE SET NULL)
orders	users	M:1	Кожне замовлення належить одному користувачеві. Поле user_id є обов'язковим (NOT NULL)
order_items	orders	M:1	Кожна позиція замовлення належить одному замовленню. При видаленні замовлення всі його позиції видаляються каскадно (ON DELETE CASCADE)
order_items	products	M:1	Позиція фіксує товар і його ціну на момент покупки. Зв'язок необов'язковий для видалення — товар може бути видалений після продажу
license_keys	products	M:1	Кожен ключ прив'язаний до конкретного продукту. NOT NULL — ключ без продукту не існує
license_keys	orders	M:1 (opt.)	Ключ призначається замовленню при обробці. order_id = NULL означає вільний (непроданий) ключ. При видаленні замовлення ключ звільняється (ON DELETE SET NULL)

Індекси таблиці license_keys: idx_license_keys_product (за полем product_id) — прискорює пошук вільних ключів для конкретного товару при обробці замовлення; idx_license_keys_order (за полем order_id) — прискорює завантаження ключів замовлення в особистому кабінеті.

Наведена схема зв'язків забезпечує цілісність даних на рівні бази даних без додаткової логіки у сервісному шарі застосунку. Особливо важливими є два правила поведінки при видаленні. По-перше, видалення замовлення автоматично видаляє всі його позиції (ON DELETE CASCADE для order_items), але звільняє ліцензійні ключі замість їх видалення (ON DELETE SET NULL для license_keys) — це дозволяє повторно продати звільнені ключі. По-друге, видалення категорії не видаляє товари, а лише від'єднує їх від категорії (ON DELETE SET NULL для products.category_id), що запобігає втраті товарних даних.

ER-діаграму наведено на плакаті 2026.КВР.122.423.17.00.00 БД. Діаграма відображає всі шість таблиць, їх поля та зв'язки між ними.

Взаємодія між Java-застосунком і базою даних PostgreSQL організована за чотирирівневою схемою: JPA-сутності → репозиторії → сервіси → контролери.

На рівні JPA-сутностей кожна таблиця відображається в окремий Java-клас, анотований @Entity. Поля класу відповідають стовпцям таблиці. Зв'язки між таблицями відображаються через анотації @ManyToOne та @OneToMany з вказівкою стратегії завантаження (FetchType). Для зменшення надлишкового коду використовується бібліотека Lombok: анотації @Getter, @Setter та @NoArgsConstructor автоматично генерують відповідні методи під час компіляції.

На рівні репозиторіїв кожна сутність має відповідний інтерфейс, що розширює JpaRepository<T, ID>. Spring Data JPA автоматично генерує реалізацію стандартних операцій (findById, save, delete, findAll). Для специфічних запитів використовуються іменовані методи (наприклад, findByEmail для UserRepository) або анотація @Query з JPQL.

Перетворення між JPA-сутностями та DTO-об'єктами (Data Transfer Object) виконується засобами MapStruct версії 1.6.3. MapStruct генерує реалізацію mapper-інтерфейсів під час компіляції, що усуває необхідність написання ручного коду

перетворення та не впливає на продуктивність у runtime. У проєкті реалізовано маперів ProductMapper та CategoryMapper.

На рівні сервісів методи, що змінюють стан бази даних (створення замовлення, призначення ліцензійних ключів, зміна статусу), анотовані @Transactional. Це гарантує атомарність операцій: наприклад, при оформленні замовлення створення запису у таблиці orders, записів у order_items та призначення license_keys відбуваються в межах однієї транзакції — у разі будь-якої помилки всі зміни відкочуються.

2.4 Програмування вебсайту

Завданням програмування вебсайту ByteStore є об'єднання всіх застосованих технологій у цілісний функціональний продукт та надання користувачам зручного інтерфейсу для взаємодії з платформою. Розробка здійснювалась у середовищі Visual Studio Code з використанням системи контролю версій Git [2]. Монорепозиторій організовано у каталозі ByteStore/ з двома незалежними підпроєктами: backend/ (серверна частина) та frontend/ (клієнтська частина).

Відповідно до архітектурного підходу, описаного у технічній документації проєкту, серверна частина реалізована за трирівневою монолітною архітектурою (Controller → Service → Repository) на основі Spring Boot[3] , тоді як клієнтська частина є Single Page Application (SPA) на базі React[3]. Взаємодія між частинами здійснюється виключно через REST API з уніфікованим JSON-конвертом у форматі { "data": …, "error": null }.

2.4.1 Написання клієнтської частини

Клієнтська частина вебсайту ByteStore розроблена мовою JavaScript стандарту ES2022[5] з використанням бібліотеки React 18.3.1[4]. Збірка та запуск у режимі розробки здійснюється через Vite 5.4.6[6], який забезпечує Hot Module Replacement (HMR) — миттєве оновлення змінених модулів у браузері без

перезавантаження сторінки. Написання скриптів виконувалось у редакторі Visual Studio Code з розширеннями ESLint та Prettier[7].

Налаштування HTTP-взаємодії. Основою комунікації між клієнтом та сервером є бібліотека Axios 1.7.7[8]. Базовий екземпляр клієнта визначено у файлі src/api/client.js. Він налаштований з базовою адресою, що зчитується зі змінної середовища VITE_API_URL або повертається до http://localhost:8080/api за замовчуванням. Ключовою особливістю є два інтерцептори. Request-інтерцептор автоматично додає заголовок Authorization: Bearer <token> до кожного запиту, витягуючи JWT з localStorage за ключем bytestore_token. Response-інтерцептор виконує дві функції: розпаковує уніфікований конверт ApiResponse, повертаючи клієнту значення response.data.data, та при отриманні статусу 401 очищає збережений токен і перенаправляє користувача на сторінку /login. Лістинг ключової логіки налаштування Axios-клієнта наведено у Додатку А.

Управління станом автентифікації. Глобальний стан входу реалізовано через AuthContext (див. Додаток Б) із використанням React Context API[4]. Контекст надає компонентам об'єкт { user, token, login, logout, isAuthenticated, isAdmin. При ініціалізації застосунку контекст зчитує збережений JWT-токен з localStorage та декодує з нього поля email і role. Функція login() зберігає токен та оновлює стан, logout() видаляє токен та скидає стан до початкового.

Управління кошиком. CartContext (див. Додаток В) реалізує персистентний стан кошика через localStorage[9] за ключем bytestore_cart. Контекст зберігає масив позицій кошика у форматі { productId, name, price, quantity та надає методи addItem, removeItem, updateQty, clearCart. Обчислювані параметри totalPrice та itemCount розраховуються через useMemo при кожній зміні масиву позицій. Під час оформлення замовлення виклик clearCart() спустошує кошик та видаляє запис з localStorage.

Маршрутизація. Усі маршрути декларовано у файлі src/router/AppRouter.jsx із використанням react-router-dom v6[10]. Для захисту маршрутів реалізовано компоненти ProtectedRoute та AdminRoute. ProtectedRoute перевіряє значення isAuthenticated з AuthContext — при хибному значенні виконується Navigate до

/login з передачею поточного шляху через state. AdminRoute додатково перевіряє isAdmin та при false перенаправляє на /403. Вкладені маршрути (nested routes) для

Сторінка каталогу. CatalogPage.jsx (див.Додаток Г) є найбільш функціонально насиченою сторінкою публічної зони. Отримання даних здійснюється через хук useQuery бібліотеки TanStack React Query 5.56.2 [11], який кешує результати за ключем ['products', filters] та автоматично керує станами завантаження, помилки та успіху. Параметри фільтрації (categoryId, minPrice, maxPrice, q, page, size, sort) синхронізуються з URL через useSearchParams, що забезпечує можливість збереження та відтворення стану фільтрів при поверненні на сторінку. Під час завантаження замість сітки товарів відображаються 12 компонентів ProductCardSkeleton з анімацією animate-pulse.

Сторінка оформлення замовлення. CheckoutPage.jsx реалізована як триетапний покроковий процес з внутрішнім лічильником кроку (useState для step: 1 | 2 | 3). Дані форм зберігаються у стані formData через useReducer для уникнення надмірного числа useState-викликів. На третьому кроці викликається функція createOrder з api/orders.js, яка відправляє POST-запит до /api/orders із тілом { items: [...] }. Після успішної відповіді викликається clearCart() та виконується navigate('/order-success/' + orderId) через useNavigate[10].

Компонентна бібліотека. Усі перевикористовувані елементи інтерфейсу зосереджені у каталозі src/components/. Компонент Button реалізовано у п'яти варіантах через об'єкт variants зі стилями Tailwind CSS[12]. Компонент Badge відображає кольорові статусні позначки для замовлень через об'єкт statusConfig, що відображає рядкові значення статусу (NEW, PROCESSING, COMPLETED, CANCELLED) у відповідні кольори та підписи. Компонент Toaster реалізує чергу toast-сповіщень через ToastContext з автоматичним видаленням через 3 секунди засобами setTimeout.

Стилізація. Конфігурація Tailwind CSS[12] розширена у файлі tailwind.config.js кастомними токенами дизайн-системи: кольорами (#1E4D8C, #00C2FF тощо), радіусами (btn: '24px', card: '8px') та шрифтами (Inter, Roboto) відповідно до handoff-пакету з каталогу frontend/design/. Компоненти Material

UI[13] тематизовані через createTheme у файлі src/main.jsx з переписаними первинним та акцентним кольорами.

2.4.2 Написання серверної частини

Серверну частину вебсайту ByteStore реалізовано мовою Java 21[14] на основі фреймворку Spring Boot 3.5.14[3]. Збірка та управління залежностями здійснюється через Apache Maven 3.9.16[15]. Середовище розробки — IntelliJ IDEA або Visual Studio Code з розширенням Java Extension Pack. Пакетна структура організована за функціональними шарами: config, controller, service, repository, entity, dto, mapper, security, exception.

Налаштування безпеки та JWT-автентифікація. Безпека застосунку базується на Spring Security 6[16] у stateless-режимі. Конфігурацію відкритих та захищених маршрутів визначено у класі SecurityConfig. До публічних маршрутів належать: POST /api/auth/**, GET /api/products/**, GET /api/categories/**, а також /swagger-ui/** та /v3/api-docs/**. Маршрути /api/orders/** та /api/users/me вимагають ролі USER; модифікуючі операції з товарами та категоріями вимагають ролі ADMIN.

Генерацію та валідацію JWT-токенів реалізовано у класі JwtUtil із використанням бібліотеки JJWT 0.12.6[17]. Токен підписується алгоритмом HMAC-SHA256, містить claim-поля sub (email) та role (роль із БД), термін дії визначається параметром jwt.expiration-ms з application-local.properties. Клас JwtFilter (Додаток Д) розширює OncePerRequestFilter та виконується для кожного HTTP-запиту: витягує токен з заголовку Authorization, валідує підпис та термін дії, завантажує UserDetails та встановлює об'єкт Authentication у SecurityContext.

Рівень контролерів. Кожен REST-контролер є класом, анотованим @RestController та @RequestMapping. Контролери не містять бізнес-логіки — вони лише приймають HTTP-запити, валідують вхідні DTO анотацією @Valid та делегують виконання відповідному сервісу. Відповідь формується через статичні фабричні методи класу ApiResponse<T>: ApiResponse.success(data) та ApiResponse.error(code, message). Усі контролери анотовані @Tag (SpringDoc), усі

ендпоінти — @Operation та @ApiResponse для автоматичної генерації документації Swagger UI [18] за адресою http://localhost:8080/swagger-ui.html.

Рівень сервісів. Сервіси реалізовані за шаблоном «інтерфейс + реалізація»: наприклад, AuthService та AuthServiceImpl. Всі методи, що змінюють стан бази даних, анотовані @Transactional[13]. Особливої уваги заслуговує OrderServiceImpl (див. Додаток Е) — клас, що реалізує логіку оформлення замовлення. Метод createOrder виконує п'ять кроків у межах однієї транзакції: валідацію наявності товарів, перевірку достатності кількості ліцензійних ключів, створення запису Order, створення записів OrderItem із фіксацією роточної ціни (priceAtPurchase), та призначення вільних LicenseKey (де order_id IS NULL) до щойно створеного замовлення.

Рівень репозиторіїв. Кожна JPA-сутність має відповідний інтерфейс-репозиторій, що розширює JpaRepository<T, ID>. Spring Data JPA[19] автоматично генерує реалізацію стандартних операцій. Для специфічних запитів використовуються іменовані методи або анотація @Query. Наприклад, LicenseKeyRepository містить метод findFreeByProductId, що повертає список незакріплених ключів через JPQL-запит: SELECT k FROM LicenseKey k WHERE k.product.id = :id AND k.order IS NULL.

Маппінг DTO. Перетворення між JPA-сутностями та DTO-об'єктами виконується засобами MapStruct 1.6.3[20]. Mapper-інтерфейси анотовані @Mapper(componentModel = "spring"), що дозволяє Spring автоматично впровадити реалізацію через @Autowired. MapStruct генерує реалізацію під час компіляції (compile-time), що унеможливлює помилки маппінгу у runtime та не впливає на продуктивність застосунку.

Обробка виключень. Централізована обробка помилок реалізована через GlobalExceptionHandler — клас, анотований @RestControllerAdvice. Кожен тип доменного виключення (ProductNotFoundException, EmailAlreadyExistsException, OrderAccessDeniedException тощо) перехоплюється відповідним методом з анотацією @ExceptionHandler та перетворюється на відповідь з правильним HTTP-статусом та уніфікованим тілом ApiResponse.error(). Це забезпечує консистентний

формат усіх помилкових відповідей без дублювання логіки обробки у кожному контролері[3].

Міграції бази даних. Схема бази даних управляється засобами Flyway 10.x [21]. Файл V1__init_schema.sql (Додаток Ж) містить DDL-скрипт створення всіх шести таблиць (users, categories, products, orders, order_items, license_keys), зовнішніх ключів та індексів. Flyway автоматично застосовує міграції при кожному запуску застосунку, порівнюючи контрольні суми файлів з таблицею flyway_schema_history. Файл V2__add_phone_to_users.sql додає колонку phone_number до таблиці users.

Документація API. Специфікацію REST API сформовано автоматично засобами SpringDoc OpenAPI 2.8.9[18] на основі анотацій контролерів. Swagger UI доступний за адресою http://localhost:8080/swagger-ui.html та дозволяє тестувати ендпоінти безпосередньо з браузера. Для захищених ендпоінтів налаштовано схему bearerAuth через клас OpenApiConfig — це дозволяє вводити JWT-токен у Swagger UI та виконувати автентифіковані запити під час тестування розробки.

2.5 Тестування вебсайту

Метою тестування вебсайту ByteStore є перевірка відповідності розробленого програмного забезпечення вимогам технічного завдання, виявлення та усунення дефектів інтерфейсу, логіки клієнтської частини та взаємодії з REST API бекенду.

У процесі тестування застосовано такі методи:

– мануальне (ручне) тестування — основний метод для даної роботи; тестувальник особисто виконує сценарії використання системи та порівнює фактичні результати з очікуваними;

– функціональне тестування — перевірка відповідності реалізованих функцій вимогам ТЗ;

– тестування навігації та маршрутизації — перевірка коректності переходів між сторінками, захисту маршрутів та редиректів;

– тестування форм та валідації — перевірка реакції форм на коректні,

некоректні та порожні вхідні дані;

– тестування безпеки (surface-рівень) — перевірка неможливості доступу до захищених сторінок без авторизації та без ролі ADMIN;

– тестування адаптивності — перевірка відображення сторінок у браузері на різних розмірах вікна (desktop 1440px, tablet 768px, mobile 375px).

Тестування виконувалось у браузері Google Chrome версії 125+ на операційній системі Windows 11, сервер Vite dev запущений на порту 5173, бекенд — на порту 8080.

Тестування проводилось за наскрізними сценаріями використання системи (user flows), що охоплюють основні шляхи покупця та адміністратора. Нижче описано кожен сценарій з результатами.

Сценарій 1. Реєстрація та вхід до системи.

На сторінці реєстрації (/register) перевірено введення коректних даних (ім'я, прізвище, email, пароль). Після відправлення форми сервер повертає JWT-токен, який зберігається в localStorage за ключем bytestore_token, а користувач перенаправляється на головну сторінку. Авторизований стан відображається у Header: кнопка «Увійти» замінюється на ім'я користувача та посилання на кабінет. Вигляд сторінки реєстрації показано на рисунку 2.2.

Перевірено також сценарії негативного тестування: введення вже зареєстрованого email призводить до відображення повідомлення про помилку «Користувач з таким email вже існує» під полем форми; відправлення форми з порожніми полями блокується клієнтською валідацією HTML5 required та повідомленням Tailwind-стилізованого компонента Input у стані error.

На сторінці входу (/login) перевірено авторизацію з коректними обліковими даними та з некоректним паролем. У другому випадку відображається toast-сповіщення з текстом помилки «Невірний email або пароль». Вигляд сторінки входу показано на рисунку 2.3.

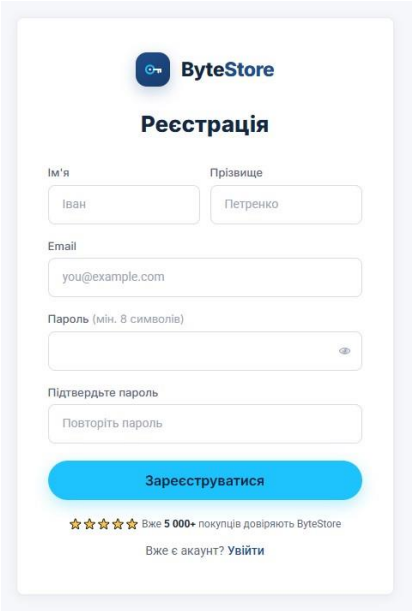


Рисунок 2.2 – Сторінка реєстрації вебсайту ByteStore

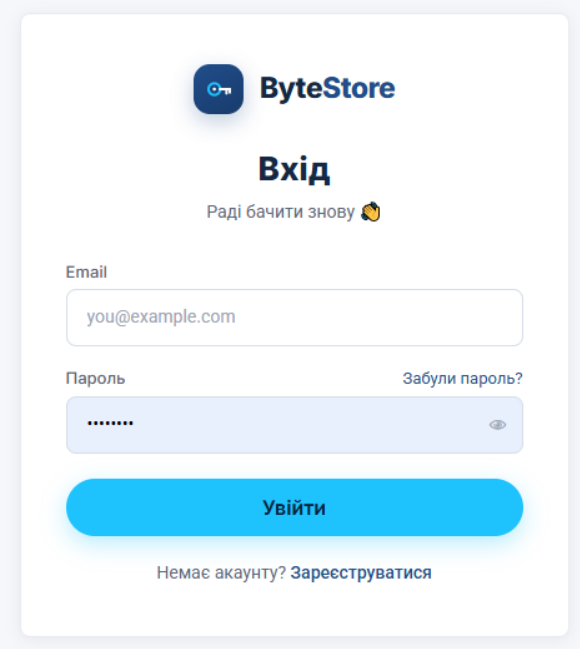


Рисунок 2.3 – Сторінка входу вебсайту ByteStore

Сценарій 2. Перегляд каталогу та фільтрація товарів.

Сторінка каталогу (/catalog) завантажується зі skeleton-заглушками на час отримання даних від API. Після завантаження відображається сітка карток товарів. Перевірено роботу фільтрів: вибір категорії оновлює список товарів без перезавантаження сторінки, URL при цьому оновлюється параметром

categoryId=<id>, що дозволяє зберегти стан фільтрів при поверненні до сторінки. Фільтрація за ціновим діапазоном повертає тільки товари у заданому інтервалі. Пошук за назвою реагує на зміну поля q у URL-параметрах. Вигляд сторінки каталогу показано на рисунку 2.4.

Перевірено коректність пагінації: при переході на наступну сторінку параметр page в URL змінюється, завантажується нова порція товарів. Empty state (порожній стан) коректно відображається при комбінації фільтрів, що не дає результатів. Сценарій 3. Сторінка товару та додавання до кошика.

При переході на сторінку конкретного товару (/catalog/:id) відображаються всі поля: зображення, назва, категорія, ціна, опис та таблиця характеристик. Accordion-секція «Як отримати ключ?» коректно розгортається та згортається при натисканні. Кнопка «Додати до кошика» додає товар до CartContext та відображає toast-сповіщення «Товар додано до кошика». Лічильник у Header-іконці кошика збільшується. Вигляд сторінки товару показано на рисунку 2.5.

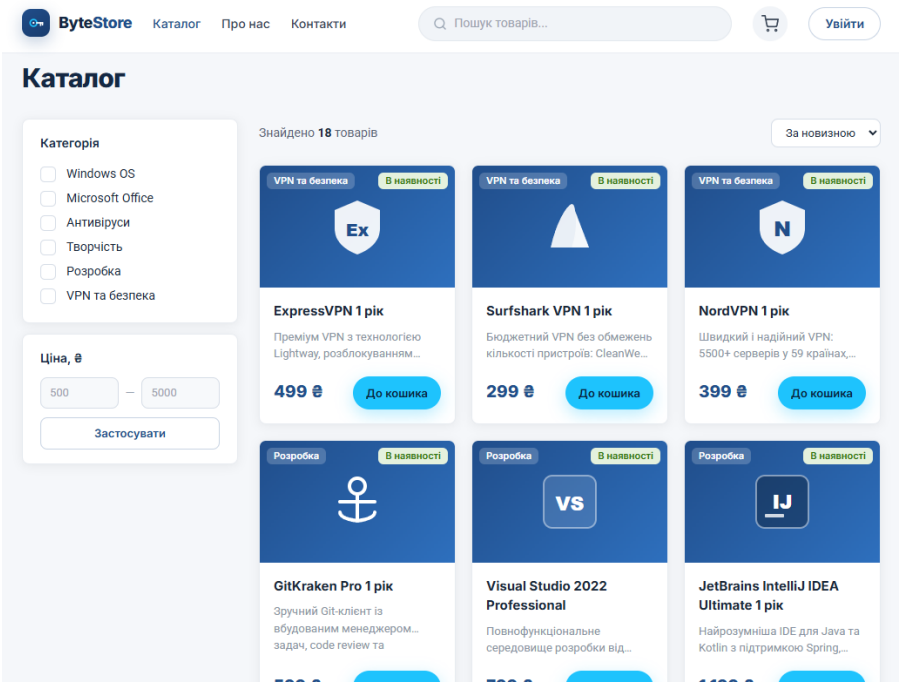


Рисунок 2.4 – Сторінка каталогу з sidebar-фільтрами

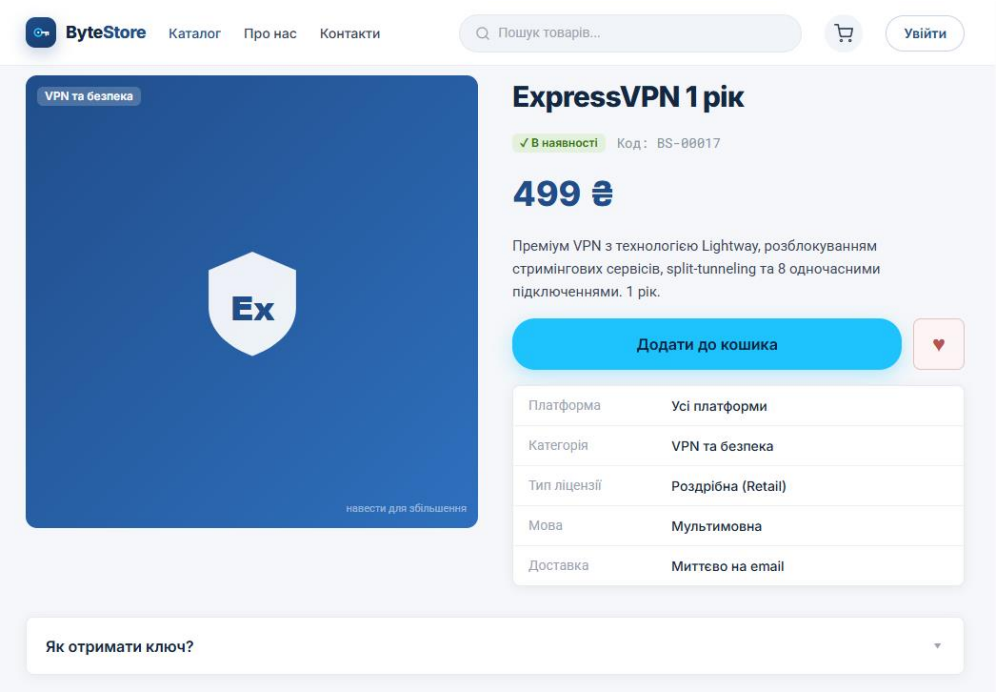


Рисунок 2.5 – Сторінка опису товару вебсайту ByteStore

Сценарій 4. Кошик та оформлення замовлення.

На сторінці кошика (/cart) перевірено роботу qty-степера: кнопки «+» та «–» коректно змінюють кількість, загальна сума перераховується миттєво. Кнопка видалення товару з кошика прибирає позицію та оновлює підсумок. При спробі перейти до оформлення замовлення без авторизації відбувається перенаправлення на /login зі збереженням поточного шляху в state. Вигляд сторінки кошика показано на рисунку 2.6.

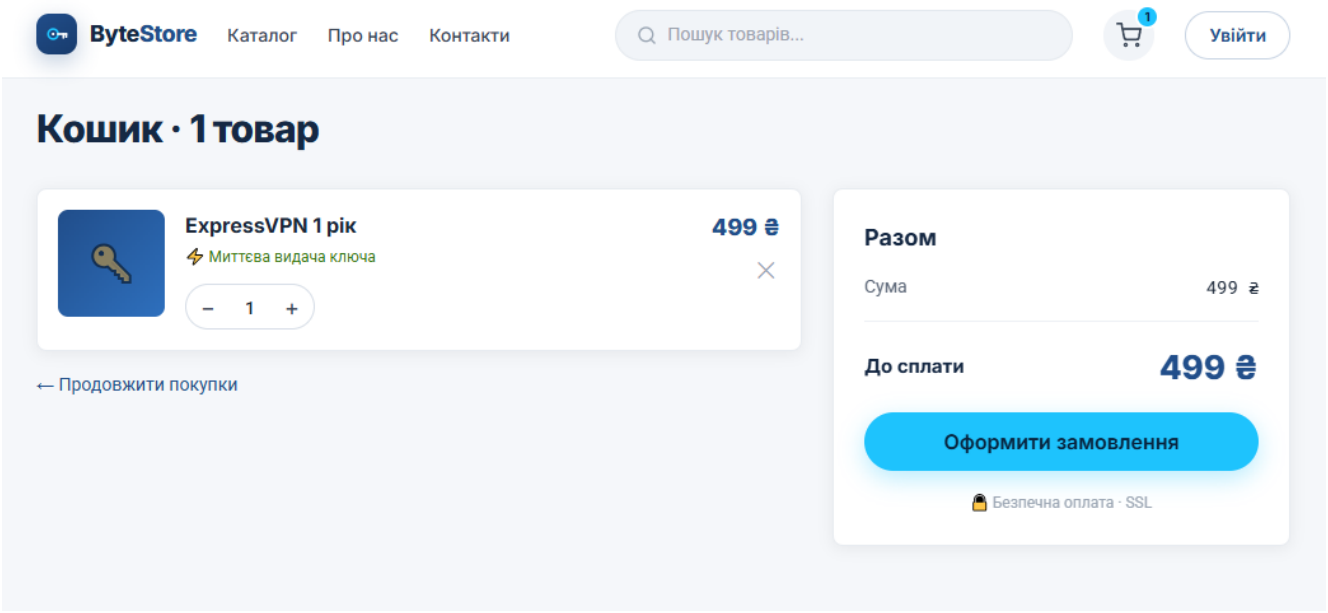


Рисунок 2.6 – Сторінка кошика вебсайту ByteStore

Після авторизації та переходу на /checkout перевірено роботу триетапного процесу. На кроці 1 поля контактної інформації заповнені попередньо (з профілю користувача). Кнопка «Далі» стає активною лише при заповнених обов'язкових полях. На кроці 2 вибір способу оплати коректно підсвічує активну картку. На кроці 3 відображається підсумок замовлення. Після натискання «Підтвердити замовлення» виконується POST /api/orders, кошик очищується і відбувається перенаправлення на /order-success/:id. Вигляд сторінки оформлення замовлення показано на рисунку 2.7.

Сценарій 5. Особистий кабінет та перегляд ліцензійних ключів.

Сторінка /account відображає sidebar-навігацію з ім'ям користувача. Перехід до підрозділів профілю та замовлень здійснюється через Outlet без перезавантаження. На сторінці /account/orders таблиця замовлень відображається з кольоровими статусними бейджами. Вигляд сторінки замовлень показано на рисунку 2.8.

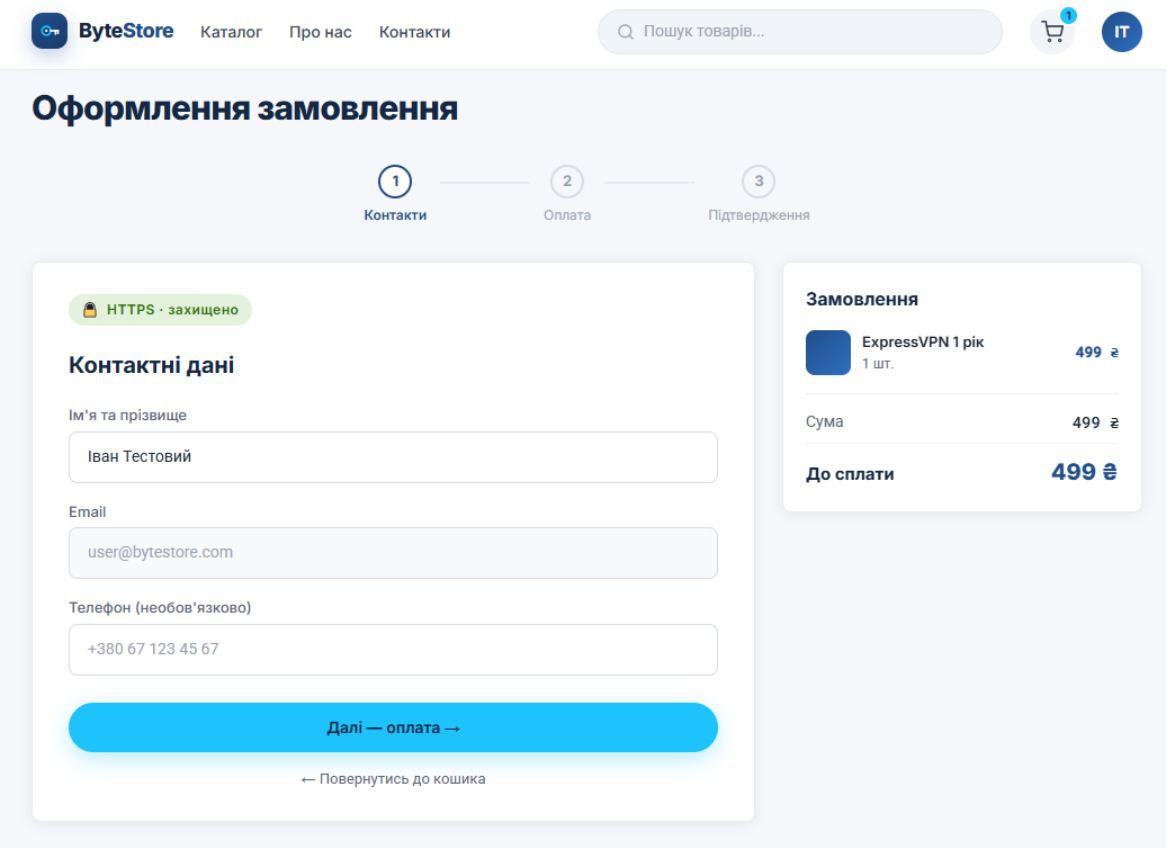


Рисунок 2.7 – Триетапний процес оформлення замовлення

На сторінці /account/orders/:id деталі замовлення відображають позиції та ліцензійні ключі. Ключі за замовчуванням приховані (blur-ефект CSS filter: blur(6px)). При натисканні кнопки «Показати ключ» blur знімається та з'являється кнопка «Копіювати». Функція копіювання використовує navigator.clipboard.writeText() та відображає toast «Ключ скопійовано». Вигляд сторінки деталей замовлення показано на рисунку 2.9.

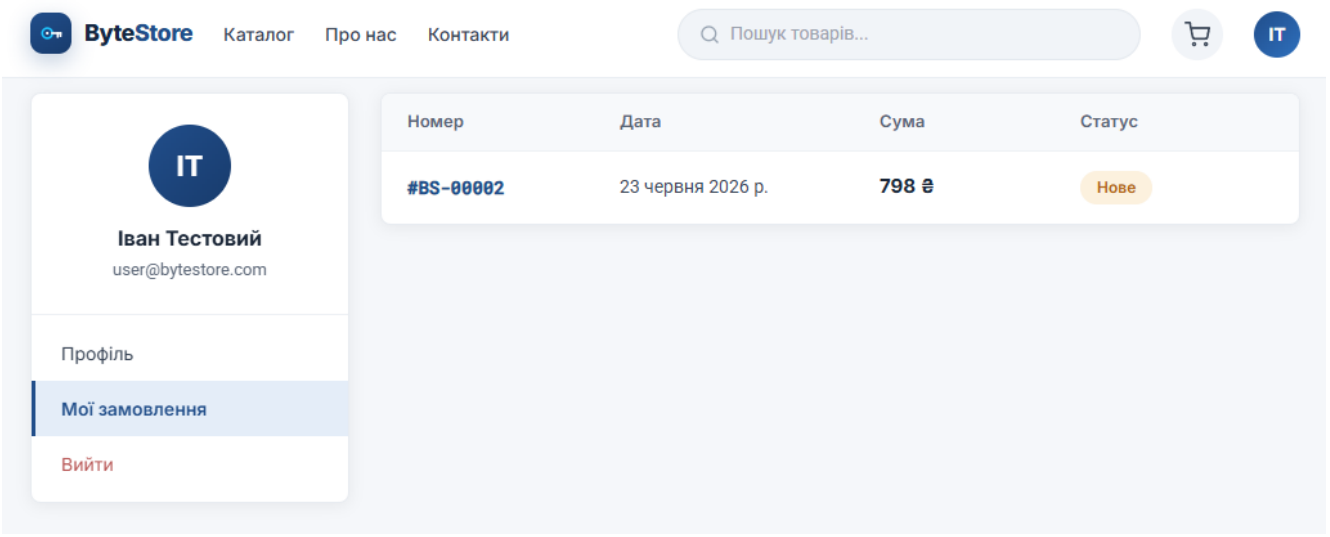


Рисунок 2.8 – Список замовлень в особистому кабінеті

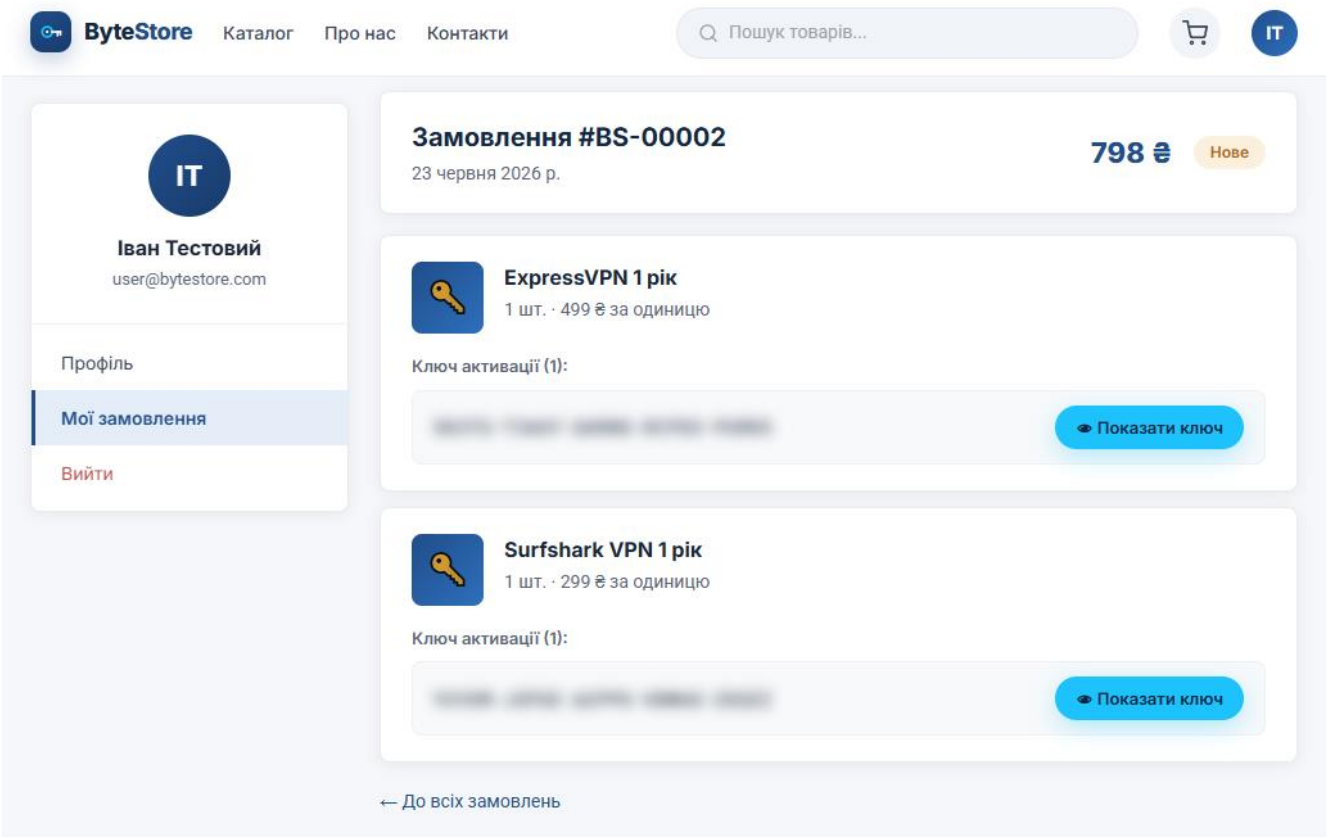


Рисунок 2.9 – Сторінка деталей замовлення з blur/reveal для ключів

Сценарій 6. Адміністративна панель.

Перевірено доступ до /admin під обліковим записом з роллю ADMIN — сторінка відкривається коректно. При спробі відкрити /admin під роллю USER відбувається перенаправлення на /403 (див.рис 2.10). Спроба відкрити /admin без

авторизації перенаправляє на /login.

На AdminDashboard (/admin) відображаються KPI-картки та таблиця товарів. Перевірено операції CRUD на сторінці /admin/products: додавання нового товару через модальну форму, редагування існуючого, видалення з підтверджуючим діалогом. Зміни миттєво відображаються у таблиці завдяки інвалідації кешу React Query після мутацій. Вигляд адміністративної панелі показано на рисунку 2.11.

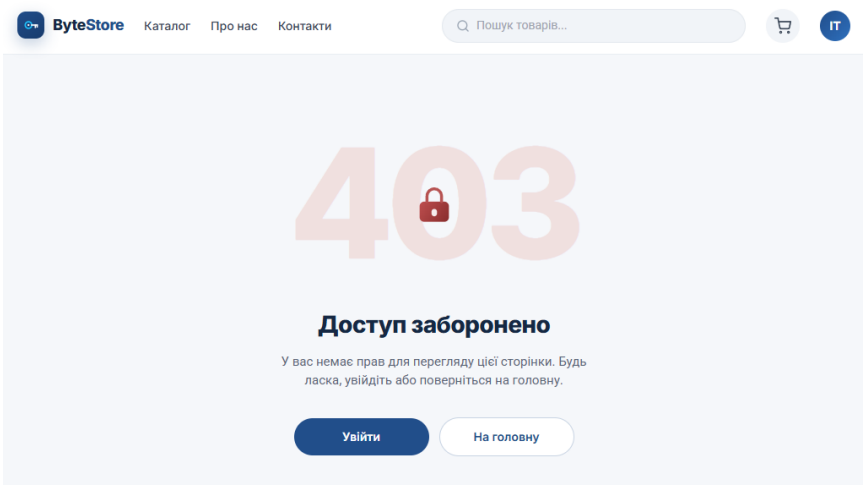


Рисунок 2.10 – Сторінка 403 — Доступ заборонено

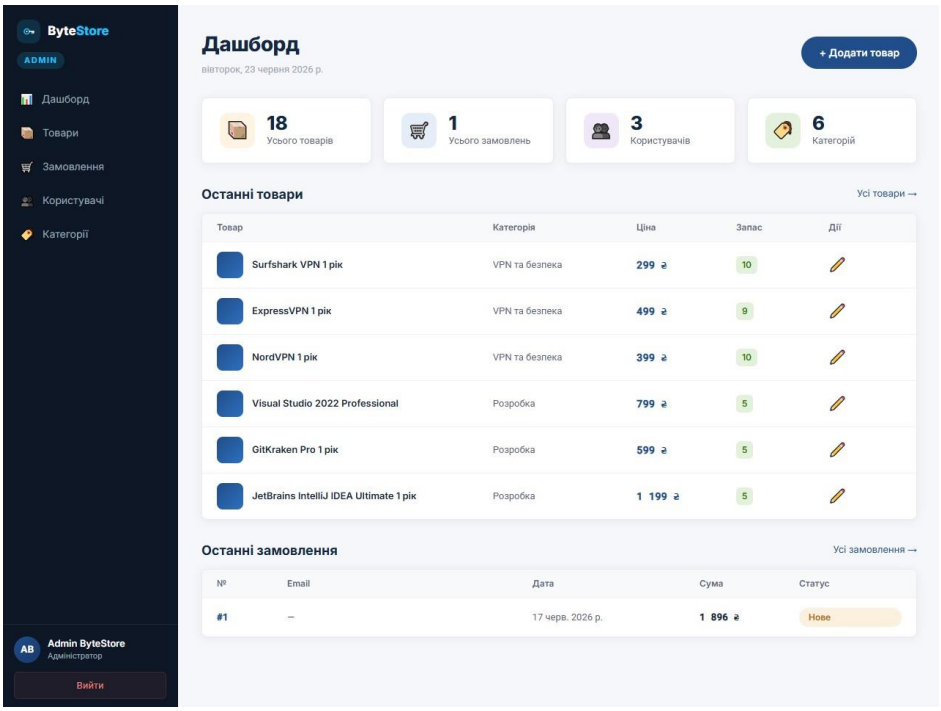


Рисунок 2.11 – Головна сторінка адміністративної панелі ByteStore

На сторінці /admin/orders перевірено інлайн-зміну статусу замовлення через dropdown. Після зміни статусу рядок таблиці оновлює бейдж без перезавантаження сторінки. Вигляд сторінки управління замовленнями показано на рисунку 2.12.

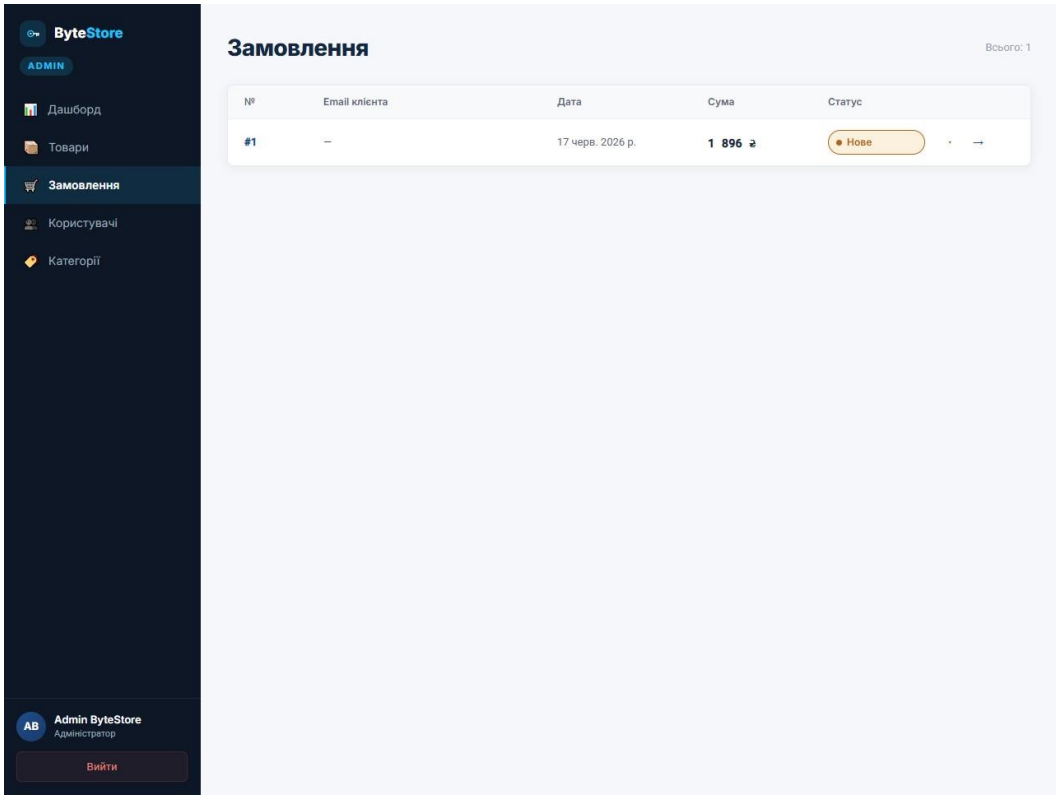


Рисунок 2.12 – Сторінка управління замовленнями в адмін-панелі

Сценарій 7. Адаптивність та системні сторінки.

Перевірено відображення сайту при ширині вікна 375px (мобільний пристрій): Header приховує навігаційні посилання, замість них відображається BottomNav з чотирма іконками (каталог, пошук, кошик, профіль). Сітка каталогу переходить в одноколонний вигляд. Sidebar фільтрів прихований та відкривається кнопкою-перемикачем. Форми реєстрації та входу зберігають читабельність. При ширині 768px (планшет) сітка товарів переходить у двоколонний вигляд. Головна сторінка ByteStore показана на рисунку 2.13.

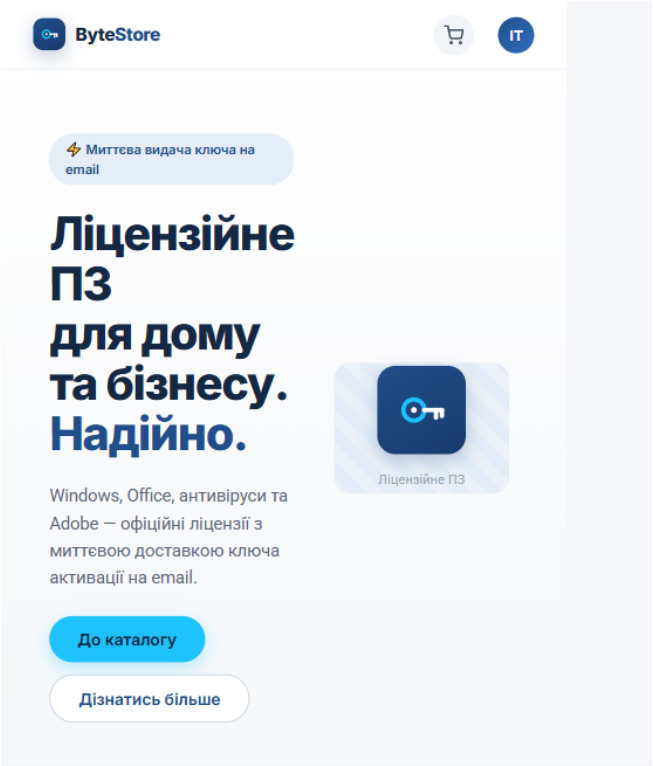


Рисунок 2.13 – Головна сторінка вебсайту ByteStore

Контрольний список охоплює 35 тест-кейсів, розподілених за функціональними модулями системи.

За результатами тестування всі 35 тест-кейси мають статус «ПРОЙДЕНО». Жодних критичних дефектів, що перешкоджають основним сценаріям використання, не виявлено. Виявлені в ході розробки незначні дефекти (некоректне скидання стану фільтрів при переході між сторінками, відсутність обмеження мінімального значення qty-степера) були усунені до фінального тестування.

За результатами мануального тестування підтверджено повну працездатність клієнтської частини вебсайту ByteStore. Усі функціональні вимоги, визначені у технічному завданні реалізовані та перевірені. Маршрутизація між сторінками функціонує коректно, захист маршрутів за ролями USER та ADMIN забезпечує необхідний рівень розмежування доступу. Форми реєстрації, входу та оформлення замовлення коректно реагують як на коректні, так і на некоректні вхідні дані.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розміщення вебсайту в Інтернеті

Розміщення вебсайту магазину «ByteStore» в мережі Інтернет передбачає послідовне виконання комплексу дій, пов’язаних із підготовкою серверного середовища, налаштуванням бази даних, розгортанням серверної та клієнтської частин, а також забезпеченням стабільного доступу користувачів до вебзастосунку.

Функціональне призначення системи реалізується через сукупність взаємопов’язаних технологічних рішень. Серверна частина побудована на основі фреймворку Spring Boot з архітектурою REST API, що забезпечує стандартизовану взаємодію між клієнтом і сервером. Автентифікація та розмежування прав доступу реалізовані засобами Spring Security із застосуванням механізму JWT-токенів. Для зберігання та управління реляційними даними використовується СУБД PostgreSQL. Клієнтська частина розроблена у вигляді односторінкового застосунку (Single Page Application) на основі бібліотеки React із застосуванням Tailwind CSS та Material UI, що забезпечує формування сучасного та адаптивного інтерфейсу користувача.

На першому етапі розгортання необхідно підготувати серверне середовище, встановивши необхідне програмне забезпечення, зокрема Java Runtime Environment, PostgreSQL та Node.js. Далі здійснюється налаштування бази даних: створюється окрема база для проєкту «ByteStore», визначаються користувачі та надаються відповідні права доступу.

Другим етапом є розгортання серверної частини. Виконується збірка Spring Boot застосунку у вигляді виконуваного JAR-файлу, після чого він запускається на сервері. У конфігураційних файлах задаються параметри підключення до бази даних, порт роботи застосунку та параметри безпеки, зокрема налаштування JWT.

Третім етапом здійснюється розгортання клієнтської частини. Вебзастосунок React збирається у production-версію, після чого статичні файли розміщуються на вебсервері, такому як Nginx або Apache. За потреби налаштовується маршрутизація

для коректної роботи SPA-додатку.

Четвертим етапом виконується налаштування зворотного проксі-сервера, який забезпечує перенаправлення запитів від клієнта до серверної частини, а також підтримку HTTPS-протоколу для захищеної передачі даних. Для цього встановлюється SSL-сертифікат, що гарантує безпечне з'єднання між користувачем і системою.

Завершальним етапом є тестування розгорнутого вебзастосунку, перевірка коректності роботи API, автентифікації, взаємодії з базою даних та відображення інтерфейсу користувача. У разі виявлення помилок здійснюється їх усунення та повторне налаштування компонентів системи.

Таким чином, запропонована процедура розгортання забезпечує стабільну роботу вебсайту «ByteStore» в мережі Інтернет, його безпечність, масштабованість та доступність для кінцевих користувачів.

3.2 Інструкція з обслуговування та наповнення вебсайту

Обслуговування та наповнення вебсайту магазину «ByteStore» є важливою складовою його стабільного функціонування та забезпечення актуальності інформації для користувачів. Даний процес включає оновлення контенту, управління каталогом програмного забезпечення, контроль роботи системи, а також виконання резервного копіювання даних.

Основним інструментом для наповнення вебсайту є адміністративна панель, яка реалізована на основі ролей та прав доступу користувачів. Доступ до адміністративної частини мають лише авторизовані користувачі з відповідним рівнем прав, що забезпечується механізмом Spring Security та JWT-аутентифікацією. Через адміністративний інтерфейс здійснюється додавання, редагування та видалення товарів, категорій, описів програмного забезпечення, цін та ліцензійних ключів.

Оновлення каталогу товарів виконується шляхом внесення нових записів до бази даних PostgreSQL. Адміністратор має можливість завантажувати зображення

продуктів, додавати описи, характеристики та іншу супровідну інформацію. Усі зміни одразу відображаються на клієнтській частині вебсайту завдяки взаємодії через REST API.

Важливою частиною обслуговування є контроль працездатності системи. Регулярно здійснюється моніторинг серверної частини, перевірка логів застосунку, а також контроль навантаження на сервер і базу даних. У разі виникнення помилок або збоїв проводиться їх діагностика та усунення.

Окрему увагу приділено резервному копіюванню даних. Рекомендується періодичне створення резервних копій бази даних PostgreSQL, що дозволяє відновити інформацію у випадку технічних збоїв або втрати даних. Резервні копії можуть зберігатися на окремому сервері або у хмарному сховищі.

Також передбачено оновлення програмного забезпечення системи. За необхідності здійснюється оновлення серверної частини (Spring Boot), клієнтського застосунку (React) та бібліотек, що використовуються в проєкті, з метою підвищення продуктивності та безпеки.

Таким чином, систематичне обслуговування та своєчасне наповнення вебсайту «ByteStore» забезпечує його стабільну роботу, актуальність інформації та високий рівень зручності для користувачів.

3.3 Інструкція з популяризації та підтримки вебсайту

Популяризація та підтримка вебсайту магазину «ByteStore» є важливими складовими забезпечення його ефективного функціонування, залучення користувачів та підвищення рівня продажів ліцензійного програмного забезпечення. Дані процеси включають комплекс заходів із цифрового маркетингу, технічної підтримки, оптимізації продуктивності та постійного вдосконалення функціоналу системи.

Основним напрямом популяризації вебсайту є пошукова оптимізація (SEO), яка передбачає оптимізацію структури сторінок, метаданих, ключових слів та контенту для покращення позицій у пошукових системах. Також важливим є

формування якісного контенту, що містить актуальні описи програмного забезпечення, акційні пропозиції та інформаційні матеріали для користувачів.

Додатково для залучення цільової аудиторії можуть використовуватися інструменти інтернет-реклами, зокрема контекстна реклама, таргетовані рекламні кампанії у соціальних мережах та email-маркетинг. Це дозволяє підвищити впізнаваність бренду «ByteStore» та збільшити кількість відвідувачів вебсайту.

Важливим аспектом підтримки є забезпечення стабільної роботи вебзастосунку. Регулярно здійснюється моніторинг продуктивності серверної частини, часу відгуку API та навантаження на базу даних. У разі виявлення зниження продуктивності проводиться оптимізація запитів до бази даних, кешування даних або масштабування серверних ресурсів.

Також здійснюється постійне оновлення функціоналу системи відповідно до потреб користувачів. Це включає додавання нових можливостей, покращення інтерфейсу користувача та усунення виявлених помилок. Оновлення програмного забезпечення виконується з дотриманням принципів безперервної інтеграції та доставки (CI/CD), що мінімізує ризики збоїв у роботі системи.

Окремо важливою складовою підтримки є технічна підтримка користувачів, яка передбачає оперативне реагування на звернення, вирішення проблем з авторизацією, оформленням замовлень та використанням функціоналу сайту.

Таким чином, комплексна популяризація та підтримка вебсайту «ByteStore» забезпечує стабільне зростання відвідуваності, підвищення рівня користувацького досвіду та довгострокову ефективність функціонування системи.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини даного кваліфікаційної роботи є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки вебсайту магазину «ByteStore», прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки. Також буде проведено оцінювання очікуваних економічних результатів від використання розробленого програмного продукту та визначено термін його окупності. Отримані результати дадуть змогу оцінити доцільність інвестування ресурсів у подальший розвиток і підтримку вебсайту.

Об’єктом розробки є вебсайт магазину «ByteStore»

Розрахунок вартості розробки виконується в декілька етапів:

- описати технологічний процес розробки із зазначенням трудомісткості кожної операції;
- визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;
- обчислити витрати на електроенергію;
- нарахувати суму амортизаційних відрахувань;
- визначити суму накладних витрат;
- скласти кошторис та визначити собівартість робіт;
- розрахувати ціну робіт;
- визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

В цьому підрозділі розглянемо основні етапи технологічного процесу для розробки вебсайту магазину «ByteStore». Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю.

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
	Планування та аналіз	Керівник проєкту	
		Веброзробник	
	Розробка технічного завдання	Веброзробник	
	Проєктування дизайну інтерфейсу	Веброзробник	
	Розробка функціоналу вебсайту	Веброзробник	
	Тестування та налагодження	Тестувальник	
	Документування	Інженер	
	Розгортання та підтримка	Веброзробник	
Разом			

Сумарний час виконання операцій технологічного процесу становить 146 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки вебсайту магазину «ByteStore».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та

діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с * К_г \quad (4.1)$$

де: T_c – тарифна ставка, грн. (приймаємо для керівника проєкту – 450 грн./год, веброзробника – 272 грн./год., інженера – 113 грн./год., тестувальника – 100 грн./год.); $К_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Керівника проекту $З_{осн}$= 450 * 8 = 3 600 грн.

Веброзробника $З_{осн}$= 272 * 115 = 31 280 грн.

Інженера $З_{осн}$= 113 * 5 = 565 грн.

Тестувальника $З_{осн}$= 100 * 18 = 1 800 грн.

Сумарна основна заробітна плата становить

$$З_{осн} = 3\,600 + 31\,280 + 565 + 1\,800 = 37\,245 \text{ грн.}$$

Додаткова заробітна плата становить 10 – 15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.} \quad (4.2)$$

де: $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

Керівника проекту $З_{дод}$= 3 600 * 0,1 = 360 грн.

Веброзробник $З_{дод}$= 31 280 * 0,1 = 3 128 грн.

Інженера $З_{дод}$= 565 * 0,1 = 56,5 грн.

Тестувальник $З_{дод}$= 1 800 * 0,1 = 180 грн.

Загальна додаткова заробітна плата становить:

$$З_{дод} = 360 + 3\,128 + 56{,}5 + 180 = 3\,724{,}5 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 37\,245 + 3\,724{,}5 = 40\,969{,}5 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$В_{єсв} = Воп * 0{,}22 \quad (4.4)$$

$В_{єсв} = 40\ 969,5 * 0,22 = 9\ 013,29$ грн.

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. 6 = 3+4+5
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
		1	2	3	4	5	6
	Кер. проєкту	450	8	3 600	360		
	Веброзробник	272	115	31 280	3 128		
	Інженер	113	5	565	56,5		
	Тестувальник	100	18	1 800	180		
Разом				37 245	3 724,5	9013,29	49 982,79

Отже, загальні витрати на оплату праці становлять 49 982,79 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_в = W * T * S \quad (4.5)$$

де: W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії (приймаємо 15, 94 грн).

В нашій системі є 1 ПК. Витрати на електроенергію для цього комп’ютера обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно табл. 4.1) і споживані потужності наступні: комп’ютер – 0,82 кВт/год.

$$З_{ек}= 0,82 * 146 * 15,94= 1\ 908,34 \text{ грн.}$$

Витрати на електроенергію становлять 1 908,34 грн.

4.4 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення

Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

$$\text{А} = \frac{\text{Б}_\text{В} * \text{Н}_\text{А}}{100\%} T, \qquad (4.6)$$

А – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_А$ – норма амортизації, 0,04 %.

Оскільки для написання програми та її тестування використовується один ПК, вартістю 35000,00 грн., то сума амортизаційних відрахувань становитиме:

$$\text{А} = \frac{35\ \ 000,00 * 0,04}{150} * 146 = 1\ 362,7 \text{ грн.}$$

4.5 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого

суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$Н_в = В_{о.п.} * 0,2 ... 0,6$ (4.7)

де: $Н_В$ – накладні витрати.

$Н_В$= 40 969,5 *0,4= 16 387,8 грн.

4.6 Складання кошторису витрат та визначення собівартості НДР

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.4.

Таблиця 4.4 - Кошторис витрат вебсайту магазину книг «BookFlow»
Собівартість ($С_В$) НДР розрахуємо за формулою:

	Зміст витрат	Сума, грн.	В % до загальної суми
	Витрати на оплату праці	49 982,79	71,8
	Витрати на електроенергію	1 908,34	2,7
	Амортизаційні відрахування	1 362,7	2
	Накладні витрати	16 387,8	23,5
	Собівартість	69 641,63	100

Св = Во.п + Вс.з +Зе + А + Нв (4.8)

Отже, собівартість дорівнює $С_В$= 69 641,63 грн.

4.7 Розрахунок ціни НДР

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = Св * (1+Р_{рен}) * (1+ПДВ), \quad (4.9)$$

де: Св – собівартість; $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

Ц = 69 641,63*(1+0,3)*(1+0,2)= 108 640,94 грн.

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності (Ток)

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.11)$$

де: Св – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проекту; і – величина дисконтної ставки (10…15%).

$$ЧТВ = -69\,641{,}63 + \frac{38\,999{,}31}{(1+0{,}1)^1} + \frac{38\,999{,}31}{(1+0{,}1)^2} + \frac{38\,999{,}31}{(1+0{,}1)^3}$$

$$= \ 27\,343{,}88 \ \text{грн}$$

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad (4.12)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{1\,956{,}88}{38\,999{,}31} \approx 2{,}1 \ \text{р.}$$

Всі дані внесемо в зведену таблицю 4.5.

Таблиця 4.5 – Техніко-економічні показники вебсайту магазину «ByteStore»

№ п/п	Показник	Значення
1.	Собівартість, грн.	69 641,63
2.	Плановий прибуток або грошовий потік, грн.	38 999,31
3.	Ціна, грн.	108 640,94
4.	Чиста теперішня вартість, грн.	27 343,88
5.	Термін окупності, рік	2,1

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2,1 року. Отже, на основі отриманих показників можна зробити висновок, що розробка вебсайту магазину «ByteStore» є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Організація охорони праці на підприємстві

Система управління охороною праці (СУОП) на підприємстві – це комплекс взаємопов'язаних заходів, політик та процедур, метою яких є забезпечення безпеки, збереження здоров'я та працездатності співробітників у процесі трудової діяльності. Правовою основою функціонування СУОП є Закон України «Про охорону праці», Кодекс законів про працю України (КЗпП), а також міжнародні стандарти, зокрема ДСТУ ISO 45001:2019.

Для інноваційних підприємств, які поєднують написання програмного коду зі створенням апаратних рішень (прототипування фізичних пристроїв, робота з датчиками, низьковольтними генераторами чи мікроконтролерами), організація охорони праці вимагає багаторівневого підходу, що враховує як ергономічні, так і технічні ризики.

Основні складові організації охорони праці на підприємстві:

– формування нормативно-правової бази підприємства. Розробка локальних актів, що регламентують безпеку на робочих місцях. Це включає положення про службу охорони праці, інструкції з безпечного виконання робіт для кожної посади (наприклад, «Інструкція з охорони праці для інженера-розробника апаратного забезпечення») та журнали реєстрації інструктажів;

– оцінка та управління професійними ризиками. Сучасний підхід вимагає превентивного виявлення небезпек. На етапі створення фізичних прототипів ризики можуть включати вдихання випарів флюсу під час паяння мікросхем, можливість мікротравм при роботі з дрібними деталями або ураження струмом при тестуванні електричних ланцюгів. Управління ризиками передбачає встановлення витяжних систем, забезпечення місцевого освітлення та використання антистатичних килимків;

– система навчання та інструктажів. Жоден працівник не допускається до роботи без проходження відповідного навчання. Система інструктажів з охорони

праці передбачає проведення вступного інструктажу спеціалістом з охорони праці для всіх новоприйнятих працівників незалежно від їхньої освіти та стажу роботи. Перед початком виконання службових обов’язків на робочому місці безпосередній керівник проводить первинний інструктаж, під час якого працівників ознайомлюють зі специфікою використання обладнання, зокрема персональних комп’ютерів, паяльних станцій та вимірювальних приладів. Для закріплення знань і підтримання належного рівня обізнаності з питань безпеки праці не рідше одного разу на півріччя проводиться повторний інструктаж. У разі зміни технологічного процесу, модернізації обладнання або виявлення порушень вимог охорони праці організовується позаплановий інструктаж;

– атестація робочих місць та медичний контроль. Підприємство зобов'язане проводити лабораторні дослідження умов праці (заміри рівня освітленості, електромагнітного випромінювання, мікроклімату). Працівники, чия діяльність пов'язана із зоровим напруженням або дрібною моторикою, повинні проходити регулярні медичні огляди для профілактики професійних захворювань (наприклад, тунельного синдрому або погіршення зору).

5.2 Вимоги до ручного інструменту для роботи з електричною проводкою

Під час конструювання апаратних продуктів, інтеграції сенсорів чи створення енергогенеруючих модулів спеціалісти постійно взаємодіють з електричною проводкою. Навіть при розробці систем з низькою напругою та малими струмами існує сувора необхідність дотримання правил безпечної експлуатації електроустановок (ПБЕЕС) та використання спеціалізованого електромонтажного інструменту.

Ручний інструмент (пасатижі, кусачки, бокорізи, викрутки, знімачі ізоляції) повинен відповідати державним стандартам якості та гарантувати захист працівника від ураження електричним струмом, а також запобігати короткому замиканню між компонентами.

Загальні вимоги до ізоляції та конструкції:

– діелектричні властивості. Інструмент повинен мати ізолювальне покриття рукояток, виконане з некрихкого, вологостійкого та маслобензостійкого матеріалу (найчастіше це полікарбонат або спеціальні види гуми). Покриття має щільно прилягати до металевої основи і не зніматися без застосування значного фізичного зусилля;

– маркування. На кожному інструменті має бути рельєфне або незмивне маркування, яке підтверджує його придатність для роботи під напругою (стандартне маркування — «1000 V», символ подвійного трикутника та рік випробування);

– ергономіка та обмежувальні бортики. Для запобігання зісковзуванню руки на неізольовану металеву частину під час роботи (наприклад, при зачищенні жорсткого кабелю), рукоятки кусачок та плоскогубців повинні закінчуватися захисними упорами (бортиками) висотою не менше 10 мм, а викруток — не менше 5 мм.

Специфічні вимоги до окремих видів інструменту (при роботі з тонкою електронікою):

– інструмент для зняття ізоляції (стрипери). Леза повинні бути гостро заточені та мати механізм регулювання глибини різу. Це критично важливо при роботі з тонкими проводами (наприклад, для п'єзоелементів або мікроконтролерів), оскільки пошкодження струмопровідної жили призводить до збільшення опору та перегріву на ділянці з'єднання;

– викрутки. Лезо викрутки має бути ізольоване по всій довжині, залишаючи відкритою лише саму робочу частину (жало) довжиною не більше 10-15 мм. Це унеможливлює випадкове торкання сусідніх контактів на щільно скомпонованій платі;

– крімпери (обтискні кліщі). Матриці інструменту не повинні мати люфтів, щоб забезпечити надійний механічний і електричний контакт при обтисканні клем, запобігаючи іскрінню в процесі експлуатації пристрою.

Правила експлуатації, зберігання та контролю:

– щоденний візуальний контроль. Перед початком кожної зміни працівник зобов'язаний оглянути інструмент. Наявність тріщин, відшарування ізоляції, раковин або задирок на металевих частинах є абсолютною підставою для вилучення інструменту з обігу;

– періодичні лабораторні випробування. Електромонтажний ручний інструмент підлягає обов'язковим періодичним випробуванням підвищеною напругою у спеціалізованих лабораторіях. Терміни випробувань регламентуються нормативними документами (як правило, 1 раз на 12 місяців);

– умови зберігання. Інструмент слід зберігати в спеціальних органайзерах або ложементах, що виключають тертя ріжучих кромок та пошкодження ізоляції гострими предметами. Забороняється залишати інструмент поблизу джерел теплового випромінювання, що може призвести до деградації діелектричного пластику.

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебсайт магазину «ByteStore»., призначений для автоматизації процесу онлайн-продажу ліцензійного програмного забезпечення. У межах роботи виконано аналіз предметної області, сформульовано технічне завдання, визначено функціональні та нефункціональні вимоги, спроєктовано структуру web-застосунку, описано базу даних, реалізовані модулі, інтерфейс користувача, адміністративну панель, порядок інсталяції, тестування та експлуатації.

У першому розділі проаналізовано існуючі підходи до створення інтернет-магазинів і визначено доцільність розробки окремого web-застосунку. Сформульовано призначення програмного продукту, перелік користувачів, основні функції, вимоги до даних, програмної документації, етапів розробки та контролю якості.

У другому розділі обґрунтовано вибір технологій. Описано архітектуру системи, структуру сторінок, базу даних, серверні дії, каталог товарів, кошик, оформлення замовлення, відгуки та адміністративну панель.

У третьому розділі наведено інструкцію з інсталяції, підготовки середовища, запуску застосунку, використання тестових наборів та експлуатації програмного комплексу. Описані сценарії перевірки дозволяють контролювати працездатність головної сторінки, каталогу, фільтрів, сторінки товару, кошика, авторизації, оформлення замовлення та адміністративних функцій.

В економічному розділі підготовлено структуру розрахунків для визначення витрат на розробку та економічної ефективності впровадження.

У розділі охорони праці розглянуто вимоги до безпечної організації робочого місця.

Поставлену мету досягнуто: створено програмний продукт, що відповідає предметній області і може бути використаний в системах електронної комерції.

ПЕРЕЛІК ПОСИЛАНЬ

1) Марціяш Г.Я., Слободян Р.О. Методичні вказівки до виконання кваліфікаційної роботи для здобувачів фахової передвищої освіти спеціальності 122 «Комп'ютерні науки». Тернопіль : ВСП «ТФК ТНТУ ім. І. Пулюя», 2024, 48с.

2) Git. Система контролю версій. — Режим доступу: https://git-scm.com/doc

3) Spring Boot Reference Documentation. — Режим доступу: https://docs.spring.io/spring-boot/docs/3.5.x/reference/htmlsingle/

4) React — A JavaScript library for building user interfaces. — Режим доступу: https://react.dev/

5) ECMAScript 2022 Language Specification (ECMA-262). — Режим доступу: https://tc39.es/ecma262/

6) Vite. Next Generation Frontend Tooling. — Режим доступу: https://vitejs.dev/guide/

7) Visual Studio Code. — Режим доступу: https://code.visualstudio.com/docs

8) Axios. Promise based HTTP client for the browser and Node.js. — Режим доступу: https://axios-http.com/docs/intro

9) Web Storage API (MDN). — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

10) React Router v6 Documentation. — Режим доступу: https://reactrouter.com/en/main

11) TanStack Query (React Query) Documentation. — Режим доступу: https://tanstack.com/query/latest/docs/react/overview

12) Tailwind CSS Documentation. — Режим доступу: https://tailwindcss.com/docs

13) MUI (Material UI) Documentation. — Режим доступу: https://mui.com/material-ui/getting-started/

14) Java 21 Documentation. Oracle Corporation. — Режим доступу: https://docs.oracle.com/en/java/javase/21/

15) Apache Maven Project Documentation. — Режим доступу: https://maven.apache.org/guides/

16) Spring Security Reference. — Режим доступу: https://docs.spring.io/spring-security/reference/index.html

17) JJWT — Java JWT Library (jjwt 0.12.6). — Режим доступу: https://github.com/jwtk/jjwt

18) SpringDoc OpenAPI Documentation. — Режим доступу: https://springdoc.org/

19) Spring Data JPA Reference Documentation. — Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

20) MapStruct Reference Guide (1.6). — Режим доступу: https://mapstruct.org/documentation/stable/reference/html/

21) Flyway Database Migration Tool. — Режим доступу: https://documentation.red-gate.com/flyway

ДОДАТКИ

Додаток А. Лістинг файлу «client.js»

```
import axios from 'axios';
const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL ?? 'http://localhost:8080/api',
  headers: { 'Content-Type': 'application/json' },
});

apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('bytestore_token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    const status = error.response?.status;
    // Don't redirect on auth-endpoint 401s (wrong password etc.)
    const isAuthRoute = error.config?.url?.includes('/auth/');

    if (status === 401 && !isAuthRoute) {
      localStorage.removeItem('bytestore_token');
      localStorage.removeItem('bytestore_user');
      window.location.href = '/login';
      return new Promise(() => {}); // halt chain while browser navigates
    }
    const apiError = error.response?.data?.error;
    return Promise.reject(apiError ?? error);
  },
);

export default apiClient;
```

Додаток Б. Лістинг файлу «AuthContext.jsx»

```
import { createContext, useCallback, useContext, useEffect, useMemo, useState } from 'react';
import { login as apiLogin, register as apiRegister } from '../api/auth';

const AuthContext = createContext(null);

const TOKEN_KEY = 'bytestore_token';
const USER_KEY  = 'bytestore_user';

function isTokenValid(token) {
  try {
    const { exp } = JSON.parse(atob(token.split('.')[1]));
    return exp * 1000 > Date.now();
  } catch {
    return false;
  }
}

function persistSession(data) {
  const userInfo = {
    email: data.email,
    firstName: data.firstName,
    lastName: data.lastName,
    role: data.role,
  };
  localStorage.setItem(TOKEN_KEY, data.token);
  localStorage.setItem(USER_KEY, JSON.stringify(userInfo));
  return userInfo;
}

export function AuthProvider({ children }) {
  const [token, setToken] = useState(() => localStorage.getItem(TOKEN_KEY));
```

```
  const [user, setUser] = useState(() => {
    const stored = localStorage.getItem(USER_KEY);
    return stored ? JSON.parse(stored) : null;
  });

  // Evict stale token on mount (e.g. after browser restart with an old
session)
  useEffect(() => {
    const stored = localStorage.getItem(TOKEN_KEY);
    if (stored && !isTokenValid(stored)) {
      localStorage.removeItem(TOKEN_KEY);
      localStorage.removeItem(USER_KEY);
      setToken(null);
      setUser(null);
    }
  }, []);

  const login = useCallback(async (credentials) => {
    const data = await apiLogin(credentials);
    const userInfo = persistSession(data);
    setToken(data.token);
    setUser(userInfo);
    return data;
  }, []);

  const register = useCallback(async (userData) => {
    const data = await apiRegister(userData);
    const userInfo = persistSession(data);
    setToken(data.token);
    setUser(userInfo);
    return data;
  }, []);

  const logout = useCallback(() => {
```

```
    localStorage.removeItem(TOKEN_KEY);
    localStorage.removeItem(USER_KEY);
    setToken(null);
    setUser(null);
  }, []);

  const value = useMemo(() => ({
    user,
    token,
    login,
    register,
    logout,
    isAuthenticated: !!token && !!user,
    isAdmin: user?.role === 'ADMIN',
  }), [user, token, login, register, logout]);

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
}

export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) throw new Error('useAuth must be used within AuthProvider');
  return ctx;
}
```

Додаток В. Лістинг файлу «CartContext.jsx»

```
import { createContext, useCallback, useContext, useMemo, useState } from
'react';

const CartContext = createContext(null);
const CART_KEY = 'bytestore_cart';

function loadCart() {
  try {
    const stored = localStorage.getItem(CART_KEY);
    return stored ? (JSON.parse(stored).items ?? []) : [];
  } catch {
    return [];
  }
}

function saveCart(items) {
  localStorage.setItem(CART_KEY, JSON.stringify({ items, updatedAt: new
Date().toISOString() }));
}

export function CartProvider({ children }) {
  const [items, setItems] = useState(loadCart);

  const addItem = useCallback((product) => {
    setItems((prev) => {
      const existing = prev.find((i) => i.productId === product.id);
      const next = existing
        ? prev.map((i) =>
            i.productId === product.id ? { ...i, quantity: i.quantity + 1 }
: i,
          )
        : [
```

```
          ...prev,
          {
            productId: product.id,
            name: product.name,
            price: product.price,
            imageUrl: product.imageUrl ?? null,
            quantity: 1,
          },
        ];
    saveCart(next);
    return next;
  });
}, []);

const removeItem = useCallback((productId) => {
  setItems((prev) => {
    const next = prev.filter((i) => i.productId !== productId);
    saveCart(next);
    return next;
  });
}, []);

const updateQty = useCallback((productId, quantity) => {
  setItems((prev) => {
    if (quantity <= 0) {
      const next = prev.filter((i) => i.productId !== productId);
      saveCart(next);
      return next;
    }
    const next = prev.map((i) =>
      i.productId === productId ? { ...i, quantity } : i,
    );
    saveCart(next);
    return next;
```

```
    });
  }, []);

  const clearCart = useCallback(() => {
    setItems([]);
    localStorage.removeItem(CART_KEY);
  }, []);

  const totalPrice = useMemo(
    () => items.reduce((sum, i) => sum + Number(i.price) * i.quantity, 0),
    [items],
  );

  const itemCount = useMemo(
    () => items.reduce((sum, i) => sum + i.quantity, 0),
    [items],
  );

  const value = useMemo(
    () => ({ items, addItem, removeItem, updateQty, clearCart, totalPrice,
itemCount }),
    [items,   addItem,   removeItem,   updateQty,   clearCart,   totalPrice,
itemCount],
  );

  return                                                <CartContext.Provider
value={value}>{children}</CartContext.Provider>;
}

export function useCart() {
  const ctx = useContext(CartContext);
  if (!ctx) throw new Error('useCart must be used within CartProvider');
  return ctx;
}
```

Додаток Г. Лістинг файлу «CatalogPage.jsx»

```
import { useState, useEffect } from 'react';
import { Link, useSearchParams } from 'react-router-dom';
import { useQuery, keepPreviousData } from '@tanstack/react-query';
import { getProducts, getCategories } from '../api/products';
import ProductCard from '../components/ProductCard';
import ProductCardSkeleton from '../components/ProductCardSkeleton';
import Pagination from '../components/Pagination';

function useIsMobile() {
  const [v, setV] = useState(() => typeof window !== 'undefined' &&
window.innerWidth < 768);
  useEffect(() => {
    const fn = () => setV(window.innerWidth < 768);
    window.addEventListener('resize', fn);
    return () => window.removeEventListener('resize', fn);
  }, []);
  return v;
}

const SORT_OPTIONS = [
  { value: 'createdAt,desc', label: 'За новизною' },
  { value: 'price,asc',      label: 'Від дешевих' },
  { value: 'price,desc',     label: 'Від дорогих' },
  { value: 'name,asc',       label: 'За назвою' },
];

const PAGE_SIZE = 12;

function Checkbox({ checked, onChange, label }) {
  return (
    <label className="flex items-center gap-3 cursor-pointer py-1 select-
none">
```

```
      <span
        onClick={onChange}
        role="checkbox"
        aria-checked={checked}
        style={{
          width: 18,
          height: 18,
          borderRadius: 5,
          border: checked ? 'none' : '1.5px solid #D7DEE8',
          background: checked ? '#1E4D8C' : '#FFFFFF',
          display: 'flex',
          alignItems: 'center',
          justifyContent: 'center',
          flexShrink: 0,
          cursor: 'pointer',
          transition: 'all 150ms',
        }}
      >
        {checked && (
          <span style={{ color: '#FFFFFF', fontSize: 11, lineHeight: 1,
fontWeight: 700 }}>✓</span>
        )}
      </span>
      <span className="flex-1 font-body text-sm" style={{ color: '#1B2738'
}}>{label}</span>
    </label>
  );
}

function FilterCard({ title, children }) {
  return (
    <div
      className="card"
      style={{ padding: '16px 20px', marginBottom: 12 }}
```

```
    >
      <h3
        className="font-heading"
        style={{ fontWeight: 600, fontSize: 14, color: '#142844', margin: '0
0 12px' }}
      >
        {title}
      </h3>
      {children}
    </div>
  );
}

function EmptyState({ onReset }) {
  return (
    <div className="card flex flex-col items-center gap-4 py-16" style={{
textAlign: 'center' }}>
      <div style={{ fontSize: 52 }}>🔍</div>
      <h2
        className="font-heading"
        style={{ fontWeight: 700, fontSize: 22, color: '#142844', margin: 0
}}
      >
        Нічого не знайдено
      </h2>
      <p
        className="font-body text-sm text-text-muted"
        style={{ margin: 0, maxWidth: 380 }}
      >
        Спробуйте змінити параметри пошуку або скинути активні фільтри
      </p>
      <button
        onClick={onReset}
        className="btn-secondary"
```

```
        style={{ marginTop: 4 }}
      >
        Скинути фільтри
      </button>
    </div>
  );
}

export default function CatalogPage() {
  const [searchParams, setSearchParams] = useSearchParams();

  // URL-driven state (page is 0-indexed for the API)
  const page       = Math.max(0, Number(searchParams.get('page') ?? 0));
  const q          = searchParams.get('q') ?? '';
  const      categoryId      =      searchParams.get('categoryId')      ?
Number(searchParams.get('categoryId')) : null;
  const minPrice   = searchParams.get('minPrice') ?? '';
  const maxPrice   = searchParams.get('maxPrice') ?? '';
  const sort       = searchParams.get('sort') ?? 'createdAt,desc';

  // Local price inputs — committed to URL only on submit
  const [priceFrom, setPriceFrom] = useState(minPrice);
  const [priceTo,   setPriceTo]   = useState(maxPrice);
  const [showFilters, setShowFilters] = useState(false);
  const isMobile = useIsMobile();

  useEffect(() => {
    setPriceFrom(searchParams.get('minPrice') ?? '');
    setPriceTo(searchParams.get('maxPrice') ?? '');
  }, [searchParams]);

  /* ── Queries ── */
  const { data: categories } = useQuery({
    queryKey: ['categories'],
```

```
    queryFn: getCategories,
    staleTime: Infinity,
  });

  const { data: productsData, isLoading, isFetching } = useQuery({
    queryKey: ['products', { page, q, categoryId, minPrice, maxPrice, sort
}],
    queryFn: () => {
      const params = { page, size: PAGE_SIZE, sort };
      if (q)          params.q          = q;
      if (categoryId) params.categoryId = categoryId;
      if (minPrice)   params.minPrice   = minPrice;
      if (maxPrice)   params.maxPrice   = maxPrice;
      return getProducts(params);
    },
    placeholderData: keepPreviousData,
  });

  function setParam(key, value) {
    setSearchParams((prev) => {
      const next = new URLSearchParams(prev);
      if (value !== '' && value != null) next.set(key, String(value));
      else next.delete(key);
      if (key !== 'page') next.set('page', '0');
      return next;
    });
  }

  function applyPrice() {
    setSearchParams((prev) => {
      const next = new URLSearchParams(prev);
      if (priceFrom) next.set('minPrice', priceFrom);
      else           next.delete('minPrice');
      if (priceTo)   next.set('maxPrice', priceTo);
```

```
      else           next.delete('maxPrice');
      next.set('page', '0');
      return next;
    });
  }

  function resetFilters() {
    setSearchParams({});
    setPriceFrom('');
    setPriceTo('');
  }

  /* ── Derived values ── */
  const products      = productsData?.content ?? [];
  const totalPages    = productsData?.totalPages ?? 0;
  const totalElements = productsData?.totalElements ?? 0;
  const isEmpty       = !isLoading && products.length === 0;
  const hasFilters    = !!(q || categoryId || minPrice || maxPrice);

  return (
    <div className="max-w-[1280px] mx-auto px-8 py-8">
      {/* Breadcrumb */}
      <nav className="font-body text-sm mb-5" style={{ color: '#9AA5B4' }}>
        <Link to="/" style={{ color: '#9AA5B4', textDecoration: 'none'
}}>Головна</Link>
        <span className="mx-2">/</span>
        <span style={{ color: '#1B2738' }}>Каталог</span>
      </nav>

      <h1
        className="font-heading"
        style={{ fontWeight: 800, fontSize: 30, letterSpacing: '-0.8px',
color: '#142844', marginBottom: 24 }}
      >
```

```
        Каталог
      </h1>

      {/* Mobile filter + sort bar */}
      {isMobile && (
        <div className="flex items-center gap-3 mb-4">
          <button
            onClick={() => setShowFilters((v) => !v)}
            style={{
              flex: 1,
              padding: '9px 16px',
              borderRadius: 20,
              border: hasFilters ? '1.5px solid #1E4D8C' : '1.5px solid
#D7DEE8',
              background: hasFilters ? '#E4EDF8' : '#FFFFFF',
              fontFamily: 'Inter, sans-serif', fontWeight: 600, fontSize:
13,
              color: hasFilters ? '#1E4D8C' : '#3A4658',
              cursor: 'pointer', display: 'flex', alignItems: 'center',
justifyContent: 'center', gap: 6,
            }}
          >
            ⚙ Фільтри{hasFilters ? ' ●' : ''}
          </button>
          <select
            value={sort}
            onChange={(e) => setParam('sort', e.target.value)}
            style={{
              flex: 1, padding: '9px 12px', borderRadius: 20,
              border: '1.5px solid #D7DEE8',
              fontFamily: 'Roboto, sans-serif', fontSize: 13,
              color: '#1B2738', background: '#FFFFFF', outline: 'none',
cursor: 'pointer',
            }}
```

```
          >
            {SORT_OPTIONS.map((o) => (
              <option key={o.value} value={o.value}>{o.label}</option>
            ))}
          </select>
        </div>
      )}

      <div className="grid gap-6 items-start" style={{ gridTemplateColumns:
isMobile ? '1fr' : '248px 1fr' }}>

        {/* ———————————— Sidebar ———————————— */}
        {(!isMobile || showFilters) && (
        <aside>
          <FilterCard title="Категорія">
            {(categories ?? []).map((cat) => (
              <Checkbox
                key={cat.id}
                checked={categoryId === cat.id}
                onChange={() => setParam('categoryId', categoryId === cat.id
? '' : cat.id)}
                label={cat.name}
              />
            ))}
          </FilterCard>

          <FilterCard title="Ціна, ₴">
            <div className="flex gap-2 items-center">
              <input
                type="number"
                value={priceFrom}
                onChange={(e) => setPriceFrom(e.target.value)}
                onKeyDown={(e) => e.key === 'Enter' && applyPrice()}
                placeholder="500"
```

```
            min={0}
            style={{
              flex: 1, minWidth: 0,
              padding: '7px 10px', borderRadius: 8,
              border: '1.5px solid #D7DEE8',
              fontFamily: 'Roboto, sans-serif', fontSize: 13,
              outline: 'none', background: '#F7F9FB',
            }}
          />
          <span className="font-body text-sm text-text-faint">–</span>
          <input
            type="number"
            value={priceTo}
            onChange={(e) => setPriceTo(e.target.value)}
            onKeyDown={(e) => e.key === 'Enter' && applyPrice()}
            placeholder="5000"
            min={0}
            style={{
              flex: 1, minWidth: 0,
              padding: '7px 10px', borderRadius: 8,
              border: '1.5px solid #D7DEE8',
              fontFamily: 'Roboto, sans-serif', fontSize: 13,
              outline: 'none', background: '#F7F9FB',
            }}
          />
        </div>
        <button
          onClick={applyPrice}
          style={{
            width: '100%', marginTop: 10, padding: '8px',
            borderRadius: 8, border: '1.5px solid #C8D4E4',
            background: '#FFFFFF', cursor: 'pointer',
            fontFamily: 'Inter, sans-serif', fontWeight: 600,
            fontSize: 13, color: '#1E4D8C',
```

```
              }}
            >
              Застосувати
            </button>
          </FilterCard>

          {hasFilters && (
            <button
              onClick={resetFilters}
              className="w-full font-body text-sm"
              style={{
                background: 'none', border: 'none', cursor: 'pointer',
                padding:  '8px  0',  color:  '#9AA5B4',  textDecoration:
'underline',
              }}
            >
              Скинути всі фільтри
            </button>
          )}
        </aside>
        )}

        <div className="flex flex-col gap-5">
          {/* Top bar */}
          <div className="flex items-center justify-between">
            <p  className="font-body  text-sm  text-text-muted"  style={{
margin: 0 }}>
              {isLoading ? (
                '…'
              ) : (
                <>Знайдено    <strong    style={{    color:    '#1B2738'
}}>{totalElements}</strong> товарів</>
              )}
            </p>
```

```
            <div className="flex items-center gap-3">
              {isFetching && !isLoading && (
                <span  className="font-body"  style={{  fontSize:  12,  color:
'#9AA5B4' }}>
                  Оновлення…
                </span>
              )}
              {!isMobile && (
                <select
                  value={sort}
                  onChange={(e) => setParam('sort', e.target.value)}
                  style={{
                    padding: '7px 12px', borderRadius: 8,
                    border: '1.5px solid #D7DEE8',
                    fontFamily: 'Roboto, sans-serif', fontSize: 13,
                    color: '#1B2738', background: '#FFFFFF',
                    outline: 'none', cursor: 'pointer',
                  }}
                >
                  {SORT_OPTIONS.map((o) => (
                    <option                                          key={o.value}
value={o.value}>{o.label}</option>
                  ))}
                </select>
              )}
            </div>
          </div>

          {/* Grid / empty */}
          {isEmpty ? (
            <EmptyState onReset={resetFilters} />
          ) : (
            <div className="grid grid-cols-3 gap-[18px]">
```

```
            {isLoading
              ?       Array(PAGE_SIZE).fill(0).map((_,       i)       =>
<ProductCardSkeleton key={i} />)
              : products.map((p) => <ProductCard key={p.id} product={p}
/>)
            }
          </div>
        )}

        {/* Pagination — component expects 1-indexed page */}
        {!isEmpty && (
          <div className="mt-4">
            <Pagination
              page={page + 1}
              totalPages={totalPages}
              onPageChange={(displayPage) => setParam('page', displayPage
- 1)}
            />
          </div>
        )}
      </div>
    </div>
  </div>
  );
}
```

Додаток Д. Лістинг файлу «JwtFilter.java»

```
package com.bytestore.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationT
oken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSou
rce;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final UserDetailsServiceImpl userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain     filterChain)     throws
ServletException, IOException {
```

```
        String authHeader = request.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        String token = authHeader.substring(7);
        if (!jwtUtil.isValid(token)) {
            filterChain.doFilter(request, response);
            return;
        }

        String email = jwtUtil.extractEmail(token);
        if (email != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(email);
            UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities());
            auth.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(auth);
        }

        filterChain.doFilter(request, response);
    }
}
```

Додаток Е. Лістинг файлу «OrderServiceImpl.java»

```
package com.bytestore.service.impl;

import com.bytestore.dto.PageResponse;
import com.bytestore.dto.order.*;
import com.bytestore.entity.*;
import com.bytestore.exception.*;
import com.bytestore.mapper.OrderMapper;
import com.bytestore.repository.*;
import com.bytestore.service.EmailService;
import com.bytestore.service.OrderService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.time.OffsetDateTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Slf4j
@Service
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {

    private final OrderRepository orderRepository;
    private final OrderItemRepository orderItemRepository;
    private final LicenseKeyRepository licenseKeyRepository;
```

```
    private final ProductRepository productRepository;
    private final UserRepository userRepository;
    private final OrderMapper orderMapper;
    private final EmailService emailService;

    @Override
    @Transactional
    public OrderResponse createOrder(OrderRequest request, String userEmail)
{
        log.info("Creating order for user: {}", userEmail);
        User user = userRepository.findByEmail(userEmail).orElseThrow();

        // Load and validate all products before touching any state
        List<Product> products = new ArrayList<>();
        for (OrderItemRequest itemReq : request.items()) {
            Product p = productRepository.findById(itemReq.productId())
                    .orElseThrow(() -> new
ProductNotFoundException(itemReq.productId()));
            if (p.getStock() < itemReq.quantity()) {
                throw new InsufficientStockException(p.getName(),
p.getStock(), itemReq.quantity());
            }
            products.add(p);
        }

        // Calculate total
        BigDecimal total = BigDecimal.ZERO;
        for (int i = 0; i < products.size(); i++) {
            total = total.add(
                    products.get(i).getPrice()

.multiply(BigDecimal.valueOf(request.items().get(i).quantity()))
            );
        }
```

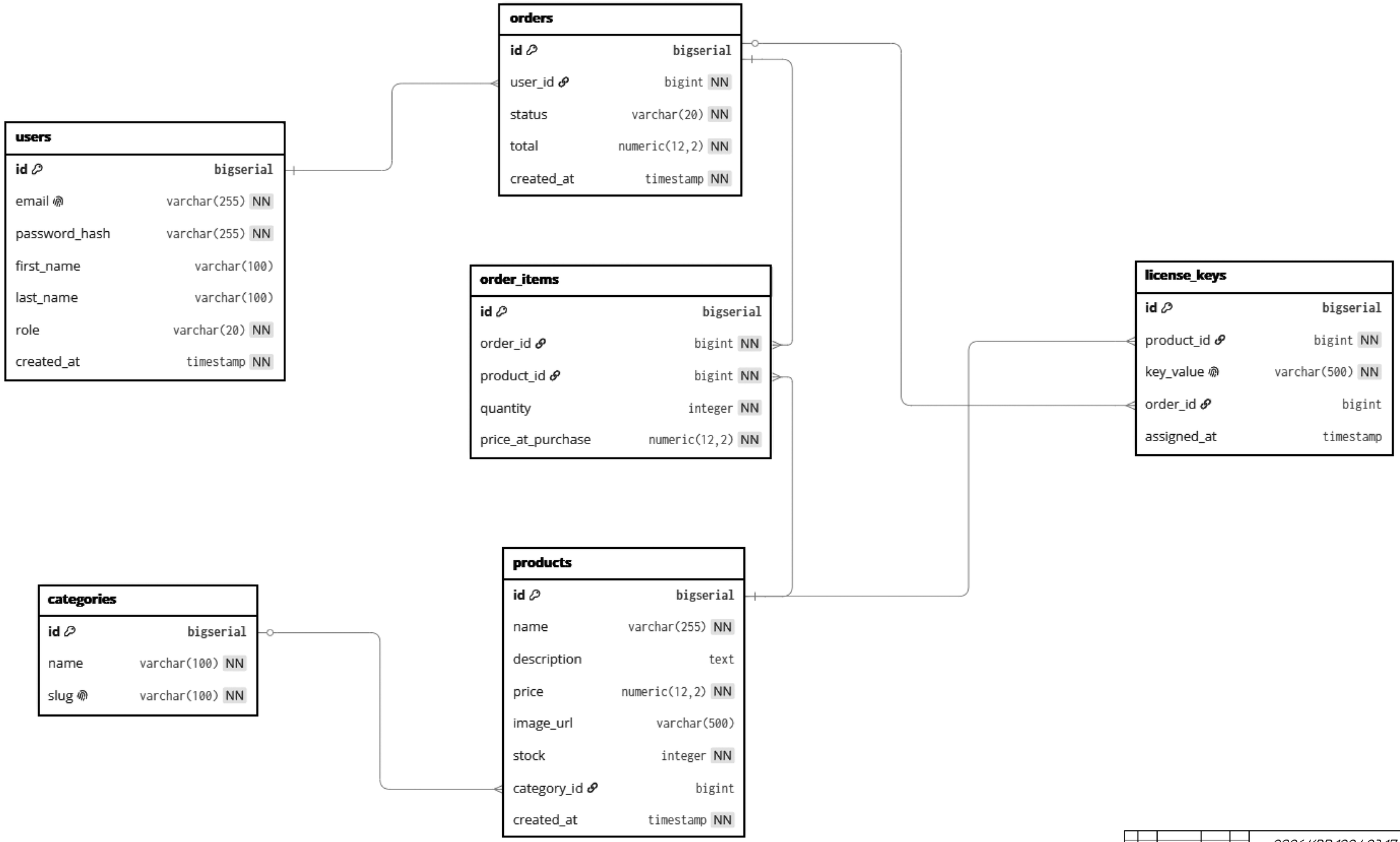
```
        Order order = orderRepository.save(

Order.builder().user(user).status(OrderStatus.NEW).total(total).build()
        );

        List<OrderItem> savedItems = new ArrayList<>();
        Map<Long, List<LicenseKey>> keysByProductId = new HashMap<>();

        for (int i = 0; i < products.size(); i++) {
            Product product = products.get(i);
            int qty = request.items().get(i).quantity();

            savedItems.add(orderItemRepository.save(
                    OrderItem.builder()
                            .order(order).product(product)

.quantity(qty).priceAtPurchase(product.getPrice())
                            .build()
            ));

            product.setStock(product.getStock() - qty);
            productRepository.save(product);

            List<LicenseKey>                    available                    =
licenseKeyRepository.findAvailableByProductId(product.getId());
            OffsetDateTime now = OffsetDateTime.now();
            List<LicenseKey>                    assigned                     =
available.stream().limit(qty).toList();
            assigned.forEach(k -> { k.setOrder(order); k.setAssignedAt(now);
});
            if                                          (!assigned.isEmpty())
licenseKeyRepository.saveAll(assigned);
            keysByProductId.put(product.getId(), assigned);
```

```
        }

        OrderResponse response = buildOrderResponse(order, savedItems,
keysByProductId);
        emailService.sendOrderConfirmation(user, response);
        return response;
    }

    @Override
    @Transactional(readOnly = true)
    public PageResponse<OrderSummaryResponse> getMyOrders(String userEmail,
Pageable pageable) {
        User user = userRepository.findByEmail(userEmail).orElseThrow();
        return PageResponse.from(
                orderRepository.findByUser(user,
pageable).map(orderMapper::toSummary)
        );
    }

    @Override
    @Transactional(readOnly = true)
    public OrderResponse getMyOrderDetail(Long orderId, String userEmail) {
        User user = userRepository.findByEmail(userEmail).orElseThrow();
        Order order = orderRepository.findById(orderId)
                .orElseThrow(() -> new OrderNotFoundException(orderId));
        if (!order.getUser().getId().equals(user.getId())) {
            throw new OrderAccessDeniedException();
        }
        return buildOrderDetailResponse(order);
    }

    @Override
    @Transactional(readOnly = true)
    public     PageResponse<OrderSummaryResponse>     getAllOrders(Pageable
```

```
pageable) {
        return PageResponse.from(

orderRepository.findAll(pageable).map(orderMapper::toSummary)
        );
    }

    @Override
    @Transactional(readOnly = true)
    public OrderResponse getOrderDetail(Long orderId) {
        Order order = orderRepository.findById(orderId)
                .orElseThrow(() -> new OrderNotFoundException(orderId));
        return buildOrderDetailResponse(order);
    }

    @Override
    @Transactional
    public OrderResponse updateOrderStatus(Long orderId, OrderStatus status)
{
        log.info("Updating order {} status to {}", orderId, status);
        Order order = orderRepository.findById(orderId)
                .orElseThrow(() -> new OrderNotFoundException(orderId));
        order.setStatus(status);
        orderRepository.save(order);
        return buildOrderDetailResponse(order);
    }

    private OrderResponse buildOrderResponse(Order order, List<OrderItem>
items,
                                             Map<Long,   List<LicenseKey>>
keysByProductId) {
        List<OrderItemResponse> itemResponses = items.stream().map(item ->
{
            List<LicenseKeyResponse> keys = keysByProductId
```

```
                    .getOrDefault(item.getProduct().getId(), List.of())
                    .stream()
                    .map(k     ->     new     LicenseKeyResponse(k.getId(),
k.getKeyValue(), k.getAssignedAt()))
                    .toList();
            return new OrderItemResponse(
                    item.getProduct().getId(), item.getProduct().getName(),
                    item.getQuantity(), item.getPriceAtPurchase(), keys
            );
        }).toList();
        return new OrderResponse(order.getId(), order.getStatus().name(),
                order.getTotal(), order.getCreatedAt(), itemResponses);
    }

    private OrderResponse buildOrderDetailResponse(Order order) {
        List<OrderItem>                          items                          =
orderItemRepository.findWithProductByOrderId(order.getId());
        List<LicenseKey>                          keys                          =
licenseKeyRepository.findWithProductByOrderId(order.getId());
        Map<Long, List<LicenseKey>> keysByProductId = keys.stream()
                .collect(Collectors.groupingBy(k                          ->
k.getProduct().getId()));
        return buildOrderResponse(order, items, keysByProductId);
    }
}
```

ER-діаграма бази даних



Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.4.23.17.00.00 БД			
Розроб.		Ратиш А.П.			Розробка вебсайту магазину «ByteStore» ER-діаграма бази даних	Літ.	Маса	Масштаб
Перевір.		Маршняк Г.Я.						
Т.контр.						Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-423 м. Тернопіль		
Н.контр.		Прийомак В.А.						
Затв.								

Копіював Формат А1

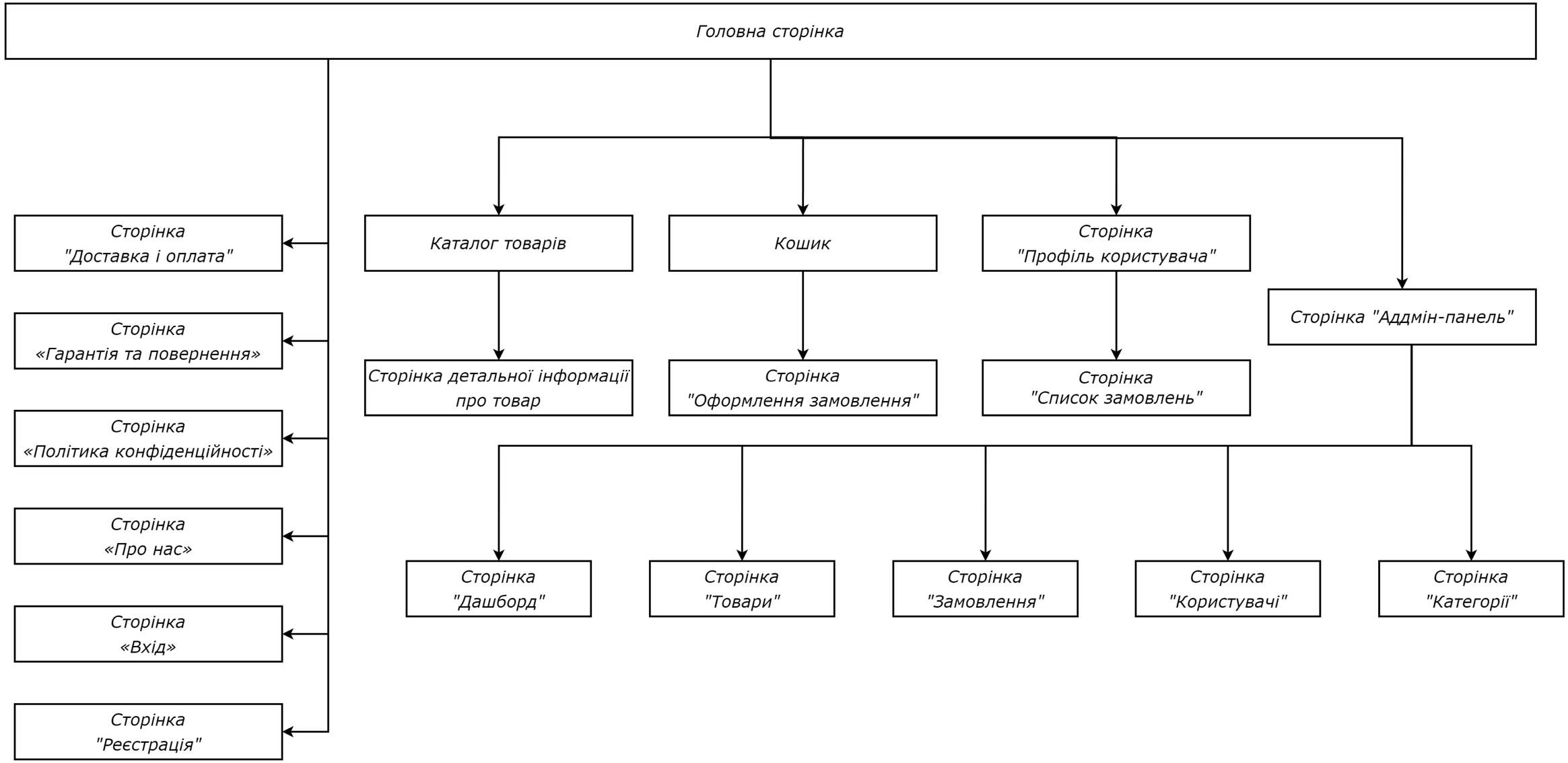
UML–діаграма варіантів використання

Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.4.23.17.00.00 ДВ		
Розроб.		Ратуш А.П.			Розробка вебсайту магазину «ByteStore» UML–діаграма варіантів використання	Літ.	Маса	Масштаб
Перевір.		Марцияш Г.Я.						
Т.контр.						Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-4.23 м. Тернопіль		
Н.контр.		Приймак В.А.						
Затв.								

Копіював

Формат А1

Схема структурна клієнтської частини



Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.4.23.17.00.00 СС		
Розроб.		Ратуш А.П.			Розробка вебсайту магазину «ByteStore»	Літ.	Маса	Масштаб
Перевір.		Марціняк Г.Я.			Схема структурна клієнтської частини			
Т.контр.						Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-423		
Н.контр.		Прийомак В.А.				м. Тернопіль		
Затв.								

Копіював Формат А1

Таблиця техніко-економічних показників

№ п/п	Показник	Одиниці вимірювання	Значення
1	Мови програмування	–	Java, JavaScript
2	Технології клієнтської частини	–	React 18.3.1, Vite 5.4.6, Material UI 5.16.7, Axios 1.7.7
3	Технології серверної частини	–	Spring Boot 3.5.14, Spring Security 6
4	База даних	–	PostgreSQL 15
5	Середовище розробки	–	IntelliJ IDEA 2026.1.3, WebStorm 2026.1.3
6	Захист	–	JWT, Spring Security
7	Собівартість	грн.	69 641
8	Плановий прибуток	грн.	38 999
9	Ціна	грн.	108 640
10	Чиста теперішня вартість	грн.	27343
11	Термін окупності	рік	2,1