Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Проєктування та розроблення веборієнтованої системи підбору комп’ютерних комплектуючих

Виконав: студент IV курсу, групи СН-41
спеціальності 122 комп’ютерні науки
(шифр і назва спеціальності)

_____ Кандибал Д.С.
(підпис) (прізвище та ініціали)

_____ Луцик Н.П.
(підпис) (прізвище та ініціали)

Керівник _____ Станько А.А.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Осухівська Г.М.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Луцику Назарію Петровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та розроблення веборієнтованої системи підбору комп'ютерних комплектуючих

Керівник роботи Станько Андрій Андрійович, доктор філософії, старший викладач КТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*

2. Термін подання студентом завершеної роботи 22 червня 2026 р.

3. Вихідні дані до роботи Літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ. 1.1 Аналіз ринку комп'ютерних комплектуючих. 1.2 Огляд існуючих систем підбору комплектуючих. 1.3 Формулювання вимог до веборієнтованої системи. 1.4 Постановка задачі проєктування системи. 1.5 Висновок до першого розділу. РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ. 2.1 Архітектура системи. 2.2 Проєктування бази даних. 2.3 Моделювання функціоналу системи. 2.4 Розробка інтерфейсу користувача. 2.5 Висновок до другого розділу. РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ. 3.1 Вибір технологій та інструментів розробки. 3.2 Реалізація серверної частини. 3.3 Реалізація 3.4 Інтеграція компонентів системи. 3.5 Методи тестування системи. 3.6 Проведення тестування. 3.7 Аналіз результатів роботи системи. 3.8 Оцінка ефективності та продуктивності. 3.9 Висновок до третього розділу
4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ ПІДБОРУ КОМП'ЮТЕРНИХ КОМПЛЕКТУЮЧИХ. 2.Актуальність теми. 3. Мета та завдання роботи. 4. Аналіз існуючих систем. 5. Архітектура системи. 6. Проєктування бази даних. 7. Реалізація системи. 8. Реалізація системи(2). 9. Тестування та результати. 10. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я. к.т.н., доц. кафедри МТ		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	*Виконано*
3.	Виконання дослідження щодо сучасних	17.02.2026-10.05.2026	*Виконано*
	веборієнтованих систем підбору комп'ютерних		
	комплектуючих		
	Розроблення архітектури, структури бази даних та		
	функціональної моделі системи		
4.	Оформлення розділу «Аналіз предметної області та	10.05.2026-17.05.2026	*Виконано*
	постановка задачі»		
5.	Оформлення розділу «Проєктування веборієнтованої	17.05.2026-21.05.2026	*Виконано*
	системи»		
6.	Оформлення розділу «Розроблення та реалізація	21.05.2026-25.05.2025	*Виконано*
	системи»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	25.05.2026-31.05.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи		

Студент _____ Луцик Н.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Станько А.А.
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)

Студенту Кандибалу Денису Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Проєктування та розроблення веборієнтованої системи підбору комп’ютерних комплектуючих

Керівник роботи Станько Андрій Андрійович, доктор філософії, старший викладач КТ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*

2. Термін подання студентом завершеної роботи 22 червня 2026 р.

3. Вихідні дані до роботи Літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ. 1.1 Аналіз ринку комп’ютерних комплектуючих. 1.2 Огляд існуючих систем підбору комлектуючих. 1.3 Формулювання вимог до веборієнтованої системи. 1.4 Постановка задачі проєктування системи. 1.5 Висновок до першого розділу. РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ. 2.1 Архітектура системи. 2.2 Проєктування бази даних. 2.3 Моделювання функціоналу системи. 2.4 Розробка інтерфейсу користувача. 2.5 Висновок до другого розділу. РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ. 3.1 Вибір технологій та інструментів розробки. 3.2 Реалізація серверної частини. 3.3 Реалізація 3.4 Інтеграція компонентів системи. 3.5 Методи тестування системи. 3.6 Проведення тестування. 3.7 Аналіз результатів роботи системи. 3.8 Оцінка ефективності та продуктивності. 3.9 Висновок до третього розділу
4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
1.ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ ПІДБОРУ КОМП’ЮТЕРНИХ КОМПЛЕКТУЮЧИХ. 2.Актуальність теми. 3. Мета та завдання роботи. 4. Аналіз існуючих систем. 5. Архітектура системи. 6. Проєктування бази даних. 7. Реалізація системи. 8. Реалізація системи(2). 9. Тестування та результати. 10. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я. к.т.н., доц. кафедри МТ		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	*Виконано*
3.	Виконання дослідження щодо сучасних	17.02.2026-10.05.2026	*Виконано*
	веборієнтованих систем підбору комп'ютерних		
	комплектуючих		
	Розроблення архітектури, структури бази даних та		
	функціональної моделі системи		
4.	Оформлення розділу «Аналіз предметної області та	10.05.2026-17.05.2026	*Виконано*
	постановка задачі»		
5.	Оформлення розділу «Проєктування веборієнтованої	17.05.2026-21.05.2026	*Виконано*
	системи»		
6.	Оформлення розділу «Розроблення та реалізація	21.05.2026-25.05.2025	*Виконано*
	системи»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	25.05.2026-31.05.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи		

Студент _____ Кандибал Д.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Станько А.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Проєктування та розроблення веборієнтованої системи підбору комп'ютерних комплектуючих // Луцик Назарій Петрович, Кандибал Денис Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2026 // С. 80, рис. 19, табл. 5, додат. 55, бібліогр. 30.

Ключові слова: веборієнтована система, комп'ютерні комплектуючі, конфігуратор ПК, React, Node.js, MySQL, електронна комерція.

Кваліфікаційна робота присвячена розробленню веборієнтованої системи підбору комп'ютерних комплектуючих, яка забезпечує автоматизацію процесу формування комп'ютерних конфігурацій, перевірку сумісності компонентів та оформлення замовлень через вебінтерфейс.

Метою роботи є проєктування та реалізація вебзастосунку для підбору комп'ютерних комплектуючих із можливістю керування товарами, формування конфігурацій персональних комп'ютерів та автоматизованої перевірки їх сумісності.

У першому розділі кваліфікаційної роботи проведено аналіз предметної області, досліджено особливості функціонування сучасних інтернет-магазинів комп'ютерної техніки та систем підбору комплектуючих, виконано аналіз аналогів і сформульовано вимоги до програмного забезпечення.

У другому розділі виконано проєктування веборієнтованої системи, розроблено архітектуру програмного забезпечення, структуру бази даних, UML-моделі та моделі взаємодії користувачів із системою.

У третьому розділі описано процес реалізації серверної та клієнтської частин застосунку із використанням технологій Node.js, Express.js, React та

MySQL. Розглянуто реалізацію каталогу товарів, конфігуратора ПК, кошика покупця, системи замовлень та адміністративної панелі.

Проведено тестування програмного забезпечення, проаналізовано результати роботи системи та оцінено її ефективність і продуктивність.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці під час розроблення програмного забезпечення.

Об’єкт дослідження – процес автоматизації підбору комп’ютерних комплектуючих та формування конфігурацій персональних комп’ютерів.

Предмет дослідження – методи, моделі та програмні засоби реалізації веборієнтованих систем електронної комерції для підбору комп’ютерних комплектуючих.

ANNOTATION

Design and Development of a Web-Based System for Computer Component Selection // Lutsyk Nazarii Petrovych, Kandybal Denys Serhiiovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, group SN-41 // Ternopil, 2026 // P. 80, fig. 19, tabl. 5, annex 55, references 30.

Keywords: web-oriented system, computer components, PC configurator, React, Node.js, MySQL, e-commerce.

The qualification work is devoted to the development of a web-oriented system for selecting computer components that automates the process of creating personal computer configurations, checking component compatibility and placing orders through a web interface.

The aim of the work is to design and implement a web application for selecting computer components with the possibility of product management, PC configuration creation and automated compatibility verification.

The first chapter analyzes the subject area, examines the features of modern computer hardware online stores and component selection systems, analyzes existing solutions and formulates software requirements.

The second chapter describes the design of the web-oriented system, including the software architecture, database structure, UML models and user interaction models.

The third chapter presents the implementation of the server-side and client-side parts of the application using Node.js, Express.js, React and MySQL technologies. The implementation of the product catalog, PC configurator, shopping cart, ordering system and administration panel is described.

The fourth chapter considers occupational safety and health issues related to software development activities.

Object of research – the process of automating the selection of computer components and creation of personal computer configurations.

Subject of research – methods, models and software tools for implementing web-oriented e-commerce systems for computer component selection.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс програмування застосунків.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

DB (Database) – база даних.

HTML (HyperText Markup Language) – мова розмітки гіпертекстових документів.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

JS (JavaScript) – мова програмування JavaScript.

JWT (JSON Web Token) – токен автентифікації користувачів.

MVC (Model-View-Controller) – архітектурний шаблон проєктування програмного забезпечення.

MySQL (My Structured Query Language) – система керування реляційними базами даних.

Node.js – серверне середовище виконання JavaScript.

ORM (Object-Relational Mapping) – технологія об’єктно-реляційного відображення.

PC (Personal Computer) – персональний комп’ютер.

REST (Representational State Transfer) – архітектурний стиль побудови вебсервісів.

SQL (Structured Query Language) – структурована мова запитів до баз даних.

UML (Unified Modeling Language) – уніфікована мова моделювання.

UI (User Interface) – інтерфейс користувача.

UX (User Experience) – користувацький досвід.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний розвиток інформаційних технологій сприяє постійному зростанню попиту на персональні комп’ютери та комплектуючі до них. Комп’ютерна техніка використовується в професійній діяльності, освіті, наукових дослідженнях, розвагах та багатьох інших сферах життя. У зв’язку з цим користувачі все частіше стикаються з необхідністю самостійного підбору комплектуючих для складання або модернізації комп’ютерних систем відповідно до власних потреб і фінансових можливостей.

Процес вибору комп’ютерних комплектуючих є досить складним через велику кількість доступних моделей, постійне оновлення асортименту та необхідність врахування сумісності між окремими компонентами. Неправильний вибір комплектуючих може призвести до несумісності обладнання, зниження продуктивності системи або перевищення запланованого бюджету. Тому актуальним завданням є створення програмних засобів, які дозволяють автоматизувати процес підбору комплектуючих та надавати користувачам рекомендації щодо формування оптимальної конфігурації персонального комп’ютера.

Веборієнтовані системи мають значні переваги порівняно з традиційними настільними додатками, оскільки забезпечують доступ до функціоналу через мережу Інтернет незалежно від місця перебування користувача та типу пристрою. Використання сучасних вебтехнологій дозволяє створювати зручні інтерактивні інтерфейси, забезпечувати швидку обробку даних та підтримувати актуальність інформації про комплектуючі.

Відповідно, актуальність теми дипломної роботи полягає в необхідності розроблення сучасної веборієнтованої системи, яка допомагатиме користувачам підбирати комп’ютерні комплектуючі з урахуванням їх сумісності, технічних характеристик та вартості. Така система сприятиме спрощенню процесу вибору обладнання, зменшенню ризику помилок та підвищенню ефективності прийняття рішень під час складання комп’ютерних систем.

Об’єктом дослідження є процес автоматизованого підбору комп’ютерних комплектуючих у вебсередовищі.

Предметом дослідження є методи, моделі та програмні засоби проєктування і розроблення веборієнтованих систем підбору комп’ютерних комплектуючих.

Метою дипломної роботи є проєктування та розроблення веборієнтованої системи підбору комп’ютерних комплектуючих, яка забезпечує формування сумісних конфігурацій персональних комп’ютерів відповідно до потреб користувача.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

– провести аналіз ринку комп’ютерних комплектуючих та особливостей їх вибору;

– дослідити існуючі системи підбору комплектуючих та визначити їх переваги і недоліки;

– сформулювати функціональні та нефункціональні вимоги до системи;

– спроєктувати архітектуру веборієнтованої системи;

– розробити структуру бази даних для зберігання інформації про комплектуючі та користувачів;

– реалізувати серверну та клієнтську частини системи;

– забезпечити перевірку сумісності комплектуючих та формування рекомендацій;

– провести тестування програмного продукту та оцінити результати його роботи.

Методи дослідження. Під час виконання роботи використано методи системного аналізу, об’єктно-орієнтованого проєктування, моделювання баз даних, вебпрограмування, тестування програмного забезпечення та аналізу ефективності інформаційних систем.

Практичне значення роботи полягає у створенні веборієнтованої системи, яка може використовуватися користувачами для підбору сумісних комп’ютерних комплектуючих, формування оптимальних конфігурацій

персональних комп’ютерів та спрощення процесу прийняття рішень під час купівлі обладнання.

Структура роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної області та сформульовано вимоги до системи. Другий розділ присвячений проєктуванню архітектури, бази даних та інтерфейсу користувача. У третьому розділі розглянуто процес розроблення та реалізації системи. Також третій розділ містить результати тестування та оцінку ефективності роботи програмного продукту. У четвертому розділі висвітлено питання безпеки життєдіяльності та охорони праці під час розроблення програмного забезпечення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз ринку комп’ютерних комплектуючих

Ринок комп’ютерних комплектуючих є одним із найбільш динамічних сегментів галузі інформаційних технологій. Його розвиток безпосередньо пов’язаний із зростанням попиту на персональні комп’ютери, ноутбуки, серверне обладнання та спеціалізовані обчислювальні системи. У сучасних умовах цифровізації суспільства комп’ютерна техніка використовується практично в усіх сферах діяльності людини, що зумовлює постійну потребу в оновленні та модернізації апаратного забезпечення.

Комп’ютерні комплектуючі являють собою окремі апаратні компоненти, з яких формується комп’ютерна система. До основних комплектуючих належать центральний процесор, материнська плата, оперативна пам’ять, відеокарта, накопичувачі даних, блок живлення, система охолодження та корпус. Кожен із цих компонентів має власні технічні характеристики, які впливають на продуктивність, енергоефективність та функціональні можливості комп’ютера.

Сучасний ринок комплектуючих характеризується високим рівнем конкуренції між виробниками. Провідними компаніями у сфері виробництва процесорів є Intel [11] та AMD [1], які постійно вдосконалюють архітектуру своїх продуктів та підвищують їх продуктивність. У сегменті графічних прискорювачів лідируючі позиції займають NVIDIA та AMD. Виробництвом оперативної пам’яті та накопичувачів займаються такі компанії, як Kingston [13], Samsung, Crucial, Western Digital та Seagate. Материнські плати та інші комплектуючі випускають ASUS, MSI, Gigabyte, ASRock та інші відомі виробники.

Однією з характерних особливостей ринку комп’ютерних комплектуючих є швидке моральне старіння продукції. Нові покоління процесорів, відеокарт та інших компонентів з’являються щороку, забезпечуючи покращення технічних характеристик і продуктивності [4]. Це змушує користувачів регулярно

оновлювати обладнання для підтримання конкурентоспроможності своїх систем та можливості використання сучасного програмного забезпечення.

Важливим фактором розвитку ринку є поширення ігрової індустрії, кіберспорту, технологій штучного інтелекту, машинного навчання та обробки великих обсягів даних. Зростання вимог до обчислювальної потужності стимулює попит на високопродуктивні процесори, відеокарти та швидкі накопичувачі даних. Особливо помітним є збільшення попиту на графічні процесори, які використовуються не лише для комп'ютерних ігор, а й для виконання складних обчислень у наукових та промислових сферах.

Значну роль у розвитку ринку відіграє електронна комерція. Більшість користувачів здійснює пошук та придбання комплектуючих через інтернет-магазини, які пропонують широкий асортимент продукції, технічні характеристики, відгуки покупців та інструменти порівняння товарів. Це сприяє підвищенню прозорості ринку та спрощує процес вибору необхідного обладнання.

Попри значну кількість доступної інформації, процес підбору комплектуючих залишається складним завданням для багатьох користувачів. Основною причиною є необхідність врахування сумісності між окремими компонентами системи. Наприклад, вибір процесора повинен відповідати сокету материнської плати, оперативна пам'ять має підтримуватися чипсетом плати, а блок живлення повинен забезпечувати достатню потужність для всіх встановлених компонентів. Неврахування цих факторів може призвести до неможливості збирання системи або її нестабільної роботи.

Аналіз сучасного ринку показує, що користувачі дедалі частіше потребують автоматизованих інструментів для підбору комплектуючих. Такі системи повинні не лише забезпечувати пошук необхідних компонентів, але й автоматично перевіряти їх сумісність, аналізувати продуктивність майбутньої конфігурації та допомагати користувачеві обирати оптимальні рішення відповідно до бюджету та поставлених завдань.

Отже, ринок комп'ютерних комплектуючих характеризується високими темпами розвитку, значною кількістю виробників та широким асортиментом

продукції. Водночас складність вибору та необхідність врахування великої кількості технічних параметрів створюють передумови для розроблення веборієнтованих систем автоматизованого підбору комплектуючих, здатних спростити процес формування комп'ютерних конфігурацій та підвищити ефективність прийняття рішень користувачами.

1.2 Огляд існуючих систем підбору комплектуючих

З розвитком інформаційних технологій та збільшенням кількості комп'ютерних комплектуючих на ринку виникла потреба у спеціалізованих програмних засобах, які допомагають користувачам формувати конфігурації персональних комп'ютерів. Такі системи забезпечують автоматизований підбір компонентів, перевірку їх сумісності, розрахунок вартості збірки та надання додаткової інформації щодо характеристик обладнання.

На сьогодні існує велика кількість веборієнтованих сервісів для підбору комп'ютерних комплектуючих. Найбільш популярними серед них є PCPartPicker, PC Builder, BuildMyPC та Newegg PC Builder. Ці системи використовуються як професійними збирачами комп'ютерної техніки, так і звичайними користувачами, які бажають самостійно сформувати оптимальну конфігурацію ПК.

Однією з найвідоміших платформ є PCPartPicker [23]. Даний сервіс дозволяє користувачам створювати конфігурації комп'ютерів із використанням великої бази комплектуючих від різних виробників. Система автоматично перевіряє сумісність між компонентами, попереджає про можливі конфлікти та надає інформацію про ціни в різних магазинах. Додатково користувачі можуть переглядати готові збірки інших користувачів та використовувати їх як основу для власних проєктів.

Іншою популярною платформою є PC Builder, яка орієнтована на швидке створення комп'ютерних конфігурацій відповідно до бюджету користувача [22]. Сервіс пропонує рекомендації щодо вибору комплектуючих та дозволяє здійснювати базову перевірку їх сумісності. Водночас функціональні

можливості системи є дещо обмеженими порівняно з більш професійними рішеннями.

Система BuildMyPC також забезпечує автоматизований підбір комплектуючих та перевірку сумісності компонентів [3]. Її особливістю є зручний інтерфейс користувача та можливість швидкого формування збірок для різних категорій користувачів, зокрема для геймерів, розробників програмного забезпечення та офісних працівників.

Компанія Newegg пропонує власний інструмент підбору комплектуючих – Newegg PC Builder [19]. Система інтегрована безпосередньо з інтернет-магазином компанії та дозволяє не лише формувати конфігурації, але й одразу здійснювати замовлення вибраних компонентів. Основною перевагою сервісу є актуальність цін та наявність інформації про доступність товарів. Для кращого порівняння функціональних можливостей існуючих систем доцільно провести їх аналіз за основними критеріями (табл. 1.1).

Таблиця 1.1 – Порівняння існуючих систем підбору комп'ютерних комплектуючих

Критерій	**PCPartPicker**	**PC Builder**	**BuildMyPC**	**Newegg PC Builder**
Перевірка сумісності комплектуючих	Так	Частково	Так	Так
Велика база комплектуючих	Так	Так	Так	Так
Формування конфігурацій за бюджетом	Так	Так	Так	Частково
Порівняння цін	Так	Ні	Частково	Ні
Інтеграція з інтернет-магазином	Частково	Ні	Ні	Так
Збереження конфігурацій	Так	Так	Так	Так
Перегляд готових збірок	Так	Ні	Частково	Ні
Простота використання	Висока	Висока	Висока	Середня
Підтримка української мови	Ні	Ні	Ні	Ні

Проведений аналіз показує, що сучасні системи підбору комплектуючих мають широкий функціонал та дозволяють значно спростити процес формування комп'ютерних конфігурацій. Проте більшість із них орієнтована на міжнародний ринок і не враховує особливості локальних користувачів. Зокрема, відсутня підтримка української мови, обмежено враховується асортимент українських

магазинів, а також не реалізовано механізми адаптації рекомендацій до потреб окремих категорій користувачів.

Також частина існуючих систем має складний інтерфейс або перевантажена великою кількістю додаткових функцій, що може ускладнювати роботу недосвідчених користувачів. Також не всі сервіси забезпечують детальний аналіз сумісності компонентів та автоматичне формування оптимальних конфігурацій відповідно до заданих параметрів.

Отже, результати аналізу свідчать про доцільність створення нової веборієнтованої системи підбору комп'ютерних комплектуючих, яка поєднуватиме зручний інтерфейс користувача, автоматичну перевірку сумісності компонентів, підтримку локального ринку та можливість формування рекомендацій на основі потреб і бюджету користувача.

1.3 Формулювання вимог до веборієнтованої системи

На основі проведеного аналізу ринку комп'ютерних комплектуючих та існуючих аналогів було сформульовано вимоги до веборієнтованої системи підбору комп'ютерних комплектуючих. Визначення вимог є важливим етапом проєктування, оскільки саме вони формують основу для подальшої розробки архітектури, структури бази даних та функціональних модулів програмного продукту.

Основною метою системи є забезпечення користувачів зручним інструментом для вибору, порівняння та замовлення комп'ютерних комплектуючих із можливістю автоматичної перевірки їх сумісності. До функціональних вимог системи належать:

– реєстрація нових користувачів та авторизація зареєстрованих користувачів;

– керування особистим профілем користувача;

– перегляд каталогу комп'ютерних комплектуючих;

– пошук товарів за назвою;

– фільтрація товарів за категоріями;

– перегляд детальної інформації про комплектуючі;

– додавання товарів до кошика;

– зміна кількості товарів у кошику;

– видалення товарів із кошика;

– оформлення замовлень із вибором способу доставки та оплати;

– перегляд історії замовлень;

– перегляд детальної інформації про оформлені замовлення;

– створення конфігурацій комп’ютерних систем;

– перевірка сумісності комплектуючих;

– порівняння декількох конфігурацій комп’ютера;

– формування рекомендацій щодо використання конфігурацій для різних завдань;

– адміністрування каталогу товарів;

– додавання нових товарів;

– редагування характеристик товарів;

– завантаження зображень комплектуючих;

– керування складськими залишками.

До нефункціональних вимог системи належать:

– забезпечення зручного та зрозумілого інтерфейсу користувача;

– підтримка роботи в сучасних веббраузерах;

– швидка обробка запитів користувачів;

– забезпечення цілісності та достовірності даних;

– захист персональних даних користувачів;

– контроль доступу до адміністративних функцій;

– можливість масштабування системи;

– підтримка подальшого розширення функціональних можливостей;

– забезпечення стабільної роботи при одночасному використанні декількома користувачами.

Особливу увагу в системі приділено перевірці сумісності комплектуючих. Для цього використовуються технічні характеристики компонентів, які зберігаються в базі даних та аналізуються під час формування конфігурації

комп’ютера. Це дозволяє своєчасно виявляти потенційні конфлікти між компонентами та запобігати помилкам при виборі обладнання.

Отже, сформульовані вимоги визначають основні функціональні можливості та характеристики майбутньої веборієнтованої системи підбору комп’ютерних комплектуючих і забезпечують основу для її подальшого проєктування та реалізації.

1.4 Постановка задачі проєктування системи

На підставі аналізу предметної області та сформульованих вимог поставлено задачу проєктування та розроблення веборієнтованої системи підбору комп’ютерних комплектуючих, яка забезпечуватиме автоматизацію процесів вибору, конфігурації та замовлення компонентів персонального комп’ютера.

Система повинна функціонувати за клієнт-серверною архітектурою та надавати користувачам можливість взаємодії через вебінтерфейс. Основними користувачами системи є покупці комп’ютерних комплектуючих та адміністратори, які здійснюють керування каталогом товарів.

У межах проєктування необхідно реалізувати такі основні підсистеми:

– підсистему керування обліковими записами користувачів;

– підсистему перегляду каталогу комплектуючих;

– підсистему пошуку та фільтрації товарів;

– підсистему конфігурації комп’ютерних збірок;

– підсистему перевірки сумісності комплектуючих;

– підсистему порівняння конфігурацій;

– підсистему керування кошиком покупця;

– підсистему оформлення та обробки замовлень;

– підсистему адміністрування товарів і складських залишків.

Для реалізації зазначених функцій необхідно розробити структуру бази даних, яка забезпечуватиме зберігання інформації про користувачів, категорії

товарів, комплектуючі, кошики та замовлення. База даних повинна підтримувати зв’язки між сутностями та забезпечувати швидкий доступ до інформації.

У процесі проєктування також необхідно створити механізм перевірки сумісності комплектуючих на основі технічних характеристик процесорів, материнських плат та оперативної пам’яті. Результатом роботи механізму повинно бути інформування користувача про можливі конфлікти між вибраними компонентами.

Для забезпечення безпеки системи необхідно реалізувати механізми аутентифікації та авторизації користувачів із розмежуванням прав доступу між звичайними користувачами та адміністраторами. Адміністративна частина системи повинна надавати можливість керування каталогом товарів, редагування характеристик комплектуючих та контролю складських залишків.

Результатом виконання поставленого завдання має стати веборієнтована система, яка забезпечує ефективний підбір комп’ютерних комплектуючих, автоматичну перевірку їх сумісності, формування комп’ютерних конфігурацій, оформлення замовлень та адміністрування товарного каталогу через зручний вебінтерфейс.

1.5 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було проведено аналіз предметної області веборієнтованих систем підбору комп’ютерних комплектуючих. Дослідження сучасного ринку комп’ютерного обладнання показало, що він характеризується високими темпами розвитку, значною кількістю виробників та широким асортиментом продукції, що ускладнює процес вибору комплектуючих для користувачів. Особливу складність становить необхідність врахування технічної сумісності між окремими компонентами комп’ютерної системи.

У результаті аналізу існуючих вебсервісів для підбору комплектуючих, зокрема PCPartPicker, PC Builder, BuildMyPC та Newegg PC Builder, було визначено їх основні переваги та недоліки. Встановлено, що сучасні системи

забезпечують перевірку сумісності компонентів і формування конфігурацій, однак більшість із них орієнтована на міжнародний ринок, не підтримує українську мову та недостатньо враховує потреби локальних користувачів.

На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до майбутньої веборієнтованої системи. Визначено основні можливості програмного продукту, серед яких реєстрація та авторизація користувачів, перегляд каталогу товарів, пошук і фільтрація комплектуючих, створення конфігурацій ПК, перевірка сумісності компонентів, порівняння збірок, керування кошиком та оформлення замовлень. Також визначено вимоги щодо безпеки, швидкодії, масштабованості та зручності використання системи.

У підрозділі постановки задачі визначено структуру майбутньої системи та перелік основних підсистем, необхідних для її функціонування. Зокрема, передбачено реалізацію модулів керування користувачами, каталогом товарів, конфігуратором ПК, механізмом перевірки сумісності, кошиком замовлень та адміністративною панеллю.

Отже, результати проведеного аналізу підтвердили актуальність розроблення веборієнтованої системи підбору комп’ютерних комплектуючих та створили теоретичне підґрунтя для її подальшого проєктування, моделювання та програмної реалізації, що розглядаються у наступних розділах роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ

2.1 Архітектура системи

Архітектура веборієнтованої системи підбору комп'ютерних комплектуючих побудована за клієнт-серверним принципом. Такий підхід забезпечує розділення логіки представлення даних, бізнес-логіки та зберігання інформації, що спрощує підтримку, масштабування та подальший розвиток програмного продукту.

Система складається з трьох основних рівнів: клієнтської частини (Frontend), серверної частини (Backend) та бази даних. Клієнтська частина реалізована за допомогою бібліотеки React та забезпечує взаємодію користувача із системою через вебінтерфейс. Серверна частина створена на платформі Node.js із використанням фреймворку Express.js і відповідає за обробку запитів, реалізацію бізнес-логіки, автентифікацію користувачів та взаємодію з базою даних. Для зберігання інформації використовується реляційна система керування базами даних MySQL.

Користувач взаємодіє з вебінтерфейсом через браузер, здійснюючи перегляд каталогу комплектуючих, формування комп'ютерних конфігурацій, оформлення замовлень та керування власним профілем. Усі запити від клієнтської частини передаються до REST API серверної частини за допомогою HTTP-протоколу. Сервер виконує обробку отриманих даних, звертається до бази даних та повертає результат клієнту у форматі JSON.

Система підтримує розмежування прав доступу між звичайними користувачами та адміністраторами. Адміністратор має доступ до функцій створення, редагування та видалення товарів, керування складськими залишками та перегляду інформації про замовлення. Для забезпечення безпеки використовується автентифікація на основі JSON Web Token (JWT), а паролі користувачів зберігаються у зашифрованому вигляді за допомогою алгоритму bcrypt.

Для реалізації функціоналу завантаження зображень використовується окремий модуль роботи з файлами. Завантажені зображення зберігаються на сервері та використовуються для відображення товарів у каталозі.

Архітектура системи забезпечує незалежність клієнтської та серверної частин, що дозволяє модернізувати окремі компоненти без впливу на роботу всієї системи. Такий підхід відповідає сучасним принципам розробки вебзастосунків і забезпечує високу продуктивність та зручність підтримки програмного продукту. UML-діаграма компонентів системи представлена на рисунку 2.1.

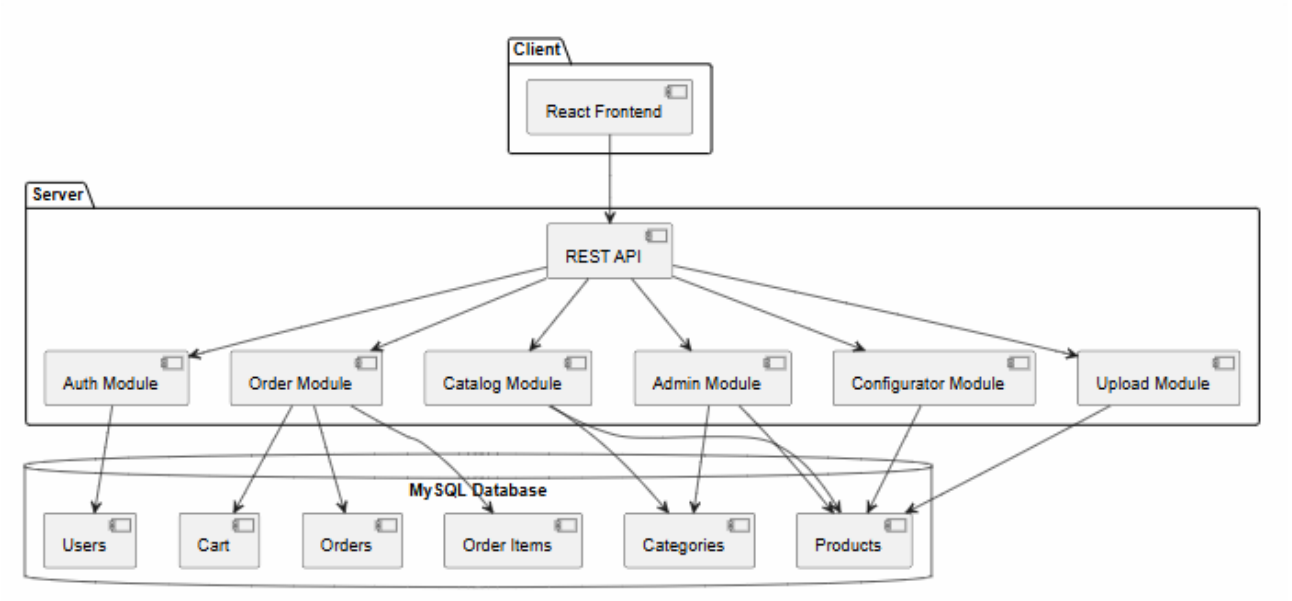

Рисунок 2.1 – UML-діаграма компонентів архітектури веборієнтованої системи підбору комп’ютерних комплектуючих

На діаграмі відображено основні компоненти системи та взаємозв’язки між ними. Центральним елементом архітектури є REST API, яке забезпечує взаємодію між клієнтською частиною та серверними модулями. Модуль автентифікації відповідає за реєстрацію та авторизацію користувачів. Модуль каталогу забезпечує доступ до інформації про комплектуючі та категорії товарів. Модуль конфігуратора реалізує формування комп’ютерних збірок та перевірку сумісності компонентів. Модуль замовлень відповідає за роботу кошика та оформлення покупок. Адміністративний модуль забезпечує керування товарами та складськими залишками, а модуль завантаження файлів реалізує роботу із

зображеннями товарів. Усі модулі взаємодіють із централізованою базою даних MySQL, що забезпечує зберігання та обробку інформації.

2.2 Проєктування бази даних

База даних є одним із ключових компонентів веборієнтованої системи підбору комп'ютерних комплектуючих, оскільки забезпечує зберігання, обробку та швидкий доступ до інформації про користувачів, товари, категорії, кошики та замовлення. Для реалізації системи було обрано реляційну систему керування базами даних MySQL, яка характеризується високою продуктивністю, надійністю та широкими можливостями інтеграції з вебтехнологіями.

Під час проєктування бази даних було виконано аналіз предметної області та визначено основні сутності, необхідні для функціонування системи. Структура бази даних побудована відповідно до принципів нормалізації, що дозволяє уникнути дублювання інформації та забезпечує цілісність даних.

До складу бази даних входять такі основні таблиці:

- users – інформація про користувачів системи;
- categories – категорії комп'ютерних комплектуючих;
- products – товари каталогу;
- cart_items – товари, додані до кошика;
- orders – замовлення користувачів;
- order_items – склад кожного замовлення.

Таблиця users містить дані про зареєстрованих користувачів системи. У ній зберігаються ім'я користувача, адреса електронної пошти, пароль у зашифрованому вигляді, номер телефону та роль користувача. Роль визначає рівень доступу до функціоналу системи та може набувати значення «користувач» або «адміністратор».

Таблиця categories використовується для зберігання категорій комплектуючих. Кожен товар належить до певної категорії, наприклад процесори, материнські плати, оперативна пам'ять, відеокарти, накопичувачі або блоки живлення.

Таблиця products є центральною таблицею бази даних і містить інформацію про всі комплектуючі, доступні в каталозі. Для кожного товару зберігаються назва, опис, ціна, залишок на складі, категорія, зображення та набір технічних характеристик. Для зберігання специфічних характеристик використовується поле формату JSON, що дозволяє гнучко описувати параметри різних типів комплектуючих без зміни структури таблиці.

Таблиця cart_items використовується для реалізації функціоналу кошика покупця. У ній зберігаються відомості про вибрані товари та їх кількість для кожного користувача.

Таблиця orders містить інформацію про оформлені замовлення. Для кожного замовлення зберігаються дані про користувача, адресу доставки, спосіб оплати, спосіб доставки, дату створення та поточний статус замовлення.

Таблиця order_items містить перелік товарів, що входять до конкретного замовлення. Така структура дозволяє реалізувати зв’язок типу «один до багатьох» між замовленням та його складовими елементами.

Для забезпечення цілісності даних між таблицями використовуються первинні та зовнішні ключі. Зв’язки між сутностями дозволяють гарантувати коректність збереженої інформації та підтримувати узгодженість даних під час виконання операцій додавання, редагування або видалення записів.

На рисунку 2.2 представлено структуру бази даних розробленої системи. Центральною сутністю є таблиця Products, яка містить інформацію про комп’ютерні комплектуючі. Кожен товар належить до певної категорії, що реалізується через зв’язок між таблицями Categories та Products. Користувачі можуть додавати товари до кошика та оформлювати замовлення, що відображено зв’язками між таблицями Users, Cart_Items, Orders та Order_Items.

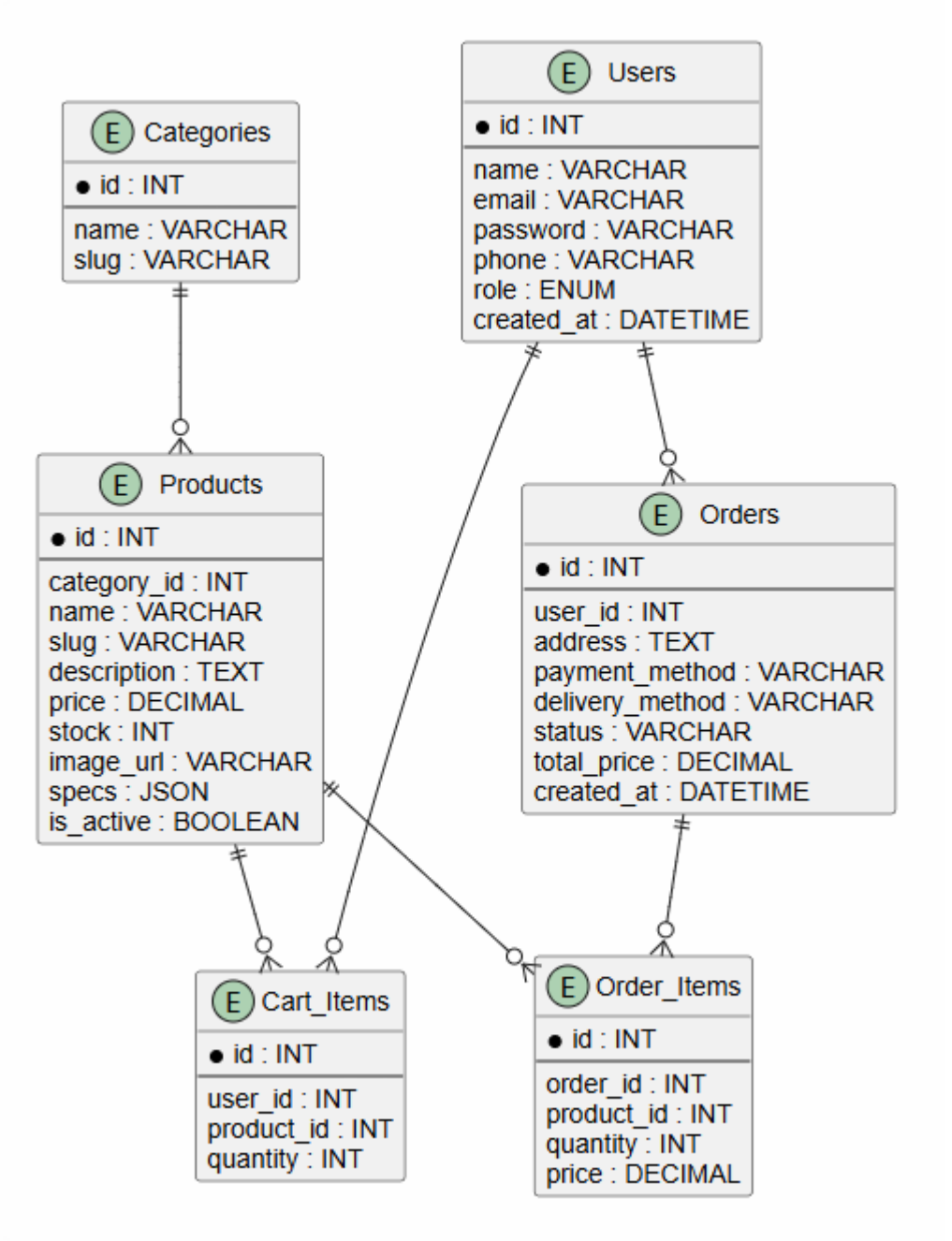

Рисунок 2.2 – UML-діаграма структури бази даних системи

Така структура забезпечує ефективне зберігання даних, підтримує логічну цілісність інформації та дозволяє реалізувати весь функціонал веборієнтованої системи підбору комп’ютерних комплектуючих.

2.3 Моделювання функціоналу системи

Моделювання функціоналу є важливим етапом проєктування програмного забезпечення, який дозволяє визначити основні сценарії взаємодії користувачів

із системою, описати функціональні можливості програмного продукту та встановити взаємозв’язки між його компонентами. Для моделювання функціоналу веборієнтованої системи підбору комп’ютерних комплектуючих доцільно використовувати UML-діаграми варіантів використання (Use Case Diagram).

У розробленій системі передбачено дві основні категорії користувачів: звичайний користувач та адміністратор. Звичайний користувач взаємодіє з каталогом комплектуючих, формує конфігурації персонального комп’ютера, здійснює покупки та переглядає власні замовлення. Адміністратор, крім функцій звичайного користувача, має можливість керувати каталогом товарів, редагувати характеристики комплектуючих та контролювати наявність продукції на складі.

Основним завданням користувача є формування сумісної комп’ютерної конфігурації шляхом вибору необхідних комплектуючих із каталогу системи. Для цього передбачено механізм перевірки сумісності компонентів, який аналізує характеристики процесора, материнської плати та оперативної пам’яті. Після формування конфігурації користувач може додати товари до кошика та оформити замовлення.

Для адміністратора реалізовано окремий набір функцій, пов’язаний із супроводом інформаційної системи. Адміністратор може додавати нові товари до каталогу, редагувати існуючі записи, завантажувати зображення комплектуючих та оновлювати інформацію про складські залишки.

На діаграмі на рисунку 2.3 представлено взаємодію користувачів із веборієнтованою системою. Звичайний користувач може виконувати реєстрацію та авторизацію, переглядати каталог товарів, здійснювати пошук і фільтрацію комплектуючих, додавати товари до кошика, оформлювати замовлення та керувати власним профілем. Однією з основних функцій системи є формування конфігурації персонального комп’ютера, яка включає автоматичну перевірку сумісності вибраних компонентів.

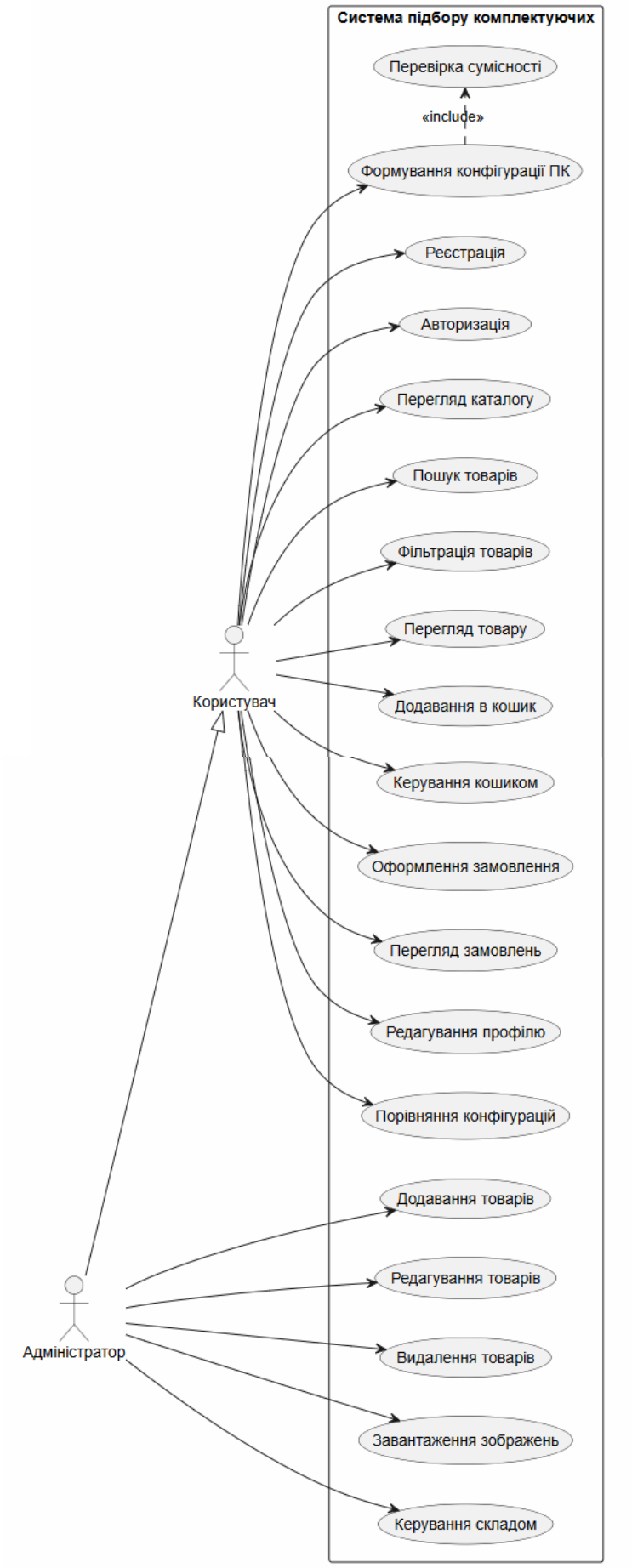

Рисунок 2.3 – UML-діаграма варіантів використання системи

Адміністратор наслідує всі можливості звичайного користувача та додатково отримує доступ до функцій керування каталогом товарів. Він може створювати нові записи про комплектуючі, редагувати існуючі товари, видаляти застарілі позиції, завантажувати зображення та контролювати залишки продукції на складі.

Для більш детального відображення роботи конфігуратора доцільно розглянути сценарій формування комп'ютерної збірки. Користувач обирає комплектуючі з каталогу та додає їх до конфігуратора. Після кожного додавання система аналізує технічні характеристики компонентів та перевіряє їх сумісність. У разі виявлення конфлікту користувач отримує відповідне повідомлення із зазначенням проблемного компонента. Якщо всі комплектуючі сумісні між собою, система підтверджує можливість створення конфігурації та дозволяє зберегти або порівняти її з іншими варіантами.

Отже, побудована модель функціоналу дозволяє наочно представити основні можливості веборієнтованої системи та визначити взаємодію між користувачами й окремими функціональними модулями програмного продукту.

2.4 Розробка інтерфейсу користувача

Інтерфейс користувача є важливою складовою веборієнтованої системи підбору комп'ютерних комплектуючих, оскільки саме через нього користувач взаємодіє з основними функціями програмного продукту. Під час розробки інтерфейсу було враховано зручність навігації, логічне розміщення елементів, доступність основних функцій та простоту використання системи для користувачів із різним рівнем технічної підготовки.

Інтерфейс системи реалізовано як вебзастосунок, доступний через браузер. Основними сторінками системи є головна сторінка каталогу, сторінка товару, конфігуратор комп'ютерної збірки, кошик, сторінка оформлення замовлення, особистий профіль користувача, історія замовлень та адміністративна панель керування товарами.

Головна сторінка містить каталог комп’ютерних комплектуючих, блок фільтрації за категоріями, пошуковий рядок та картки товарів. Картка товару містить назву комплектуючого, зображення, ціну, кількість на складі та кнопку для додавання товару до кошика або конфігуратора. Такий підхід дозволяє користувачеві швидко переглянути основну інформацію та перейти до детального опису товару.

Окрему роль у системі відіграє сторінка конфігуратора. Вона призначена для формування комп’ютерної збірки з вибраних комплектуючих. На цій сторінці користувач бачить список доданих компонентів, загальну вартість збірки, повідомлення про сумісність та блок порівняння двох конфігурацій. Завдяки цьому користувач може не лише сформувати власну збірку, а й оцінити її придатність для ігор, програмування або відеомонтажу.

Інтерфейс кошика забезпечує перегляд доданих товарів, зміну їх кількості, видалення позицій та перехід до оформлення замовлення. На сторінці оформлення замовлення користувач вводить адресу доставки, обирає спосіб оплати та спосіб доставки. Після підтвердження система створює замовлення та зменшує кількість товарів на складі.

Для зареєстрованих користувачів передбачено особистий кабінет, у якому можна переглядати та редагувати персональні дані. Також користувач має доступ до історії замовлень та деталей кожного окремого замовлення.

Адміністративна панель призначена для керування каталогом комплектуючих. Адміністратор може додавати нові товари, редагувати назву, опис, ціну, залишок, категорію, характеристики у форматі JSON та зображення товару. Такий підхід дозволяє підтримувати актуальність каталогу та швидко вносити зміни до інформації про комплектуючі.

Під час проєктування інтерфейсу було використано принципи адаптивності та простоти. Основні елементи розміщені так, щоб користувач міг виконати потрібну дію з мінімальною кількістю переходів між сторінками. Колірна схема побудована на поєднанні світлого фону, темного тексту та акцентного помаранчевого кольору, що робить інтерфейс сучасним і візуально зрозумілим.

На рисунку 2.4 наведено спрощений прототип головної сторінки системи з каталогом товарів, пошуком, фільтрацією та блоком конфігуратора.

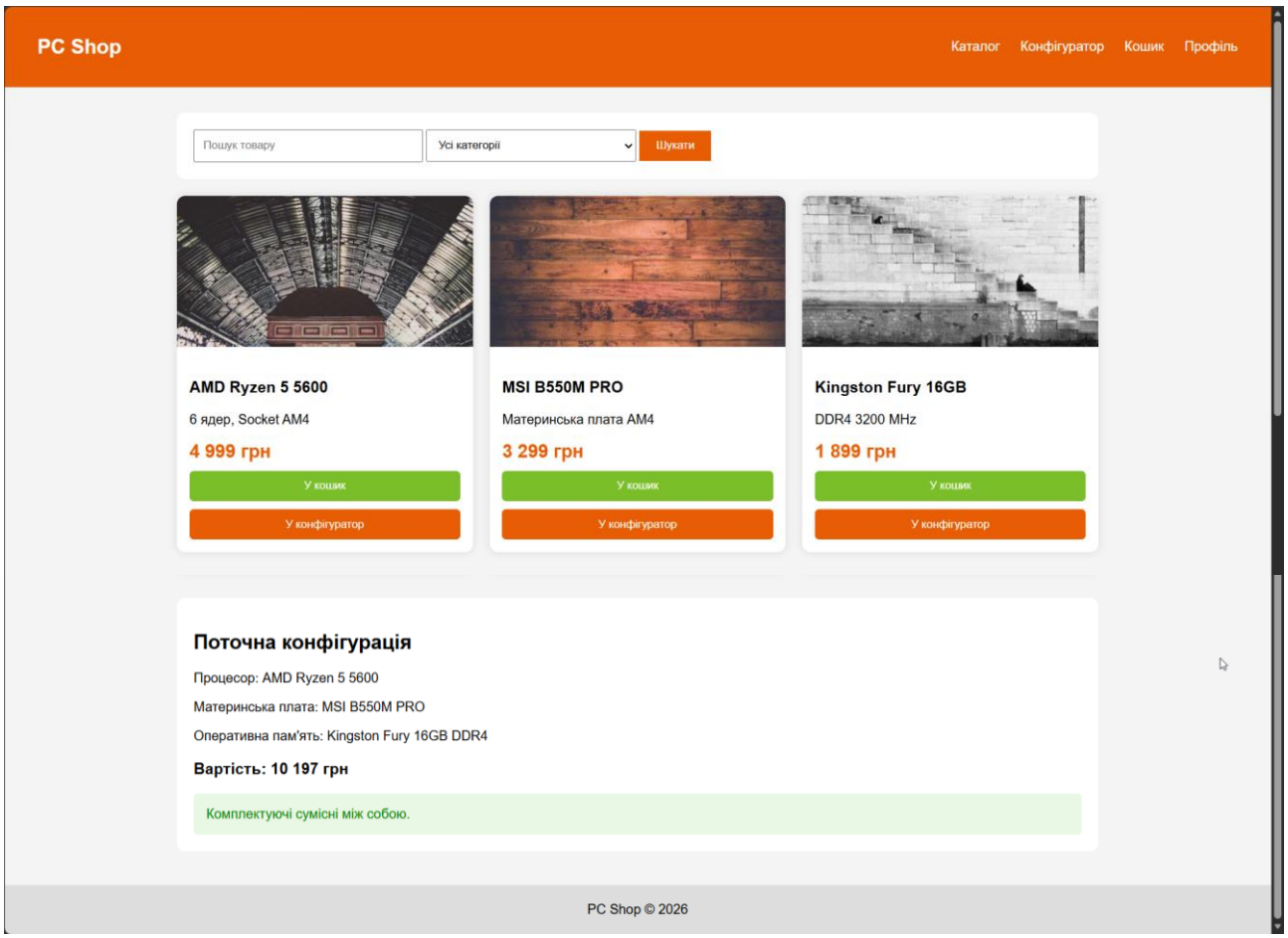

Рисунок 2.4 – HTML-прототип головної сторінки веборієнтованої системи

Наведений прототип демонструє загальну структуру інтерфейсу користувача. Він містить верхню навігаційну панель, блок пошуку та фільтрації товарів, каталог комплектуючих у вигляді карток, кнопки додавання товарів до кошика та конфігуратора, а також блок поточної комп’ютерної збірки з повідомленням про сумісність. У подальшій реалізації цей прототип було розширено та адаптовано за допомогою React-компонентів, маршрутизації React Router і стилізації Tailwind CSS.

2.5 Висновок до другого розділу

У другому розділі кваліфікаційної роботи було виконано проєктування веборієнтованої системи підбору комп’ютерних комплектуючих. На основі сформульованих вимог розроблено архітектуру програмного продукту, яка побудована за клієнт-серверним принципом та забезпечує розподіл функцій між клієнтською частиною, сервером застосунку та базою даних. Обрана архітектура дозволяє забезпечити масштабованість, надійність та зручність подальшого супроводу системи.

У процесі проєктування було розроблено структуру бази даних, що включає сутності користувачів, категорій товарів, комплектуючих, кошика та замовлень. Визначено зв’язки між таблицями та реалізовано логічну модель зберігання даних, яка забезпечує цілісність інформації та ефективне виконання операцій пошуку, обробки та оновлення даних.

Для формалізації функціональних можливостей системи побудовано UML-діаграми, які відображають взаємодію користувачів із програмним продуктом. У результаті моделювання визначено основні сценарії використання системи, включаючи перегляд каталогу, пошук товарів, формування конфігурацій комп’ютера, перевірку сумісності комплектуючих, оформлення замовлень та адміністрування каталогу товарів.

Особливу увагу приділено проєктуванню механізму формування комп’ютерних збірок і перевірки сумісності компонентів. Передбачено автоматичний аналіз технічних характеристик процесорів, материнських плат та оперативної пам’яті, що дозволяє виявляти потенційні конфлікти між комплектуючими та підвищує якість підбору конфігурацій.

У межах проєктування інтерфейсу користувача створено структуру основних сторінок системи, визначено принципи навігації та організацію взаємодії користувача з функціональними модулями. Розроблений прототип інтерфейсу забезпечує зручний доступ до каталогу товарів, конфігуратора, кошика, профілю користувача та адміністративної панелі, що сприяє підвищенню зручності використання системи.

Отже, у результаті виконання другого розділу сформовано повну проєктну модель веборієнтованої системи підбору комп’ютерних комплектуючих, яка включає архітектуру програмного забезпечення, структуру бази даних, модель функціоналу та інтерфейс користувача. Отримані результати створюють основу для практичної реалізації програмного продукту та його подальшого тестування у наступних розділах роботи.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір технологій та інструментів розробки

Одним із найважливіших етапів створення веборієнтованої системи підбору комп'ютерних комплектуючих є вибір технологій та інструментів розробки. Від правильності цього вибору залежить продуктивність системи, зручність підтримки програмного коду, можливість подальшого масштабування та швидкість реалізації функціоналу. Для розроблення програмного продукту було обрано сучасний стек вебтехнологій, який забезпечує високу швидкодію, гнучкість та широкі можливості для створення інтерактивних вебзастосунків. Архітектура системи побудована за принципом клієнт-серверної взаємодії, що відповідає сучасним підходам до розробки вебсистем [26].

Для реалізації клієнтської частини використано бібліотеку React [24]. React є однією з найпоширеніших технологій для створення користувацьких інтерфейсів та дозволяє будувати вебзастосунки на основі багаторазово використовуваних компонентів. Використання React забезпечує високу швидкість оновлення інтерфейсу завдяки механізму Virtual DOM та спрощує підтримку програмного коду. Для навігації між сторінками застосовано React Router [25], який реалізує маршрутизацію без перезавантаження сторінок та покращує користувацький досвід.

Основою структури вебсторінок стали технології HTML та CSS [6, 10]. Для стилізації інтерфейсу використано фреймворк Tailwind CSS [27], який дозволяє швидко створювати адаптивні користувацькі інтерфейси за допомогою готових класів стилів. Це значно скорочує час розробки та забезпечує єдиний стиль оформлення всіх компонентів системи.

Як інструмент збірки проєкту використано Vite [30]. Перевагою даного інструмента є висока швидкість запуску середовища розробки та підтримка миттєвого оновлення сторінок під час внесення змін до програмного коду. Для програмування клієнтської та серверної частин використовувалась мова JavaScript [15].

Серверну частину реалізовано на платформі Node.js [20] із використанням фреймворку Express.js [7]. Node.js дозволяє виконувати JavaScript-код на стороні сервера та забезпечує ефективну обробку великої кількості одночасних запитів. Express.js значно спрощує створення REST API, організацію маршрутизації та реалізацію серверної логіки.

Для забезпечення безпечної автентифікації користувачів використано технологію JSON Web Token (JWT) [12]. JWT дозволяє передавати інформацію про авторизованого користувача між клієнтом і сервером без необхідності зберігання сесій на сервері. Для захисту паролів використовується бібліотека bcryptjs [2], яка реалізує сучасні алгоритми хешування та забезпечує надійне зберігання облікових даних користувачів.

Для роботи із завантаженням файлів використовується бібліотека Multer [17]. Вона забезпечує приймання та обробку файлів, що надходять через HTTP-запити, а також їх подальше збереження на сервері. Взаємодія між клієнтською та серверною частинами здійснюється через REST API [26], а передача даних виконується у форматі JSON. Для виконання HTTP-запитів використовується стандартний механізм Fetch API [14].

Для зберігання інформації про користувачів, комплектуючі, конфігурації, кошики та замовлення використовується система керування базами даних MySQL [18]. Вона забезпечує надійне зберігання інформації, підтримує реляційні зв’язки між таблицями та дозволяє ефективно виконувати складні SQL-запити. Взаємодія серверної частини з базою даних реалізована через бібліотеку mysql2 та механізм пулу з’єднань.

Під час проєктування структури системи використовувалися принципи UML-моделювання [28], що дозволило формалізувати функціональні вимоги та описати взаємодію користувачів із системою. Для контролю версій програмного коду застосовувалася система Git [8], яка забезпечує зручне керування змінами та спрощує процес розробки.

Розробка програмного продукту виконувалась у середовищі Visual Studio Code [29]. Даний редактор коду підтримує велику кількість розширень для

роботи з JavaScript, React, Node.js та MySQL, що суттєво підвищує продуктивність розробника.

Структура програмного проєкту складається з двох основних частин: каталогу client, який містить клієнтську частину на React, та каталогу server, у якому реалізовано серверну логіку, REST API, взаємодію з базою даних і завантаження файлів. Такий підхід забезпечує чітке розділення відповідальності між компонентами системи та відповідає сучасним стандартам розробки вебзастосунків. Технології та інструменти розробки систематизовано в таблиці 3.1.

Таблиця 3.1 – Технології та інструменти розробки

Технологія	**Призначення**
React	Розробка клієнтської частини
React Router	Маршрутизація сторінок
Tailwind CSS	Стилізація інтерфейсу
Vite	Збірка та запуск проєкту
Node.js	Виконання серверного коду
Express.js	Реалізація REST API
JWT	Автентифікація користувачів
bcryptjs	Хешування паролів
Multer	Завантаження файлів
MySQL	Зберігання даних
mysql2	Робота з базою даних
JSON	Формат обміну даними
Visual Studio Code	Середовище розробки
Git	Контроль версій

Отже, обраний набір технологій повністю відповідає вимогам до веборієнтованої системи підбору комп’ютерних комплектуючих та забезпечує ефективну реалізацію всіх необхідних функціональних можливостей програмного продукту.

3.2 Реалізація серверної частини

Серверна частина веборієнтованої системи підбору комп’ютерних комплектуючих реалізована за допомогою платформи Node.js із використанням фреймворку Express.js. Такий підхід дозволяє забезпечити високу швидкодію системи, простоту масштабування та зручну інтеграцію з клієнтською частиною,

реалізованою на React. Основним завданням серверної частини є обробка HTTP-запитів користувачів, взаємодія з базою даних MySQL, реалізація бізнес-логіки, авторизація користувачів, керування каталогом товарів, кошиком та замовленнями.

Архітектура серверної частини побудована за принципом REST API. Для кожного функціонального модуля створено окремий набір маршрутів (routes), що забезпечує модульність та спрощує підтримку програмного коду. Сервер підтримує роботу з каталогом товарів, обліковими записами користувачів, кошиком покупця, замовленнями та адміністративною панеллю. Для запуску вебсервера використовується головний файл застосунку, у якому виконується підключення необхідних бібліотек, налаштування middleware та реєстрація маршрутів (лістинг 3.1).

Лістинг 3.1 – Ініціалізація та запуск сервера Express.js

```
const express = require("express");
const cors = require("cors");
const cookieParser = require("cookie-parser");

const app = express();

app.use(cors({
    origin: CLIENT_ORIGIN,
    credentials: true,
}));

app.use(express.json());
app.use(cookieParser());

app.listen(PORT, () => {
    console.log(`API http://localhost:${PORT}`);
});
```

Для зберігання даних використовується реляційна база даних MySQL. Підключення до БД реалізовано через бібліотеку mysql2, яка підтримує асинхронне виконання запитів та пул з’єднань (лістинг 3.2).

Лістинг 3.2 – Підключення до бази даних MySQL через пул з’єднань

```
const mysql = require("mysql2/promise");
```

```
function getPool() {
    if (!pool) {
        pool = mysql.createPool({
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PASSWORD,
            database: process.env.DB_NAME,
            connectionLimit: 10
        });
    }
    return pool;
}
```

Використання пулу з’єднань дозволяє повторно використовувати відкриті з’єднання з базою даних, що суттєво зменшує навантаження на сервер та підвищує продуктивність системи. Для забезпечення безпечного доступу до персональних даних реалізовано систему автентифікації та авторизації користувачів на основі технології JWT (JSON Web Token). Після успішного входу в систему сервер генерує токен, який зберігається у cookie браузера та використовується для подальшої ідентифікації користувача (лістинг 3.3).

Лістинг 3.3 – Генерація JWT-токена для автентифікації користувача

```
const jwt = require("jsonwebtoken");

function signToken(payload) {
    return jwt.sign(payload, secret(), {
        expiresIn: "7d"
    });
}
```

Перевірка токена виконується за допомогою спеціального middleware, який автоматично перевіряє наявність та коректність токена перед доступом до захищених ресурсів системи (лістинг 3.4).

Лістинг 3.4 – Middleware перевірки авторизації користувача

```
function requireAuth(req, res, next) {
    const token = req.cookies?.[COOKIE_NAME];

    if (!token) {
        return res.status(401).json({
            error: "Потрібна авторизація"
        });
```

```
    }

    const payload = verifyToken(token);
    req.userId = Number(payload.sub);

    next();
}
```

Для реєстрації нового користувача реалізовано окремий API-маршрут. Перед збереженням пароля у базі даних він шифрується за допомогою алгоритму bcrypt (лістинг 3.5).

Лістинг 3.5 – Реєстрація нового користувача в системі

```
router.post("/register", async (req, res) => {

    const hash = bcrypt.hashSync(password, 10);

    await pool.query(
        `INSERT INTO users
        (email, password_hash, full_name, phone)
        VALUES (:email, :hash, :full_name, :phone)`,
        {
            email,
            hash,
            full_name,
            phone
        }
    );
});
```

Одним із ключових модулів системи є каталог комплектуючих. Для отримання списку товарів реалізовано API-маршрут, який підтримує фільтрацію за категоріями та пошук за назвою товару (лістинг 3.6).

Лістинг 3.6 – Отримання списку товарів із бази даних

```
router.get("/products", async (req, res) => {

    const [items] = await pool.query(
        `SELECT p.id,
                p.name,
                p.price,
                p.stock,
                c.name AS category_name
         FROM products p
         INNER JOIN categories c
```

```
         ON c.id = p.category_id`
    );

    res.json({ items });
});
```

Для отримання детальної інформації про конкретний товар використовується окремий маршрут, який приймає унікальний slug товару (лістинг 3.7).

Лістинг 3.7 – Отримання детальної інформації про товар

```
router.get("/products/:slug", async (req, res) => {

    const [rows] = await pool.query(
        `SELECT *
         FROM products
         WHERE slug = :slug`,
         { slug }
    );

    res.json({
        product: rows[0]
    });
});
```

Важливим функціональним модулем системи є кошик покупця. Після вибору товару користувач може додати його до кошика, змінити кількість або видалити позицію (лістинг 3.8).

Лістинг 3.8 – Додавання товару до кошика користувача

```
router.post("/items", requireAuth, async (req, res) => {

    await pool.query(
        `INSERT INTO cart_items
        (user_id, product_id, quantity)
        VALUES (:uid, :pid, :qty)`,
        {
            uid: req.userId,
            pid: productId,
            qty: addQty
        }
    );

    res.status(201).json({
        ok: true
```

```
    });
});
```

Для оформлення замовлення реалізовано механізм транзакцій MySQL. Це дозволяє гарантувати цілісність даних навіть у випадку помилок під час виконання операцій (лістинг 3.9).

Лістинг 3.9 – Створення замовлення з використанням транзакцій MySQL

```
await conn.beginTransaction();

const [ins] = await conn.query(
    `INSERT INTO orders
    (user_id, status, total)
    VALUES (:uid, 'pending', :total)`
);

await conn.commit();
```

Під час оформлення замовлення система автоматично перевіряє наявність товару на складі, розраховує загальну вартість покупки, створює запис у таблиці замовлень та оновлює залишки товарів. Для адміністрування каталогу реалізовано окремий набір маршрутів, доступ до яких мають лише користувачі з роллю адміністратора. Перевірка ролі виконується за допомогою middleware requireAdmin (лістинг 3.10).

Лістинг 3.10 – Перевірка прав адміністратора

```
async function requireAdmin(req, res, next) {

    const [rows] = await pool.query(
        `SELECT role
         FROM users
         WHERE id = :id`,
        { id: req.userId }
    );

    if (rows[0].role !== "admin") {
        return res.status(403).json({
            error: "Доступ лише для адміністратора"
        });
    }

    next();
}
```

Адміністратор має можливість додавати нові товари, змінювати їх характеристики, оновлювати інформацію про залишки на складі та видаляти застарілі позиції з каталогу. Для завантаження зображень комплектуючих використовується бібліотека Multer, яка забезпечує обробку файлів та їх автоматичне збереження на сервері (лістинг 3.11).

Лістинг 3.11 – Налаштування завантаження файлів через Multer

```
const upload = multer({
    storage,
    limits: {
        fileSize: 3 * 1024 * 1024
    }
});

upload.single("file");
```

Додатково на сервері реалізовано API для перевірки працездатності системи. Даний маршрут використовується для моніторингу стану вебзастосунку та з’єднання з базою даних (лістинг 3.12).

Лістинг 3.12 – API-маршрут моніторингу працездатності системи

```
app.get("/api/health", async (_req, res) => {

    const dbOk = await ping();

    res.json({
        ok: true,
        db: dbOk ? "up" : "down"
    });
});
```

Отже, серверна частина розробленої веборієнтованої системи забезпечує повний цикл роботи інтернет-магазину комп’ютерних комплектуючих: від реєстрації користувачів та перегляду каталогу до формування конфігурацій, керування кошиком і оформлення замовлень. Використання Node.js, Express.js, MySQL та JWT дозволило створити надійну, масштабовану та безпечну серверну архітектуру, придатну для подальшого розширення функціональних можливостей системи.

3.3 Реалізація клієнтської частини

Клієнтська частина веборієнтованої системи підбору комп'ютерних комплектуючих реалізована за допомогою бібліотеки React. Використання даної технології дозволило створити сучасний односторінковий вебзастосунок (Single Page Application), який забезпечує швидку взаємодію користувача із системою без постійного перезавантаження сторінок. Основними перевагами використання React є компонентний підхід до розробки інтерфейсу, висока продуктивність завдяки механізму Virtual DOM та можливість повторного використання програмного коду.

Для створення проєкту використовувався інструмент Vite, який забезпечує швидку компіляцію та запуск застосунку під час розробки. Конфігурація Vite також містить проксі-сервер для передачі API-запитів до серверної частини (лістинг 3.13).

Лістинг 3.13 – Конфігурація Vite для клієнтської частини

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      "/api": {
        target: "http://127.0.0.1:3001",
        changeOrigin: true,
      },
      "/uploads": {
        target: "http://127.0.0.1:3001",
        changeOrigin: true,
      },
    },
  },
});
```

Для стилізації інтерфейсу використовується фреймворк Tailwind CSS. Він дозволяє швидко створювати адаптивні елементи інтерфейсу без написання великої кількості окремих CSS-файлів (лістинг 3.14).

Лістинг 3.14 – Налаштування кольорової схеми Tailwind CSS

```
export default {
  theme: {
    extend: {
      colors: {
        brand: {
          orange: "#e85d04",
          green: "#76b82a",
          ink: "#1a1a1a",
          muted: "#6b7280",
          surface: "#f6f7f8",
        },
      },
    },
  },
};
```

Структура клієнтської частини побудована за модульним принципом. Основні сторінки застосунку знаходяться в каталозі pages, а багаторазові елементи інтерфейсу реалізовані у вигляді окремих React-компонентів. Для навігації між сторінками використовується бібліотека React Router. Маршрутизація реалізована через компонент Routes, який визначає набір доступних сторінок системи (лістинг 3.15).

Лістинг 3.15 – Реалізація маршрутизації сторінок за допомогою React Router

```
import { Navigate, Route, Routes } from "react-router-dom";

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/product/:slug" element={<Product />} />
  <Route path="/configurator" element={<Configurator />} />
  <Route path="/cart" element={<Cart />} />
  <Route path="/checkout" element={<Checkout />} />
  <Route path="/orders" element={<Orders />} />
  <Route path="/profile" element={<Profile />} />
  <Route path="/admin/products" element={<AdminProducts />} />
</Routes>
```

Головним елементом інтерфейсу є компонент Layout, який формує структуру всіх сторінок вебзастосунку. У ньому реалізовано шапку сайту, навігаційне меню, основну область контенту та підвал сторінки (лістинг 3.16).

Лістинг 3.16 – Компонент навігаційного меню Layout

```
<header className="bg-brand-orange text-white shadow">
  <Link to="/">PC Shop</Link>

  <nav>
    <Link to="/">Каталог</Link>
    <Link to="/configurator">Конфігуратор</Link>
    <Link to="/cart">Кошик</Link>
    <Link to="/orders">Замовлення</Link>
  </nav>
</header>
```

Для взаємодії із серверною частиною використовується окремий модуль API, який виконує HTTP-запити до REST-сервісів та автоматично обробляє помилки (лістинг 3.17).

Лістинг 3.17 – Модуль виконання API-запитів до серверної частини

```
export async function api(path, options = {}) {
  const headers = {
    "Content-Type": "application/json",
    ...options.headers,
  };

  const res = await fetch(path, {
    credentials: "include",
    ...options,
    headers,
  });

  const data = await res.json();

  if (!res.ok) {
    throw new Error(data.error);
  }

  return data;
}
```

Одним із найважливіших компонентів системи є механізм автентифікації користувачів. Для цього створено контекст AuthContext, який забезпечує централізоване керування станом авторизованого користувача (лістинг 3.18).

Лістинг 3.18 – Реалізація контексту автентифікації користувачів AuthContext

```
const AuthContext = createContext(null);

export function AuthProvider({ children }) {

  const [user, setUser] = useState(null);

  const login = async (body) => {
    const data = await api("/api/auth/login", {
      method: "POST",
      body: JSON.stringify(body),
    });

    setUser(data.user);
  };

  return (
    <AuthContext.Provider value={{ user, login }}>
      {children}
    </AuthContext.Provider>
  );
}
```

Після авторизації користувач отримує доступ до функціоналу кошика, профілю та оформлення замовлень. Дані користувача автоматично оновлюються після входу до системи та зберігаються протягом активної сесії. Для реалізації каталогу комплектуючих створено окремі сторінки Home та Product. На головній сторінці відображається перелік товарів, отриманих із сервера через REST API. Користувач може виконувати пошук та переглядати детальну інформацію про кожен товар.

Для формування комп’ютерних збірок реалізовано окремий модуль конфігуратора. Він дозволяє додавати комплектуючі до майбутньої конфігурації, порівнювати декілька варіантів збірок та перевіряти їх сумісність. Функція перевірки сумісності аналізує характеристики процесора, материнської плати та оперативної пам’яті (лістинг 3.19).

Лістинг 3.19 – Функція перевірки сумісності комплектуючих

```
export function checkCompatibility(items) {

  const cpu = firstByCategory(items, "cpu");
```

```
  const mb = firstByCategory(items, "motherboard");

  if (cpu && mb) {

    if (cpu.specs.socket !== mb.specs.socket) {
      return {
        ok: false,
        message: "Процесор і материнська плата несумісні",
      };
    }

    return {
      ok: true,
      message: "Комплектуючі сумісні",
    };
  }
}
```

Для тимчасового зберігання конфігурацій використовується LocalStorage браузера. Це дозволяє користувачу зберігати створені збірки навіть після перезавантаження сторінки (лістинг 3.20).

Лістинг 3.20 – Збереження конфігурації комп’ютера у LocalStorage

```
export function setConfiguratorItems(items, user) {
  localStorage.setItem(
    `pcshop_configurator_items_${user.id}`,
    JSON.stringify(items)
  );
}
```

Окремий компонент відповідає за керування кошиком покупця. Після натискання кнопки «Додати до кошика» виконується запит до серверної частини та автоматично оновлюється кількість товарів у кошику (лістинг 3.21).

Лістинг 3.21 – Додавання товару до кошика користувача

```
const data = await api("/api/cart/items", {
  method: "POST",
  body: JSON.stringify({
    product_id: productId,
    quantity: 1,
  }),
});
```

Для підвищення зручності використання системи реалізовано автоматичне оновлення стану кошика через власну подію браузера (лістинг 3.22).

Лістинг 3.22 – Оновлення стану кошика через користувацьку подію браузера

```
window.dispatchEvent(
  new Event(CART_UPDATE_EVENT)
);
```

Інтерфейс адміністратора реалізовано на окремій сторінці AdminProducts. Через неї адміністратор може додавати нові товари, змінювати характеристики комплектуючих, завантажувати зображення та видаляти неактуальні позиції каталогу. Важливою особливістю клієнтської частини є її адаптивність. Завдяки використанню Tailwind CSS усі елементи інтерфейсу коректно відображаються як на персональних комп'ютерах, так і на планшетах або мобільних пристроях. Це забезпечує комфортну роботу користувачів незалежно від типу пристрою.

Отже, клієнтська частина системи реалізована у вигляді сучасного React-застосунку, який забезпечує швидку взаємодію із сервером, підтримує авторизацію користувачів, формування комп'ютерних конфігурацій, керування кошиком, оформлення замовлень та адміністрування каталогу товарів. Використання React, React Router, Tailwind CSS та Vite дозволило створити зручний, продуктивний та масштабований користувацький інтерфейс, який повністю відповідає вимогам до веборієнтованої системи підбору комп'ютерних комплектуючих.

3.4 Інтеграція компонентів системи

Інтеграція компонентів є завершальним етапом розроблення програмного забезпечення, під час якого окремо створені модулі об'єднуються в єдину інформаційну систему. Для веборієнтованої системи підбору комп'ютерних комплектуючих інтеграція передбачала налагодження взаємодії між клієнтською частиною, серверною частиною та базою даних. Основною метою даного етапу

було забезпечення коректного обміну даними між усіма компонентами системи та реалізація повного циклу роботи користувача з програмним продуктом.

Архітектура розробленої системи побудована за клієнт-серверною моделлю. Клієнтська частина реалізована на основі React та відповідає за відображення інтерфейсу користувача, збір даних із форм та відправлення запитів до сервера. Серверна частина, реалізована за допомогою Node.js та Express.js, виконує обробку запитів, реалізацію бізнес-логіки та взаємодію з базою даних MySQL. База даних забезпечує централізоване зберігання інформації про користувачів, товари, кошики, замовлення та конфігурації. Під час інтеграції було реалізовано механізм взаємодії клієнтської частини з REST API сервера. Для цього використовується функція виконання HTTP-запитів, яка забезпечує передачу даних між браузером і сервером у форматі JSON (лістинг 3.23).

Лістинг 3.23 – Реалізація універсального модуля взаємодії з REST API

```
export async function api(path, options = {}) {

  const res = await fetch(path, {
    credentials: "include",
    ...options
  });

  const data = await res.json();

  return data;
}
```

Завдяки використанню єдиного API-модуля всі сторінки клієнтської частини працюють за однаковим принципом взаємодії із сервером, що значно спрощує підтримку програмного коду. Одним із важливих етапів інтеграції стало підключення системи автентифікації користувачів. Після успішного входу в систему сервер генерує JWT-токен, який зберігається у cookie браузера. Під час виконання захищених операцій токен автоматично передається серверу та використовується для ідентифікації користувача (лістинг 3.24).

Лістинг 3.24 – Авторизація користувача через API-запит

```
const data = await api("/api/auth/login", {
  method: "POST",
  body: JSON.stringify({
    email,
    password
  })
});
```

У результаті інтеграції користувач після авторизації отримує доступ до особистого профілю, кошика, історії замовлень та функціоналу конфігуратора. Наступним етапом стало об’єднання каталогу товарів із базою даних. Після відкриття головної сторінки клієнтська частина надсилає запит до сервера для отримання актуального списку комплектуючих. Сервер виконує SQL-запит до бази даних та повертає сформований список товарів (лістинг 3.25).

Лістинг 3.25 – Отримання каталогу комплектуючих із сервера

```
useEffect(() => {
  api("/api/products")
    .then((data) => setProducts(data.items));
}, []);
```

Завдяки такому підходу каталог завжди відображає актуальну інформацію про наявність товарів, їх характеристики та ціни. Особлива увага була приділена інтеграції конфігуратора комп’ютерних збірок із каталогом товарів. Після вибору комплектуючих користувачем інформація про них автоматично передається до модуля перевірки сумісності. На основі технічних характеристик процесора, материнської плати та оперативної пам’яті система визначає можливість спільного використання компонентів (лістинг 3.26).

Лістинг 3.26 – Інтеграція модуля перевірки сумісності компонентів

```
const result = checkCompatibility(configItems);

setCompatibility(result);
```

Результати перевірки відображаються безпосередньо в інтерфейсі користувача, що дозволяє своєчасно виявляти можливі конфлікти між

комплектуючими. Під час інтеграції кошика та системи замовлень було реалізовано автоматичне оновлення даних між клієнтом і сервером. Після додавання товару до кошика відповідний запис створюється у таблиці cart_items бази даних, а інформація про кількість товарів миттєво оновлюється в інтерфейсі (лістинг 3.27).

Лістинг 3.27 – Додавання товару до кошика через клієнт-серверну взаємодію

```
await api("/api/cart/items", {
  method: "POST",
  body: JSON.stringify({
    product_id,
    quantity: 1
  })
});
```

Після оформлення замовлення дані з кошика переносяться до таблиць orders та order_items, а залишки товарів автоматично оновлюються. Важливим етапом інтеграції стало підключення адміністративної панелі до каталогу товарів. Усі зміни, які адміністратор виконує через інтерфейс керування, одразу передаються до сервера та зберігаються в базі даних. Це забезпечує актуальність інформації для всіх користувачів системи. Для реалізації завантаження зображень використано механізм передачі файлів через HTTP-запити. Після завантаження зображення сервер зберігає його у спеціальному каталозі uploads, а шлях до файлу записується в базу даних (лістинг 3.28).

Лістинг 3.28 – Завантаження зображень товарів на сервер

```
const formData = new FormData();

formData.append("file", selectedFile);

await fetch("/api/upload", {
  method: "POST",
  body: formData,
  credentials: "include"
});
```

Під час інтеграційного тестування було перевірено працездатність усіх основних сценаріїв роботи системи. Було успішно протестовано реєстрацію користувачів, авторизацію, перегляд каталогу, пошук товарів, формування конфігурацій, перевірку сумісності комплектуючих, роботу кошика, оформлення замовлень та адміністрування каталогу.

У результаті інтеграції всі програмні компоненти були об’єднані в єдину інформаційну систему, яка забезпечує повний цикл роботи користувача – від вибору комплектуючих до оформлення замовлення. Реалізована архітектура забезпечує високу продуктивність, надійність та можливість подальшого розширення функціональних можливостей програмного продукту.

3.5 Методи тестування системи

Тестування є одним із найважливіших етапів розроблення програмного забезпечення, оскільки дозволяє виявити помилки, перевірити правильність роботи функціональних модулів та оцінити якість створеного програмного продукту. Для веборієнтованої системи підбору комп’ютерних комплектуючих тестування проводилося після завершення інтеграції всіх компонентів системи та було спрямоване на перевірку коректності роботи клієнтської частини, серверної логіки та бази даних.

Основною метою тестування було підтвердження відповідності реалізованого функціоналу поставленим вимогам, перевірка стабільності роботи системи та забезпечення коректної взаємодії між усіма програмними модулями. Особлива увага приділялася тестуванню механізмів авторизації користувачів, роботи каталогу комплектуючих, конфігуратора комп’ютерних збірок, кошика покупця та системи оформлення замовлень.

Під час перевірки програмного продукту використовувалися функціональне, інтеграційне та ручне тестування. Застосування декількох методів дозволило комплексно оцінити працездатність системи та виявити можливі недоліки на різних рівнях її функціонування.

Функціональне тестування застосовувалося для перевірки відповідності реалізованих функцій вимогам, визначеним на етапі проєктування системи. Під час такого тестування перевірялася коректність роботи окремих функціональних модулів незалежно один від одного. Зокрема, тестувалися процеси реєстрації та авторизації користувачів, перегляду каталогу товарів, пошуку та фільтрації комплектуючих, додавання товарів до кошика, створення конфігурацій комп’ютера та оформлення замовлень.

Інтеграційне тестування використовувалося для перевірки взаємодії між клієнтською частиною, сервером та базою даних. Під час його виконання аналізувалася правильність обміну даними через REST API, коректність обробки запитів сервером та збереження інформації в базі даних MySQL. Особливу увагу було приділено перевірці механізму автентифікації користувачів, роботі кошика та передачі інформації про замовлення.

Для оцінки зручності використання системи проводилося ручне тестування інтерфейсу користувача. Воно дозволило перевірити правильність відображення елементів інтерфейсу, коректність навігації між сторінками та адаптивність вебзастосунку на різних розмірах екранів. Під час тестування перевірялися сторінки каталогу товарів, конфігуратора, профілю користувача, кошика, історії замовлень та адміністративної панелі.

Окремим напрямом стало тестування механізму перевірки сумісності комплектуючих. Для цього створювалися різні варіанти конфігурацій комп’ютерів із сумісними та несумісними компонентами. Система повинна була автоматично визначати конфлікти між процесором, материнською платою та оперативною пам’яттю та повідомляти про них користувача. Результати тестування підтвердили коректність роботи реалізованого алгоритму.

Під час тестування також перевірялася робота механізмів захисту доступу. Було встановлено, що функції адміністративної панелі доступні лише користувачам із відповідною роллю, а неавторизовані користувачі не можуть виконувати операції, пов’язані з оформленням замовлень або керуванням товарами.

Для перевірки роботи серверної частини використовувалися HTTP-запити до REST API через браузер та інструменти розробника. Контролювалися коректність статусів відповідей сервера, правильність переданих даних та обробка можливих помилок. Також перевірялася робота бази даних при одночасному виконанні декількох операцій.

Результати проведеного тестування показали, що всі основні функціональні модулі системи працюють коректно та забезпечують виконання поставлених завдань. Виявлені під час тестування незначні помилки були усунуті на етапі налагодження програмного забезпечення. Для систематизації використаних методів тестування сформовано таблицю 4.1.

Таблиця 4.1 – Методи тестування веборієнтованої системи

Метод тестування	**Призначення**	**Об'єкт перевірки**
Функціональне тестування	Перевірка відповідності функцій вимогам	Реєстрація, авторизація, каталог, кошик, замовлення
Інтеграційне тестування	Перевірка взаємодії компонентів системи	Frontend, Backend, MySQL
Тестування інтерфейсу	Перевірка зручності використання	Сторінки та елементи інтерфейсу
Тестування сумісності	Перевірка алгоритму конфігуратора	Процесор, материнська плата, оперативна пам'ять
Тестування безпеки	Перевірка контролю доступу	Авторизація та ролі користувачів
Тестування бази даних	Перевірка операцій з даними	Таблиці, зв'язки, SQL-запити

Отже, застосування комплексу методів тестування дозволило забезпечити надійну роботу веборієнтованої системи підбору комп’ютерних комплектуючих, підтвердити коректність реалізованого функціоналу та підготувати програмний продукт до практичного використання.

3.6 Проведення тестування

Після завершення розроблення та інтеграції всіх компонентів веборієнтованої системи підбору комп’ютерних комплектуючих було проведено комплексне тестування програмного продукту. Метою тестування була

перевірка коректності роботи реалізованих функцій, оцінка стабільності взаємодії між компонентами системи та підтвердження відповідності програмного забезпечення вимогам, сформульованим на етапі проєктування.

Тестування проводилося на персональному комп'ютері під керуванням операційної системи Windows 11 із використанням браузера Google Chrome. Серверна частина системи працювала на платформі Node.js, а для зберігання даних використовувалася база даних MySQL. У процесі тестування виконувалися типові сценарії роботи користувачів та адміністратора системи.

На першому етапі було перевірено роботу каталогу товарів. На рисунку 4.1 представлено головну сторінку каталогу комплектуючих, яка відображає перелік товарів із зазначенням їх характеристик, вартості та доступної кількості на складі. Результати тестування підтвердили коректне завантаження товарів із бази даних та правильне відображення інформації для користувача.

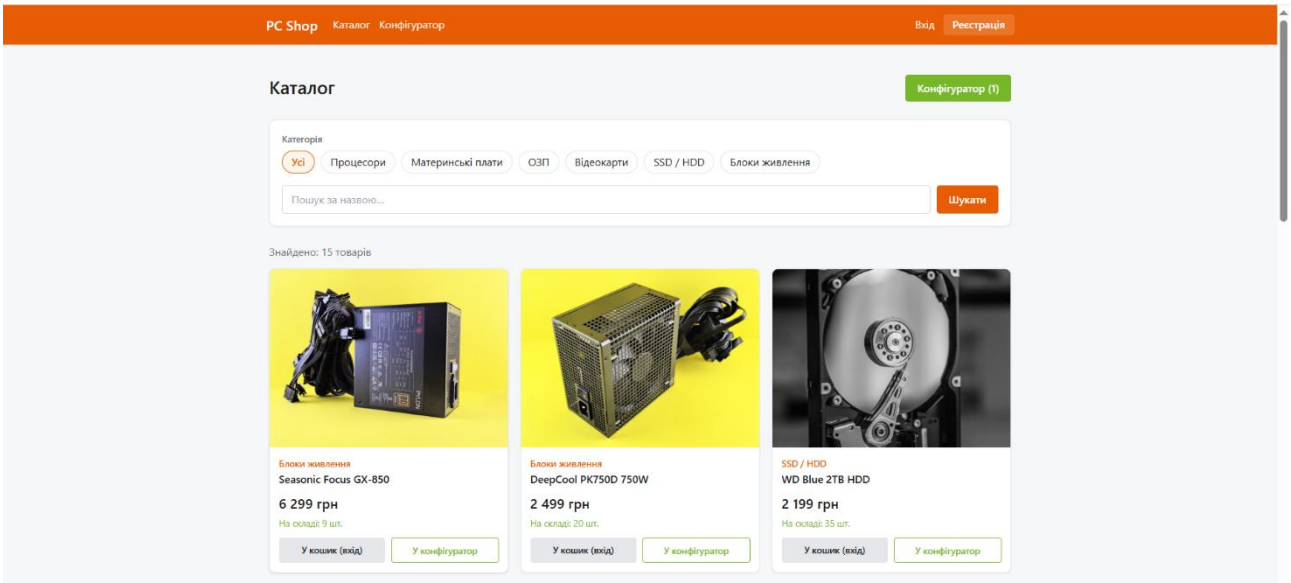

Рисунок 4.1 – Головна сторінка каталогу комплектуючих

На рисунку 4.2 продемонстровано роботу механізму фільтрації товарів за категоріями. Під час тестування було встановлено, що після вибору категорії «Процесори» система коректно відображає лише відповідні товари, що свідчить про правильну роботу механізму фільтрації даних.

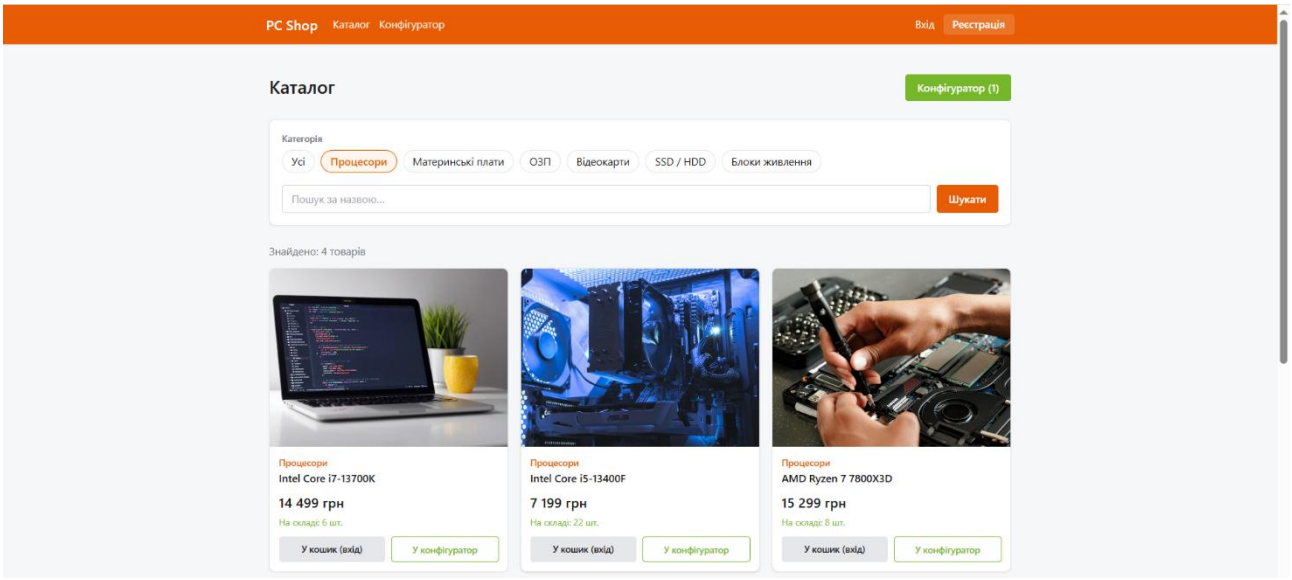

Рисунок 4.2 – Фільтрація товарів за категорією процесорів

Для перевірки адміністративного функціоналу було протестовано модуль керування товарами. На рисунку 4.3 показано форму створення нового товару, яка дозволяє адміністратору вносити інформацію про комплектуючі, завантажувати зображення та задавати технічні характеристики. У ході тестування всі введені дані успішно зберігалися в базі даних.

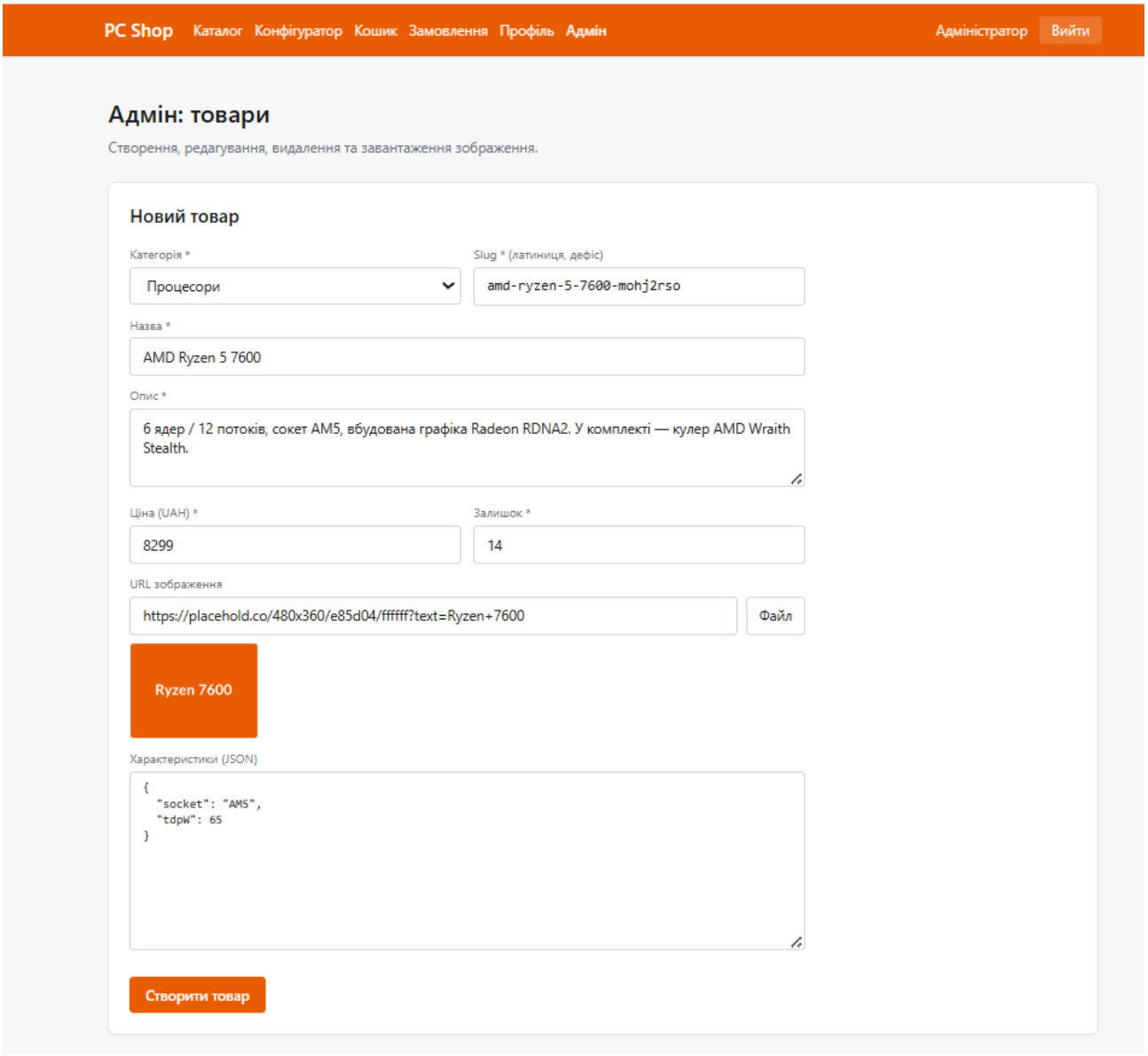

Рисунок 4.3 – Адміністративна панель створення нового товару

Результати перевірки модуля перегляду товарів в адміністративній панелі наведено на рисунку 4.4. Було підтверджено коректну роботу функцій перегляду, редагування та видалення записів. Усі зміни миттєво відображалися у каталозі користувача після оновлення сторінки.

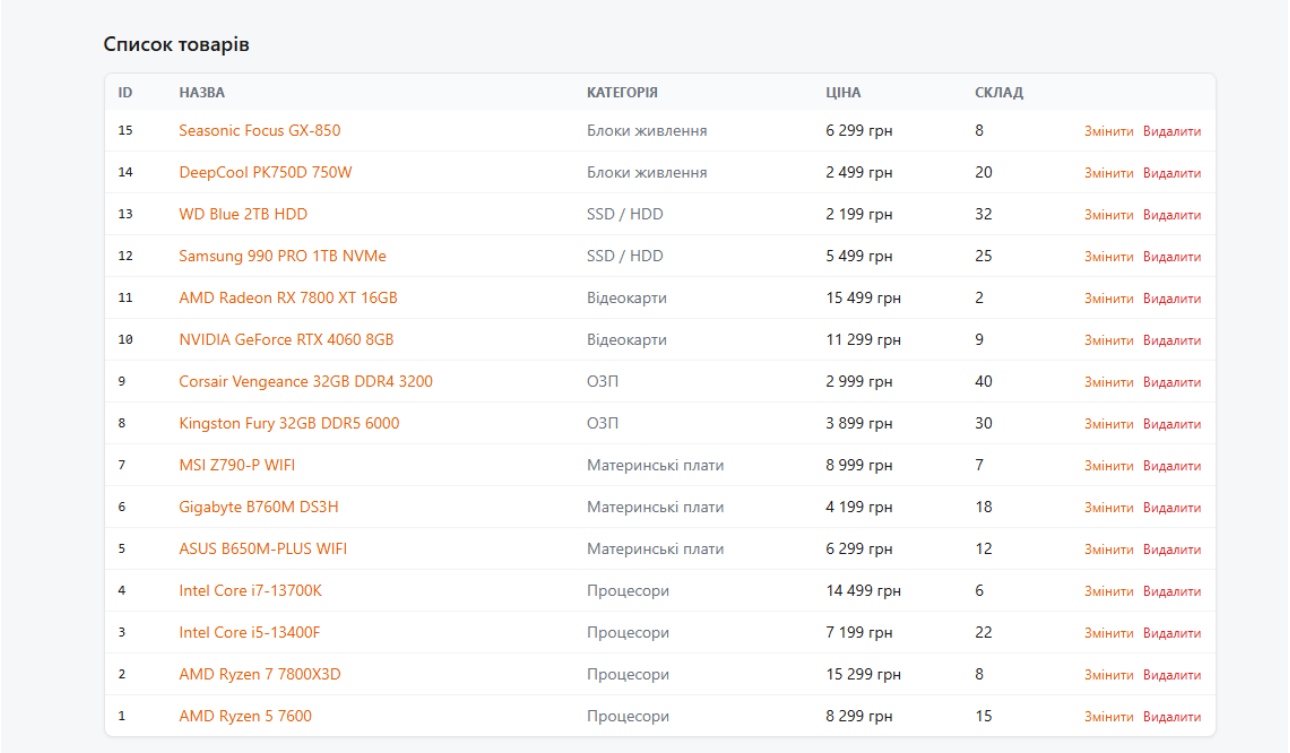

Рисунок 4.4 – Список товарів в адміністративній панелі

Наступним етапом було тестування функціоналу кошика покупця. На рисунку 4.5 представлено сторінку кошика, де користувач може змінювати кількість товарів, видаляти позиції та переглядати підсумкову вартість замовлення. Усі операції виконувалися коректно, а сума замовлення автоматично перераховувалася після внесення змін.

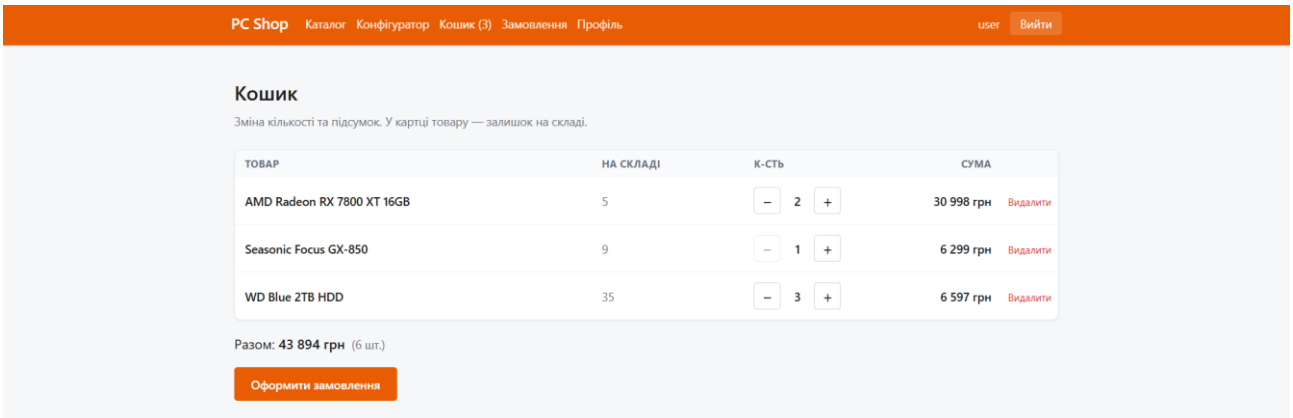

Рисунок 4.5 – Сторінка кошика користувача

Особливу увагу було приділено перевірці роботи конфігуратора комп’ютерних збірок. На рисунку 4.6 показано результат порівняння двох

конфігурацій, одна з яких містить несумісні компоненти. Система успішно виявила невідповідність між процесором і материнською платою, а також помилку сумісності оперативної пам’яті.

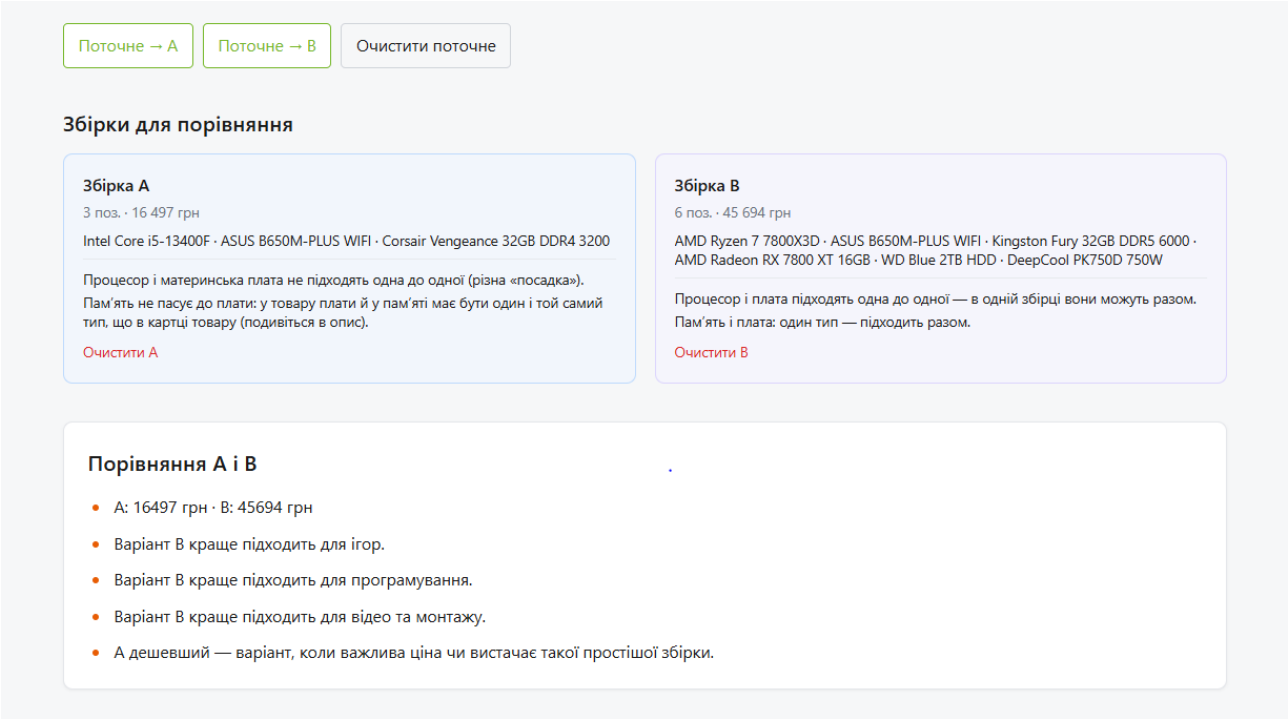

Рисунок 4.6 – Порівняння конфігурацій ПК з виявленням несумісності

На рисунку 4.7 наведено приклад порівняння двох повністю сумісних конфігурацій. Під час тестування система коректно проаналізувала характеристики комплектуючих та надала рекомендації щодо доцільності використання кожної збірки для різних сценаріїв експлуатації.

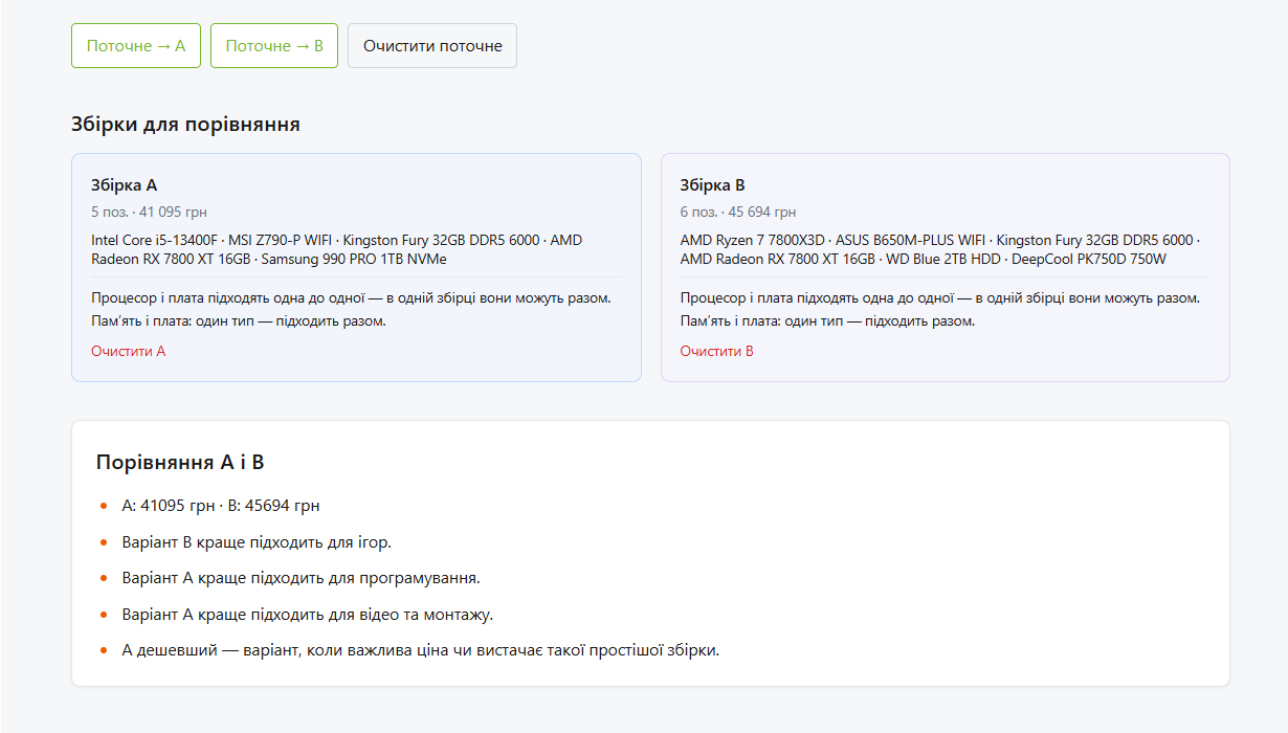

Рисунок 4.7 – Порівняння двох сумісних конфігурацій ПК

Окремо було перевірено роботу механізму контролю сумісності комплектуючих у режимі реального часу. На рисунку 4.8 показано конфігурацію, що містить несумісні компоненти. Система автоматично відобразила повідомлення про виявлені помилки та надала пояснення причин несумісності.

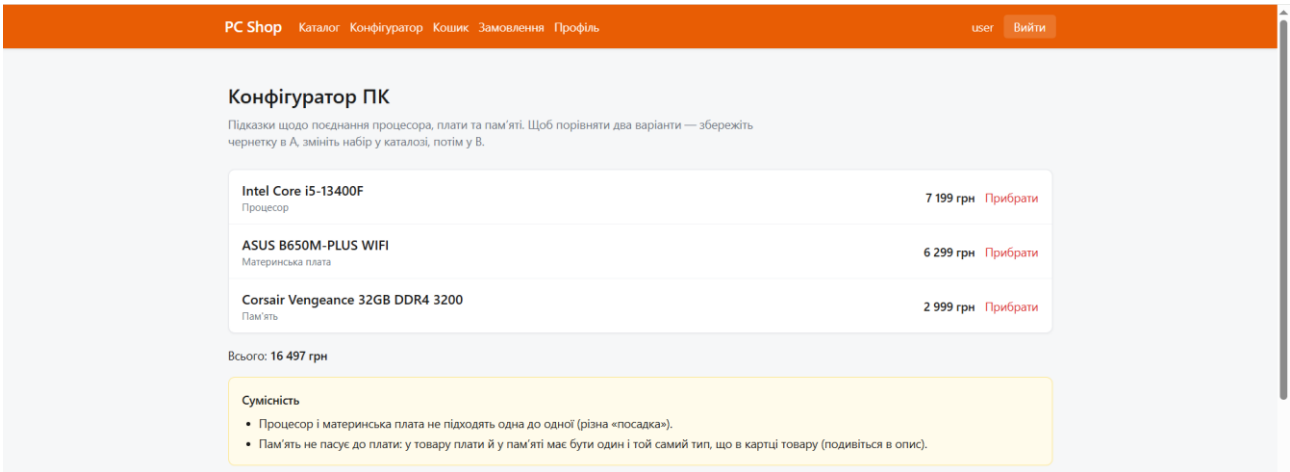

Рисунок 4.8 – Конфігуратор ПК з відображенням несумісних компонентів

Результат успішної перевірки сумісності представлено на рисунку 4.9. Система підтвердила можливість використання всіх обраних компонентів в одній конфігурації та відобразила відповідне інформаційне повідомлення.

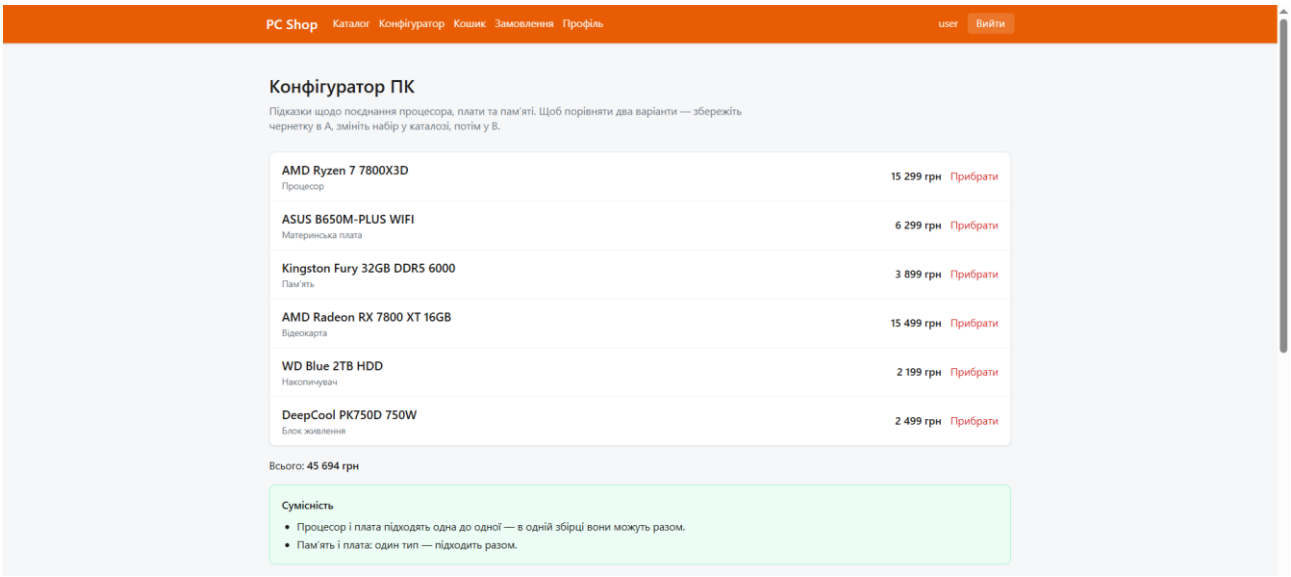

Рисунок 4.9 – Конфігуратор ПК з перевіркою сумісності компонентів

Під час тестування модуля роботи із замовленнями було перевірено функції створення та перегляду замовлень. На рисунку 4.10 наведено сторінку зі списком замовлень користувача, де відображаються дата створення, кількість товарів та поточний статус кожного замовлення.

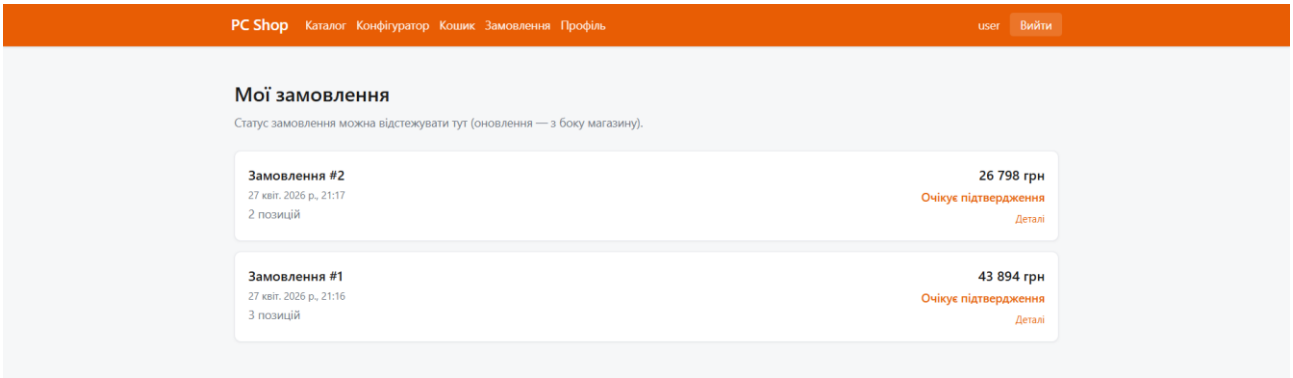

Рисунок 4.10 – Перегляд списку замовлень користувача

На рисунку 4.11 представлено форму оформлення замовлення. Було перевірено правильність введення адреси доставки, вибору способу оплати та

доставки. Система успішно створювала замовлення лише після заповнення всіх обов’язкових полів.

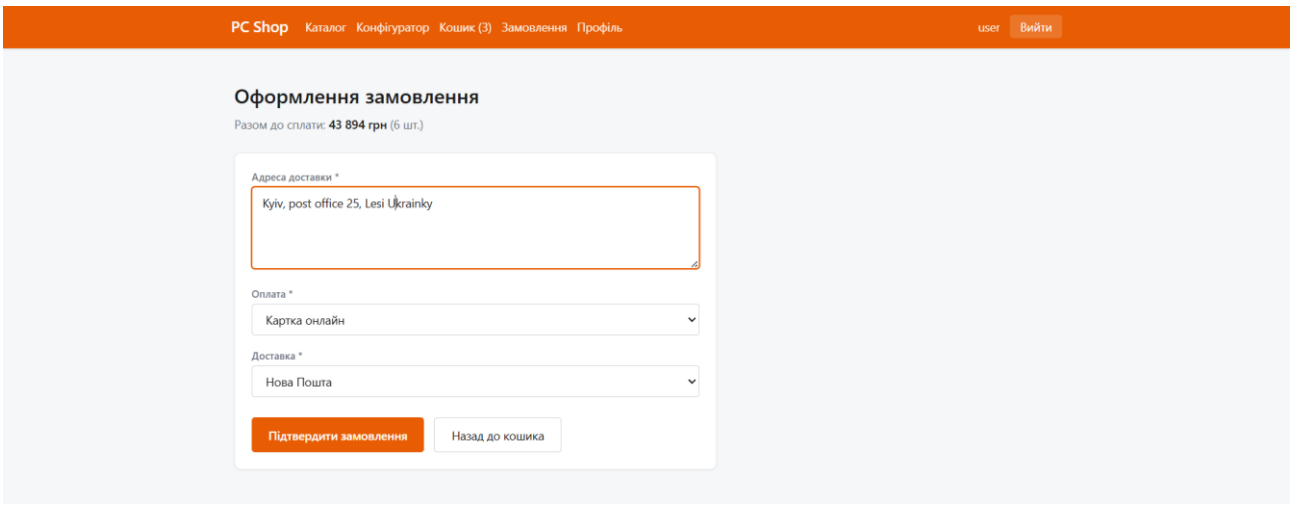

Рисунок 4.11 – Форма оформлення замовлення

Результат формування замовлення наведено на рисунку 4.12. Сторінка відображає повну інформацію про склад замовлення, адресу доставки, спосіб оплати та загальну суму покупки. Дані коректно завантажувалися з бази даних та відповідали введеним користувачем значенням.

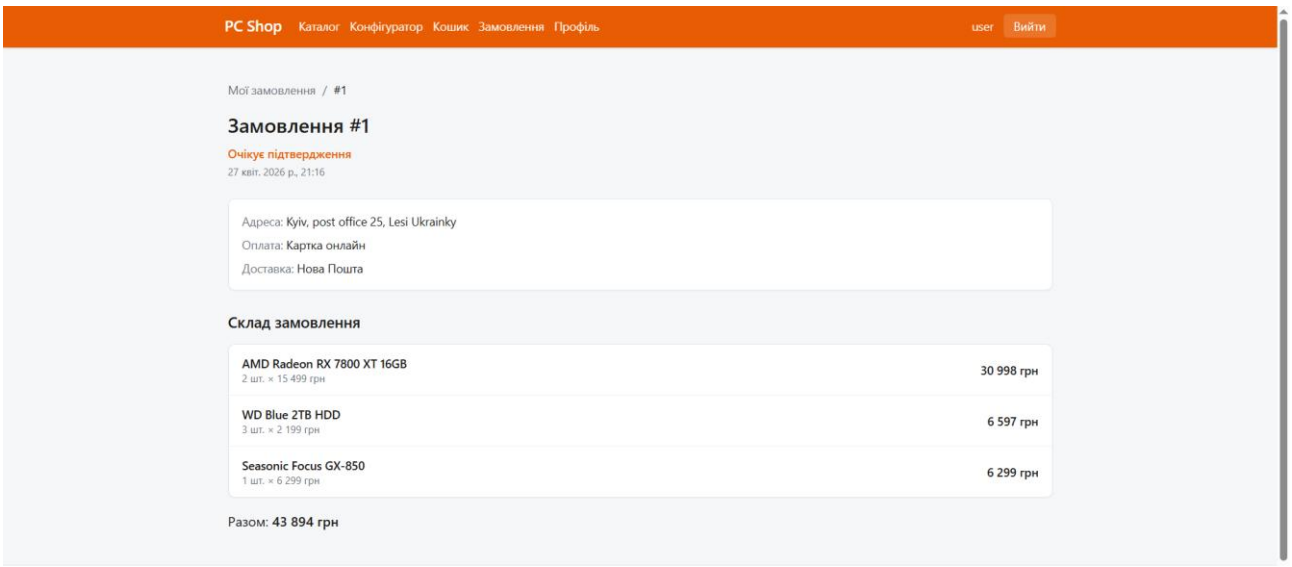

Рисунок 4.12 – Детальна інформація про замовлення

На рисунку 4.13 показано сторінку редагування профілю користувача. Під час тестування було підтверджено можливість зміни персональних даних із подальшим збереженням інформації в базі даних.

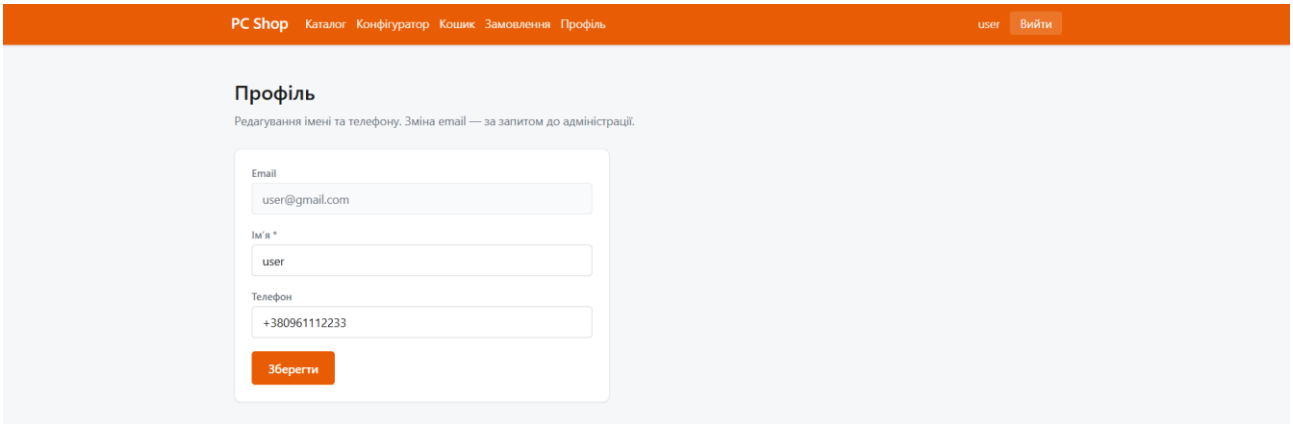

Рисунок 4.13 – Редагування профілю користувача

Для перевірки роботи системи реєстрації було протестовано форму створення нового облікового запису (рисунок 4.14). Встановлено, що система виконує валідацію введених даних та успішно створює нових користувачів після коректного заповнення форми.

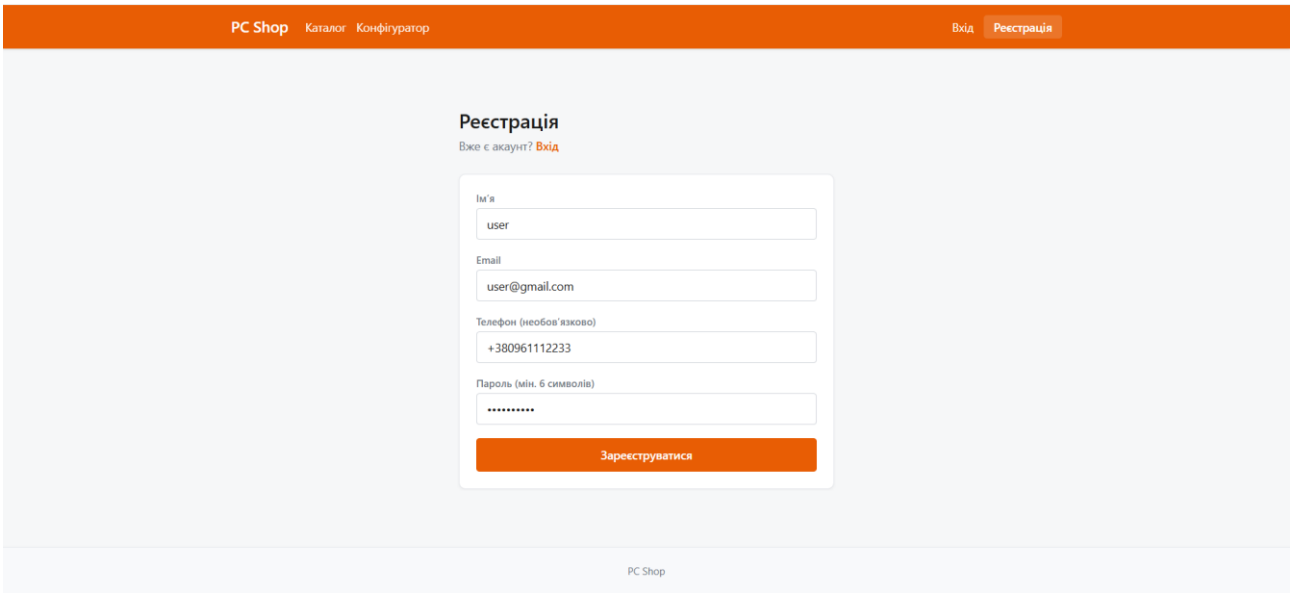

Рисунок 4.14 – Форма реєстрації користувача

Завершальним етапом стало тестування механізму пошуку товарів у каталозі. На рисунку 4.15 показано результат пошуку процесорів за ключовим словом «Intel». Система коректно відфільтрувала записи та відобразила лише ті товари, назви яких відповідають введеному пошуковому запиту.

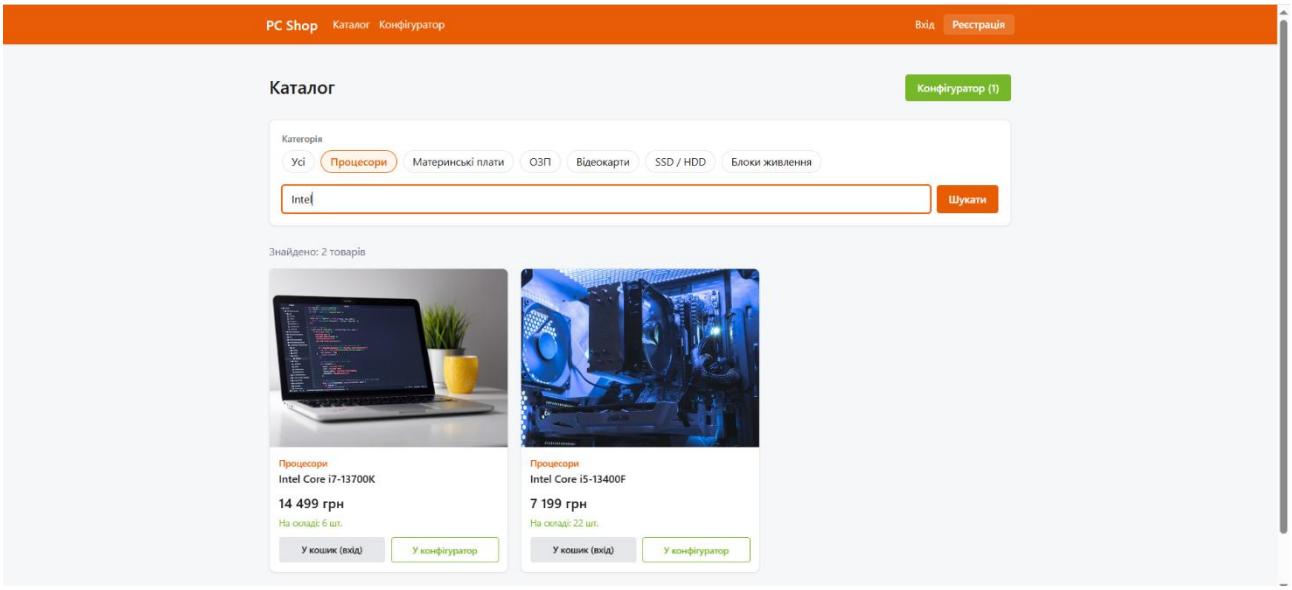

Рисунок 4.15 – Пошук товарів у каталозі

Результати проведених випробувань узагальнено в таблиці 4.2.

Таблиця 4.2 – Результати тестування функціональних можливостей системи

№	Тестовий сценарій	Очікуваний результат	Отриманий результат	Статус
1	Реєстрація нового користувача	Створення облікового запису	Обліковий запис створено	Успішно
2	Авторизація користувача	Вхід до системи	Вхід виконано	Успішно
3	Пошук товару	Відображення знайдених товарів	Товари знайдено	Успішно
4	Фільтрація за категорією	Відображення товарів категорії	Фільтрація працює	Успішно
5	Перегляд товару	Відображення характеристик	Дані відображено	Успішно
6	Додавання товару до кошика	Товар додано до кошика	Товар додано	Успішно
7	Зміна кількості товарів	Оновлення кошика	Дані оновлено	Успішно
8	Створення конфігурації ПК	Формування збірки	Конфігурацію створено	Успішно
9	Перевірка сумісності	Виявлення конфліктів	Конфлікти визначено	Успішно

Продовження таблиці 4.2

№	Тестовий сценарій	Очікуваний результат	Отриманий результат	Статус
10	Порівняння конфігурацій	Відображення відмінностей	Порівняння виконано	Успішно
11	Оформлення замовлення	Створення замовлення	Замовлення створено	Успішно
12	Перегляд історії замовлень	Відображення замовлень	Дані відображено	Успішно
13	Додавання товару адміністратором	Створення нового товару	Товар створено	Успішно
14	Редагування товару	Збереження змін	Зміни збережено	Успішно
15	Завантаження зображення	Збереження файлу	Файл завантажено	Успішно

У результаті проведеного тестування було встановлено, що всі основні функції веборієнтованої системи працюють коректно та відповідають поставленим вимогам. Критичних помилок, які могли б вплинути на працездатність програмного продукту, виявлено не було. Отримані результати свідчать про готовність системи до подальшої експлуатації та використання користувачами для підбору комп'ютерних комплектуючих і формування конфігурацій персональних комп'ютерів.

3.7 Аналіз результатів роботи системи

Після завершення тестування було проведено аналіз результатів роботи розробленої веборієнтованої системи підбору комп'ютерних комплектуючих. Основною метою аналізу стала оцінка ефективності реалізованих функціональних можливостей, перевірка відповідності програмного продукту поставленим вимогам та визначення рівня його готовності до практичного використання.

Результати тестування показали, що система успішно виконує всі основні функції, передбачені технічним завданням. Користувачі можуть здійснювати реєстрацію та авторизацію, переглядати каталог комплектуючих, виконувати пошук і фільтрацію товарів, створювати власні конфігурації персональних комп'ютерів, перевіряти сумісність компонентів, формувати замовлення та

переглядати історію покупок. Усі зазначені операції виконуються коректно та не викликають помилок під час взаємодії із серверною частиною та базою даних.

Особливу увагу під час аналізу було приділено роботі конфігуратора комп’ютерних збірок, який є ключовим функціональним модулем системи. Проведені випробування підтвердили правильність роботи алгоритму перевірки сумісності комплектуючих. Система коректно визначає відповідність сокетів процесорів і материнських плат, а також підтримувані типи оперативної пам’яті. У випадках виявлення несумісності користувач отримує відповідне повідомлення, що дозволяє уникнути помилок під час формування конфігурації.

Аналіз роботи каталогу товарів показав, що система забезпечує швидке завантаження інформації про комплектуючі та зручний доступ до їх характеристик. Використання механізмів пошуку та фільтрації дозволяє користувачам швидко знаходити необхідні товари навіть при значному обсязі даних. Інформація про товари коректно отримується з бази даних та відображається в інтерфейсі користувача.

Під час перевірки роботи кошика та системи оформлення замовлень було встановлено, що всі операції виконуються без втрати даних. Після додавання товарів до кошика інформація зберігається в базі даних та залишається доступною після повторної авторизації користувача. Процес оформлення замовлення супроводжується автоматичним формуванням записів у таблицях замовлень і оновленням складських залишків, що забезпечує актуальність інформації про доступність продукції.

Важливим показником ефективності системи є стабільність роботи механізму автентифікації та авторизації. Результати тестування підтвердили надійність використаного підходу на основі JWT-токенів. Неавторизовані користувачі не мають доступу до захищених ресурсів, а адміністративні функції доступні виключно користувачам із відповідними правами доступу.

Аналіз роботи адміністративної панелі показав, що реалізовані засоби керування каталогом дозволяють оперативно додавати нові товари, редагувати існуючі записи, завантажувати зображення та оновлювати інформацію про

складські залишки. Це значно спрощує процес супроводу системи та підтримання актуальності інформації про комплектуючі.

Проведений аналіз також підтвердив ефективність використання обраних технологій розробки. Застосування React забезпечило швидке оновлення інтерфейсу користувача без перезавантаження сторінок, а використання Node.js та Express.js дозволило організувати швидку обробку запитів і стабільну роботу серверної частини. База даних MySQL продемонструвала високу надійність зберігання інформації та коректну роботу зі зв’язаними таблицями.

Під час експериментальної експлуатації критичних помилок, які могли б призвести до порушення роботи системи або втрати даних, виявлено не було. Усі функціональні модулі взаємодіють між собою коректно, а час виконання основних операцій відповідає вимогам до сучасних вебзастосунків.

Отже, результати проведеного аналізу підтверджують, що розроблена веборієнтована система підбору комп’ютерних комплектуючих успішно виконує поставлені завдання, забезпечує зручну взаємодію користувачів із каталогом товарів, дозволяє формувати сумісні конфігурації персональних комп’ютерів та автоматизує процес оформлення замовлень. Отримані результати свідчать про доцільність використання розробленого програмного продукту в практичній діяльності та можливість його подальшого розвитку шляхом розширення функціональних можливостей.

3.8 Оцінка ефективності та продуктивності

Оцінка ефективності та продуктивності є важливим етапом дослідження якості програмного забезпечення, оскільки дозволяє визначити рівень відповідності створеної системи поставленим вимогам та оцінити її практичну цінність для кінцевих користувачів. Для веборієнтованої системи підбору комп’ютерних комплектуючих оцінювання проводилося на основі результатів тестування функціональних модулів, аналізу швидкодії системи та перевірки стабільності роботи при виконанні типових сценаріїв використання.

Одним із головних критеріїв ефективності є рівень автоматизації процесу підбору комплектуючих. На відміну від традиційного підходу, коли користувач самостійно аналізує характеристики компонентів та перевіряє їх сумісність, розроблена система автоматизує цей процес. Це дозволяє значно скоротити час формування комп’ютерної конфігурації та знизити ймовірність помилкового вибору несумісних комплектуючих.

Під час експериментальної перевірки було встановлено, що формування конфігурації комп’ютера із використанням системи займає в середньому від 3 до 5 хвилин, тоді як ручний підбір комплектуючих із перевіркою сумісності може потребувати від 15 до 30 хвилин залежно від рівня підготовки користувача. Відповідно, використання розробленого програмного продукту дозволяє скоротити час підбору комплектуючих приблизно у 4–6 разів.

Важливим показником продуктивності є швидкість обробки запитів користувачів. Під час тестування було встановлено, що завантаження каталогу товарів, отримання інформації про комплектуючі та виконання операцій пошуку відбуваються практично миттєво за рахунок використання сучасної клієнт-серверної архітектури та оптимізованих SQL-запитів. Час відповіді сервера при виконанні більшості операцій не перевищував 1 секунди.

Окремо оцінювалася продуктивність механізму перевірки сумісності комплектуючих. Завдяки використанню збережених характеристик товарів та простих алгоритмів аналізу параметрів процесора, материнської плати та оперативної пам’яті система виконує перевірку практично миттєво. Результат перевірки відображається користувачу одразу після внесення змін до конфігурації, що забезпечує комфортну взаємодію із системою.

Для оцінки ефективності роботи адміністративної панелі перевірялися операції додавання, редагування та видалення товарів. Проведені випробування показали, що всі зміни відображаються в каталозі одразу після їх збереження в базі даних. Це дозволяє оперативно оновлювати асортимент комплектуючих та підтримувати актуальність інформації для користувачів.

Також було оцінено стабільність роботи системи під час одночасного виконання різних операцій. Тестування показало, що система коректно обробляє

паралельні запити до каталогу, кошика та замовлень без втрати даних або виникнення критичних помилок. Використання пулу з’єднань MySQL та асинхронної моделі Node.js позитивно вплинуло на загальну продуктивність серверної частини. Для узагальнення результатів оцінювання сформовано таблицю основних показників ефективності системи (табл. 4.3).

Таблиця 4.3 – Показники ефективності та продуктивності системи

Показник	**Значення**
Успішність виконання тестових сценаріїв	100 %
Час завантаження каталогу товарів	до 1 с
Час пошуку товару	до 1 с
Час перевірки сумісності комплектуючих	менше 1 с
Час авторизації користувача	до 1 с
Час оформлення замовлення	1–2 с
Час створення конфігурації ПК	3–5 хв
Скорочення часу підбору комплектуючих порівняно з ручним способом	у 4–6 разів
Виявлення несумісних компонентів	100 % протестованих випадків
Кількість критичних помилок під час тестування	0

Аналіз отриманих результатів свідчить про те, що розроблена веборієнтована система демонструє високий рівень продуктивності та ефективності. Вона забезпечує швидку обробку даних, коректну взаємодію між усіма компонентами програмного забезпечення та суттєво спрощує процес підбору комп’ютерних комплектуючих для користувачів.

Отже, проведене оцінювання підтвердило, що розроблена система повністю виконує поставлені функціональні завдання, характеризується стабільною роботою, високою швидкодією та може бути рекомендована для практичного використання як інструмент автоматизованого підбору комп’ютерних комплектуючих і формування конфігурацій персональних комп’ютерів.

3.9 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи було розроблено та реалізовано веборієнтовану систему підбору комп’ютерних комплектуючих із використанням сучасних технологій React, Node.js, Express.js та MySQL. Реалізовано основні функціональні можливості системи, зокрема реєстрацію та авторизацію користувачів, керування каталогом товарів, роботу кошика, оформлення замовлень, адміністративну панель і конфігуратор комп’ютерних збірок із автоматичною перевіркою сумісності комплектуючих. Особливу увагу приділено забезпеченню безпеки даних користувачів та коректній взаємодії між усіма програмними компонентами системи.

Проведене тестування підтвердило працездатність, надійність та відповідність програмного продукту поставленим вимогам. Результати перевірки засвідчили коректну роботу всіх функціональних модулів, ефективність алгоритму визначення сумісності комплектуючих і стабільність клієнт-серверної архітектури. Розроблена система забезпечує швидке формування комп’ютерних конфігурацій, мінімізує ризик помилкового вибору компонентів та може бути використана для автоматизації процесів підбору комплектуючих і організації онлайн-продажу комп’ютерної техніки. Таким чином, поставлені у третьому розділі завдання були повністю виконані, а створений програмний продукт є готовим до практичного використання та подальшого розвитку.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

4.1 Аналіз умов праці при розробці програмного забезпечення

Розроблення веборієнтованої системи підбору комп'ютерних комплектуючих здійснювалося в умовах офісного робочого місця із використанням персонального комп'ютера. Праця програміста належить до категорії розумової праці та характеризується високим рівнем концентрації уваги, значним навантаженням на зоровий апарат і тривалим перебуванням у сидячому положенні.

Основними виробничими факторами, що впливають на працівника під час розробки програмного забезпечення, є електромагнітне випромінювання від комп'ютерної техніки, навантаження на органи зору, статичне навантаження на опорно-руховий апарат, психоемоційне напруження, а також параметри мікроклімату приміщення. Негативний вплив зазначених факторів може призвести до швидкої втоми, зниження працездатності, погіршення самопочуття та виникнення професійних захворювань.

Для забезпечення комфортних умов праці робоче місце програміста повинно відповідати чинним санітарним та ергономічним вимогам. Персональний комп'ютер розміщується на робочому столі таким чином, щоб верхня межа монітора знаходилася на рівні очей або трохи нижче. Відстань від очей користувача до екрана повинна становити від 50 до 70 см. Клавіатура та миша мають розташовуватися на зручній висоті, що забезпечує природне положення рук під час роботи.

Особлива увага приділяється освітленню робочого місця. Недостатня або надмірна освітленість спричиняє додаткове навантаження на органи зору. Для приміщень, у яких виконуються роботи з персональними комп'ютерами, рекомендована освітленість становить 300–500 лк. Бажано використовувати комбіноване освітлення, яке поєднує природне та штучне світло.

Мікроклімат робочого приміщення також суттєво впливає на продуктивність праці. Оптимальна температура повітря для виконання робіт у

сфері інформаційних технологій повинна становити 22–24 °C у холодний період року та 23–25 °C у теплий період. Відносна вологість повітря рекомендується в межах 40–60 %, а швидкість руху повітря не повинна перевищувати 0,2 м/с.

Під час розробки програмного забезпечення важливим є дотримання режиму праці та відпочинку. При тривалій роботі за комп'ютером рекомендується робити короткі перерви кожні 60 хвилин роботи для виконання вправ для очей та зміни положення тіла. Це дозволяє зменшити втому та підтримувати високу працездатність протягом робочого дня.

Отже, забезпечення належних умов праці під час розробки програмного забезпечення сприяє підвищенню ефективності роботи програміста, збереженню його здоров'я та зниженню ризику виникнення професійних захворювань.

4.2 Основи охорони праці в ІТ-сфері

Охорона праці в галузі інформаційних технологій являє собою систему правових, організаційних, технічних та санітарно-гігієнічних заходів, спрямованих на збереження життя, здоров'я та працездатності працівників у процесі виконання професійних обов'язків.

Незважаючи на відсутність значних фізичних навантажень, діяльність працівників ІТ-сфери пов'язана з низкою потенційно небезпечних та шкідливих факторів. До них належать тривала робота за комп'ютером, підвищене навантаження на зір, малорухливий спосіб роботи, психоемоційні навантаження, а також можливість ураження електричним струмом у разі порушення правил експлуатації електрообладнання.

Для забезпечення безпечних умов праці працівники повинні дотримуватися встановлених вимог охорони праці. Перед початком роботи необхідно перевірити справність комп'ютерного обладнання, цілісність електричних кабелів та наявність заземлення. Забороняється використовувати пошкоджене обладнання або самостійно виконувати ремонт електротехнічних пристроїв без відповідної кваліфікації.

Важливим елементом охорони праці є дотримання правил електробезпеки. Усі комп’ютери, сервери та периферійні пристрої повинні підключатися через справну електромережу із захисним заземленням. Для запобігання пошкодженню обладнання та втраті даних рекомендується використовувати джерела безперебійного живлення.

Значна увага приділяється питанням пожежної безпеки. Робочі приміщення повинні бути обладнані засобами пожежогасіння, а працівники повинні бути ознайомлені з правилами евакуації та діями у разі виникнення надзвичайних ситуацій. Не допускається перевантаження електромережі великою кількістю підключених пристроїв або використання несправних електроподовжувачів.

Для профілактики професійних захворювань працівникам ІТ-сфери рекомендується регулярно виконувати фізичні вправи, дотримуватися правильної постави під час роботи та використовувати ергономічні меблі. Також необхідно проводити періодичні медичні огляди та контролювати стан органів зору.

Особливого значення набуває психологічна безпека працівників. Робота програмістів часто супроводжується високою відповідальністю, необхідністю виконання складних завдань та дотриманням жорстких термінів. Тому важливими чинниками є раціональна організація робочого процесу, підтримка сприятливого психологічного клімату в колективі та запобігання професійному вигоранню.

Отже, дотримання вимог охорони праці в ІТ-сфері дозволяє створити безпечні та комфортні умови роботи, підвищити ефективність праці працівників і забезпечити збереження їхнього здоров’я та працездатності.

4.3 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи було розглянуто питання безпеки життєдіяльності та охорони праці під час розроблення веборієнтованої системи підбору комп’ютерних комплектуючих. Проведений аналіз умов праці

програміста показав, що основними факторами, які впливають на ефективність його роботи та стан здоров’я, є тривале використання комп’ютерної техніки, підвищене навантаження на органи зору, статичне навантаження на опорно-руховий апарат, психоемоційне напруження та параметри мікроклімату робочого приміщення.

У результаті дослідження встановлено, що дотримання ергономічних вимог до організації робочого місця, нормативних параметрів освітлення, температурного режиму та вологості повітря сприяє зниженню втомлюваності працівника, підвищенню продуктивності праці та збереженню його працездатності протягом робочого дня. Важливе значення має також раціональне чергування періодів роботи та відпочинку, що дозволяє мінімізувати негативний вплив тривалої роботи за комп’ютером.

Розгляд основ охорони праці в ІТ-сфері дозволив визначити основні заходи щодо забезпечення безпечних умов праці, зокрема дотримання правил електробезпеки, пожежної безпеки, використання справного обладнання та організації безпечного робочого середовища. Окрему увагу приділено профілактиці професійних захворювань і підтриманню психологічного комфорту працівників, що є важливими чинниками ефективної діяльності в галузі інформаційних технологій.

Отже, виконаний аналіз підтвердив, що створення безпечних і комфортних умов праці є необхідною складовою професійної діяльності програміста. Дотримання вимог охорони праці та безпеки життєдіяльності сприяє збереженню здоров’я працівників, підвищенню якості виконання робіт та забезпеченню ефективного процесу розроблення програмного забезпечення.

ВИСНОВКИ

У ході виконання курсової роботи було вирішено актуальну задачу проєктування та розроблення веборієнтованої системи підбору комп'ютерних комплектуючих, призначеної для автоматизації процесу формування персональних комп'ютерних конфігурацій, перевірки сумісності компонентів та оформлення замовлень через вебінтерфейс. Проведене дослідження підтвердило доцільність використання сучасних вебтехнологій для створення інформаційних систем електронної комерції, орієнтованих на підвищення зручності взаємодії користувача з каталогом товарів та автоматизацію процесу вибору комплектуючих.

У теоретичній частині роботи було здійснено аналіз предметної області, розглянуто особливості функціонування сучасних інтернет-магазинів комп'ютерної техніки, досліджено принципи роботи конфігураторів комп'ютерних систем та визначено основні вимоги до програмного забезпечення. На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до системи, які стали основою подальшого проєктування.

У процесі проєктування було розроблено архітектуру програмної системи, спроєктовано структуру бази даних та створено UML-моделі, які відображають взаємодію користувачів із системою. Розроблена база даних забезпечує зберігання інформації про користувачів, товари, категорії, замовлення, позиції замовлень і конфігурації комп'ютерних збірок. Запропонована структура даних дозволяє ефективно обробляти інформацію та забезпечує можливість подальшого розширення функціоналу системи.

Практичним результатом роботи стала реалізація повнофункціонального вебзастосунку, який включає каталог комплектуючих, систему реєстрації та авторизації користувачів, конфігуратор персональних комп'ютерів, механізм перевірки сумісності комплектуючих, кошик покупця, модуль оформлення замовлень, систему перегляду історії покупок та адміністративну панель

керування товарами. Для реалізації серверної частини було використано Node.js та Express.js, для клієнтської частини – React, а для зберігання даних – MySQL.

У результаті тестування було підтверджено працездатність усіх основних модулів системи. Перевірка функціоналу каталогу, пошуку, фільтрації товарів, роботи конфігуратора, механізмів реєстрації та авторизації, оформлення замовлень і адміністрування показала коректність виконання операцій та відповідність реалізованого програмного забезпечення встановленим вимогам. Критичних помилок, які могли б впливати на працездатність системи, під час тестування не виявлено.

Практичне значення отриманих результатів полягає у можливості використання розробленої системи як основи для створення повноцінного інтернет-магазину комп’ютерної техніки або сервісу автоматизованого підбору комплектуючих. Реалізований механізм перевірки сумісності дозволяє зменшити кількість помилок під час формування конфігурацій ПК та підвищити якість обслуговування користувачів.

Отримані результати є достовірними, оскільки підтверджуються результатами модульного, функціонального та інтеграційного тестування системи. Подальший розвиток проєкту може бути спрямований на впровадження системи рекомендацій комплектуючих, інтеграцію з платіжними сервісами, реалізацію автоматичного підбору компонентів на основі бюджету користувача, а також використання технологій штучного інтелекту для формування оптимальних конфігурацій комп’ютерних систем.

ПЕРЕЛІК ДЖЕРЕЛ

1. AMD Processors Specifications. *AMD*. URL: https://www.amd.com/en/products/processors (дата звернення: 02.06.2026).

2. bcrypt.js Documentation. *npm*. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 02.06.2026).

3. Build My PC. *Build My PC*. URL: https://buildmypc.net (дата звернення: 02.06.2026).

4. Computer Components Buying Guide. *Tom's Hardware*. URL: https://www.tomshardware.com (дата звернення: 02.06.2026).

5. CPU Benchmark Database. *PassMark Software*. URL: https://www.cpubenchmark.net (дата звернення: 02.06.2026).

6. CSS Cascading Style Sheets. *World Wide Web Consortium*. URL: https://www.w3.org/Style/CSS (дата звернення: 02.06.2026).

7. Express.js Documentation. *Express*. URL: https://expressjs.com (дата звернення: 02.06.2026).

8. Git Documentation. *Git*. URL: https://git-scm.com/doc (дата звернення: 02.06.2026).

9. GPU Benchmark Database. *PassMark Software*. URL: https://www.videocardbenchmark.net (дата звернення: 02.06.2026).

10. HTML Living Standard. *WHATWG*. URL: https://html.spec.whatwg.org (дата звернення: 02.06.2026).

11. Intel Processors Specifications. *Intel*. URL: https://www.intel.com/content/www/us/en/products/processors.html (дата звернення: 02.06.2026).

12. JSON Web Tokens Introduction. *JWT.io*. URL: https://jwt.io/introduction (дата звернення: 02.06.2026).

13. Kingston Memory Products. *Kingston Technology*. URL: https://www.kingston.com (дата звернення: 02.06.2026).

14. MDN Web Docs. Fetch API. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 02.06.2026).

15. MDN Web Docs. JavaScript Guide. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 02.06.2026).

16. MSI Motherboards. *MSI*. URL: https://www.msi.com/Motherboards (дата звернення: 02.06.2026).

17. Multer Documentation. *Express.js Middleware*. URL: https://github.com/expressjs/multer (дата звернення: 02.06.2026).

18. MySQL Documentation. *MySQL*. URL: https://dev.mysql.com/doc (дата звернення: 02.06.2026).

19. Newegg PC Builder. *Newegg*. URL: https://www.newegg.com/tools/custom-pc-builder (дата звернення: 02.06.2026).

20. Node.js Documentation. *Node.js*. URL: https://nodejs.org/docs/latest/api (дата звернення: 02.06.2026).

21. npm Documentation. *npm*. URL: https://docs.npmjs.com (дата звернення: 02.06.2026).

22. PC Builder. *PC Builder*. URL: https://pcbuilder.net (дата звернення: 02.06.2026).

23. PCPartPicker. *PCPartPicker*. URL: https://pcpartpicker.com (дата звернення: 02.06.2026).

24. React Documentation. *React*. URL: https://react.dev (дата звернення: 02.06.2026).

25. React Router Documentation. *React Router*. URL: https://reactrouter.com (дата звернення: 02.06.2026).

26. REST API Tutorial. *REST API Tutorial*. URL: https://restfulapi.net (дата звернення: 02.06.2026).

27. Tailwind CSS Documentation. *Tailwind CSS*. URL: https://tailwindcss.com/docs (дата звернення: 02.06.2026).

28. UML Resource Page. *Object Management Group*. URL: https://www.omg.org/spec/UML (дата звернення: 02.06.2026).

29. Visual Studio Code Documentation. *Visual Studio Code*. URL: https://code.visualstudio.com/docs (дата звернення: 02.06.2026).

30. Vite Documentation. *Vite*. URL: https://vitejs.dev/guide (дата звернення: 02.06.2026).

ДОДАТКИ

Додаток А

Лістинг коду системи

```
/* setup: npm run setup (з папки server) */
require("dotenv").config();
const fs = require("fs");
const path = require("path");
const readline = require("readline");
const mysql = require("mysql2/promise");
const bcrypt = require("bcryptjs");

const DB_HOST = process.env.DB_HOST || "127.0.0.1";
const DB_USER = process.env.DB_USER || "root";
const DB_PASSWORD = process.env.DB_PASSWORD ?? "";
const DB_NAME = process.env.DB_NAME || "pc_shop_mvp";

const sqlDir = path.join(__dirname, "sql");

function baseConfig(withoutDatabase) {
  const cfg = {
    host: DB_HOST,
    user: DB_USER,
    password: DB_PASSWORD,
    multipleStatements: true,
  };
  if (!withoutDatabase) cfg.database = DB_NAME;
  return cfg;
}

async function connect(withoutDatabase) {
  return mysql.createConnection(baseConfig(withoutDatabase));
}

async function databaseExists(conn) {
  const [rows] = await conn.query("SHOW DATABASES LIKE ?", [DB_NAME]);
  return rows.length > 0;
}

async function dropAllTables(conn) {
  await conn.query("SET FOREIGN_KEY_CHECKS = 0");
  const [tables] = await conn.query(
    "SELECT TABLE_NAME AS t FROM information_schema.TABLES WHERE TABLE_SCHEMA = ?",
    [DB_NAME]
  );
  for (const row of tables) {
    await conn.query(`DROP TABLE IF EXISTS \`${row.t}\``);
  }
  await conn.query("SET FOREIGN_KEY_CHECKS = 1");
}

async function ensureDatabase(connNoDb) {
  await connNoDb.query(`CREATE DATABASE IF NOT EXISTS \`${DB_NAME}\` CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci`);
}

async function runSqlFile(conn, filename) {
  const full = path.join(sqlDir, filename);
  const sql = fs.readFileSync(full, "utf8");
  await conn.query(sql);
}

async function seedUsers(conn) {
  const hash = bcrypt.hashSync("password123", 10);
  await conn.query(
    `INSERT INTO users (email, password_hash, full_name, phone, role) VALUES
      ('admin@shop.local', ?, 'Адміністратор', '', 'admin'),
      ('demo@shop.local', ?, 'Демо клієнт', '+380501112233', 'customer')
     ON DUPLICATE KEY UPDATE password_hash = VALUES(password_hash), full_name = VALUES(full_name), role =
VALUES(role)`,
    [hash, hash]
  );
}

async function checkState() {
```

```
  let conn;
  try {
    conn = await connect(true);
    const exists = await databaseExists(conn);
    if (!exists) {
      console.log(`База "${DB_NAME}" не знайдена. Оберіть "Ініціалізувати БД".`);
      return;
    }
    await conn.end();
    conn = await connect(false);
    const [tables] = await conn.query("SHOW TABLES");
    const names = tables.map((row) => Object.values(row)[0]);
    console.log(`База "${DB_NAME}" існує. Таблиць: ${names.length}`);
    for (const t of names) {
      const [c] = await conn.query(`SELECT COUNT(*) AS n FROM \`${t}\``);
      console.log(`  - ${t}: ${c[0].n} рядків`);
    }
  } catch (e) {
    console.error("Помилка перевірки:", e.message);
  } finally {
    if (conn) await conn.end().catch(() => {});
  }
}

async function initDatabase() {
  let connNoDb = null;
  let conn = null;
  try {
    connNoDb = await connect(true);
    await ensureDatabase(connNoDb);
    await connNoDb.end();
    connNoDb = null;

    conn = await connect(false);
    console.log("Скидання таблиць…");
    await dropAllTables(conn);
    console.log("Створення схеми…");
    await runSqlFile(conn, "schema.sql");
    console.log("Заповнення мок-даними…");
    await runSqlFile(conn, "seed.sql");
    await seedUsers(conn);
    console.log("Готово. Користувачі: admin@shop.local / demo@shop.local, пароль: password123");
  } catch (e) {
    console.error("Помилка ініціалізації:", e.message);
  } finally {
    if (connNoDb) await connNoDb.end().catch(() => {});
    if (conn) await conn.end().catch(() => {});
  }
}


async function deleteDatabase() {
  let conn;
  try {
    conn = await connect(true);
    await conn.query(`DROP DATABASE IF EXISTS \`${DB_NAME}\``);
    console.log(`Базу "${DB_NAME}" видалено.`);
  } catch (e) {
    console.error("Помилка видалення:", e.message);
  } finally {
    if (conn) await conn.end().catch(() => {});
  }
}

function prompt(rl, q) {
  return new Promise((resolve) => rl.question(q, resolve));
}

async function main() {
  const rl = readline.createInterface({ input: process.stdin, output: process.stdout });
  try {
    while (true) {
      console.log(`
--- PC Shop / MySQL ---
1 – Перевірити стан БД
2 – Ініціалізувати БД (+ мок-дані)
3 – Видалити базу даних
0 – Вихід
`);
```

```
      const a = (await prompt(rl, "Оберіть: ")).trim();
      if (a === "1") await checkState();
      else if (a === "2") await initDatabase();
      else if (a === "3") {
        const ok = (await prompt(rl, `Це видалить "${DB_NAME}". Введіть YES: `)).trim();
        if (ok === "YES") await deleteDatabase();
        else console.log("Скасовано.");
      } else if (a === "0") break;
      else console.log("Невідома опція.");
    }
  } finally {
    rl.close();
  }
}

main();
```

```
SET NAMES utf8mb4;

CREATE TABLE IF NOT EXISTS users (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  email VARCHAR(190) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  full_name VARCHAR(120) NOT NULL DEFAULT '',
  phone VARCHAR(40) NOT NULL DEFAULT '',
  role ENUM('customer', 'admin') NOT NULL DEFAULT 'customer',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  UNIQUE KEY uq_users_email (email)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS categories (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(120) NOT NULL,
  slug VARCHAR(120) NOT NULL,
  sort_order INT NOT NULL DEFAULT 0,
  PRIMARY KEY (id),
  UNIQUE KEY uq_categories_slug (slug)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS products (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  category_id BIGINT UNSIGNED NOT NULL,
  name VARCHAR(255) NOT NULL,
  slug VARCHAR(200) NOT NULL,
  description TEXT NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  stock INT NOT NULL DEFAULT 0,
  image_url VARCHAR(512) NOT NULL DEFAULT '',
  specs JSON NULL,
  is_active TINYINT(1) NOT NULL DEFAULT 1,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  UNIQUE KEY uq_products_slug (slug),
  KEY idx_products_category (category_id),
  CONSTRAINT fk_products_category FOREIGN KEY (category_id) REFERENCES categories (id)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS cart_items (
  user_id BIGINT UNSIGNED NOT NULL,
  product_id BIGINT UNSIGNED NOT NULL,
  quantity INT NOT NULL DEFAULT 1,
  PRIMARY KEY (user_id, product_id),
  CONSTRAINT fk_cart_user FOREIGN KEY (user_id) REFERENCES users (id)
    ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_cart_product FOREIGN KEY (product_id) REFERENCES products (id)
    ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS orders (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_id BIGINT UNSIGNED NOT NULL,
  status ENUM('pending', 'processing', 'shipped', 'delivered', 'cancelled') NOT NULL DEFAULT 'pending',
  delivery_address VARCHAR(500) NOT NULL,
  payment_method VARCHAR(80) NOT NULL,
  shipping_method VARCHAR(80) NOT NULL,
  total DECIMAL(10, 2) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
  PRIMARY KEY (id),
  KEY idx_orders_user (user_id),
  CONSTRAINT fk_orders_user FOREIGN KEY (user_id) REFERENCES users (id)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS order_items (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  order_id BIGINT UNSIGNED NOT NULL,
  product_id BIGINT UNSIGNED NOT NULL,
  quantity INT NOT NULL,
  price_at_purchase DECIMAL(10, 2) NOT NULL,
  PRIMARY KEY (id),
  KEY idx_order_items_order (order_id),
  CONSTRAINT fk_order_items_order FOREIGN KEY (order_id) REFERENCES orders (id)
    ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_order_items_product FOREIGN KEY (product_id) REFERENCES products (id)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
const mysql = require("mysql2/promise");
require("dotenv").config();

function poolConfig() {
  return {
    host: process.env.DB_HOST || "127.0.0.1",
    user: process.env.DB_USER || "root",
    password: process.env.DB_PASSWORD ?? "",
    database: process.env.DB_NAME || "pc_shop_mvp",
    waitForConnections: true,
    connectionLimit: 10,
    namedPlaceholders: true,
  };
}

let pool;

function getPool() {
  if (!pool) {
    pool = mysql.createPool(poolConfig());
  }
  return pool;
}

async function ping() {
  const p = getPool();
  const [rows] = await p.query("SELECT 1 AS ok");
  return rows[0]?.ok === 1;
}

module.exports = { getPool, ping, poolConfig };
```

```
require("dotenv").config();
const path = require("path");
const express = require("express");
const cors = require("cors");
const cookieParser = require("cookie-parser");
const { ping } = require("./db");
const { requireAuth } = require("./middleware/auth");
const { requireAdmin } = require("./middleware/admin");
const authRoutes = require("./routes/auth");
const adminProductsRoutes = require("./routes/adminProducts");
const { uploadAdminImage } = require("./routes/adminUpload");
const catalogRoutes = require("./routes/catalog");
const cartRoutes = require("./routes/cart");
const ordersRoutes = require("./routes/orders");

const app = express();
const PORT = Number(process.env.PORT) || 3001;
const CLIENT_ORIGIN = process.env.CLIENT_ORIGIN || "http://localhost:5173";

app.use(
  cors({
    origin: CLIENT_ORIGIN,
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());
```

```
app.use("/uploads", express.static(path.join(__dirname, "../uploads"), { maxAge: "7d" }));

app.use("/api/auth", authRoutes);
app.use("/api/catalog", catalogRoutes);
app.use("/api/cart", cartRoutes);
app.use("/api/orders", ordersRoutes);
app.use("/api/admin/products", requireAuth, requireAdmin, adminProductsRoutes);
app.post("/api/admin/upload", requireAuth, requireAdmin, uploadAdminImage);

app.get("/api/health", async (_req, res) => {
  try {
    const dbOk = await ping();
    res.json({ ok: true, db: dbOk ? "up" : "down" });
  } catch {
    res.status(503).json({ ok: false, db: "down" });
  }
});

app.listen(PORT, () => {
  console.log(`API http://localhost:${PORT}`);
});

const jwt = require("jsonwebtoken");

const COOKIE_NAME = "pcshop_token";

function secret() {
  const s = process.env.JWT_SECRET;
  if (!s || s.length < 8) {
    if (process.env.NODE_ENV === "production") {
      throw new Error("JWT_SECRET у .env має бути не коротший за 8 символів");
    }
    console.warn("[auth] JWT_SECRET не задано – використовується dev-значення (лише для розробки)");
    return "pc-shop-dev-secret-min-8-chars";
  }
  return s;
}

function signToken(payload) {
  return jwt.sign(payload, secret(), { expiresIn: "7d" });
}

function verifyToken(token) {
  return jwt.verify(token, secret());
}

module.exports = { COOKIE_NAME, signToken, verifyToken };

const PAYMENT_METHODS = ["Картка онлайн", "Готівка при отриманні"];
const SHIPPING_METHODS = ["Нова Пошта", "Самовивіз з магазину"];

function isAllowedPayment(v) {
  return PAYMENT_METHODS.includes(String(v || "").trim());
}

function isAllowedShipping(v) {
  return SHIPPING_METHODS.includes(String(v || "").trim());
}

module.exports = {
  PAYMENT_METHODS,
  SHIPPING_METHODS,
  isAllowedPayment,
  isAllowedShipping,
};

function deliveryAddressIsPlausible(s) {
  const t = String(s || "").trim();
  const meaningful = t.match(/\p{L}|\p{N}/gu);
  return meaningful != null && meaningful.length >= 5;
}

module.exports = { deliveryAddressIsPlausible };

const { getPool } = require("../db");

async function requireAdmin(req, res, next) {
  if (!req.userId) {
```

```
    return res.status(401).json({ error: "Потрібна авторизація" });
  }
  try {
    const pool = getPool();
    const [rows] = await pool.query(`SELECT role FROM users WHERE id = :id LIMIT 1`, { id: req.userId });
    const role = rows[0]?.role;
    if (role !== "admin") {
      return res.status(403).json({ error: "Доступ лише для адміністратора" });
    }
    next();
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Помилка перевірки прав" });
  }
}

module.exports = { requireAdmin };

const { COOKIE_NAME, verifyToken } = require("../lib/authToken");

function requireAuth(req, res, next) {
  const token = req.cookies?.[COOKIE_NAME];
  if (!token) {
    return res.status(401).json({ error: "Потрібна авторизація" });
  }
  try {
    const payload = verifyToken(token);
    req.userId = Number(payload.sub);
    if (!req.userId) {
      return res.status(401).json({ error: "Недійсний токен" });
    }
    next();
  } catch {
    return res.status(401).json({ error: "Недійсний або прострочений токен" });
  }
}

module.exports = { requireAuth };

const express = require("express");
const { getPool } = require("../db");

const router = express.Router();

function parseSpecsInput(raw) {
  if (raw == null || raw === "") return null;
  if (typeof raw === "object") return raw;
  const s = String(raw).trim();
  if (!s) return null;
  try {
    return JSON.parse(s);
  } catch {
    throw new Error("JSON");
  }
}

function slugOk(s) {
  return /^[a-z0-9]+(?:-[a-z0-9]+)*$/.test(String(s || "").trim());
}

router.get("/", async (_req, res) => {
  try {
    const pool = getPool();
    const [rows] = await pool.query(
      `SELECT p.id, p.category_id, p.name, p.slug, p.description, p.price, p.stock, p.image_url, p.specs,
p.is_active, p.created_at,
              c.name AS category_name, c.slug AS category_slug
       FROM products p
       INNER JOIN categories c ON c.id = p.category_id
       ORDER BY p.id DESC`
    );
    const items = rows.map((r) => {
      let specs = null;
      if (r.specs != null) {
        try {
          specs = typeof r.specs === "string" ? JSON.parse(r.specs) : r.specs;
        } catch {
          specs = null;
        }
```

```
      }
      return {
        ...r,
        price: Number(r.price),
        stock: Number(r.stock),
        is_active: Number(r.is_active) === 1,
        specs,
      };
    });
    res.json({ items });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити товари" });
  }
});

router.post("/", async (req, res) => {
  const category_id = Number(req.body?.category_id);
  const name = String(req.body?.name || "").trim();
  const slug = String(req.body?.slug || "").trim().toLowerCase();
  const description = String(req.body?.description || "").trim();
  const price = Number(req.body?.price);
  const stock = parseInt(req.body?.stock, 10);
  const image_url = String(req.body?.image_url || "").trim().slice(0, 512);

  if (!category_id || !name || !slug || !description) {
    return res.status(400).json({ error: "Заповніть категорію, назву, slug та опис" });
  }
  if (!slugOk(slug)) {
    return res.status(400).json({ error: "Slug: лише малі літери, цифри та дефіси" });
  }
  if (Number.isNaN(price) || price < 0) {
    return res.status(400).json({ error: "Некоректна ціна" });
  }
  if (Number.isNaN(stock) || stock < 0) {
    return res.status(400).json({ error: "Некоректний залишок" });
  }

  let specs = null;
  try {
    specs = parseSpecsInput(req.body?.specs);
  } catch {
    return res.status(400).json({ error: "Характеристики (specs) мають бути валідним JSON" });
  }

  const pool = getPool();
  try {
    const [r] = await pool.query(
      `INSERT INTO products (category_id, name, slug, description, price, stock, image_url, specs)
       VALUES (:cid, :name, :slug, :desc, :price, :stock, :img, :specs)`,
      {
        cid: category_id,
        name,
        slug,
        desc: description,
        price,
        stock,
        img: image_url,
        specs: specs == null ? null : JSON.stringify(specs),
      }
    );
    res.status(201).json({ id: r.insertId });
  } catch (e) {
    if (e.code === "ER_DUP_ENTRY") {
      return res.status(409).json({ error: "Товар з таким slug вже є" });
    }
    if (e.code === "ER_NO_REFERENCED_ROW_2") {
      return res.status(400).json({ error: "Некоректна категорія" });
    }
    console.error(e);
    res.status(500).json({ error: "Не вдалося створити товар" });
  }
});

router.patch("/:id", async (req, res) => {
  const id = Number(req.params.id);
  if (!id) {
    return res.status(400).json({ error: "Некоректний id" });
  }
```

```
const category_id = req.body?.category_id != null ? Number(req.body.category_id) : null;
const name = req.body?.name != null ? String(req.body.name).trim() : null;
const slug = req.body?.slug != null ? String(req.body.slug).trim().toLowerCase() : null;
const description = req.body?.description != null ? String(req.body.description).trim() : null;
const price = req.body?.price != null ? Number(req.body.price) : null;
const stock = req.body?.stock != null ? parseInt(req.body.stock, 10) : null;
const image_url = req.body?.image_url != null ? String(req.body.image_url).trim().slice(0, 512) : null;

if (slug != null && !slugOk(slug)) {
  return res.status(400).json({ error: "Slug: лише малі літери, цифри та дефіси" });
}
if (price != null && (Number.isNaN(price) || price < 0)) {
  return res.status(400).json({ error: "Некоректна ціна" });
}
if (stock != null && (Number.isNaN(stock) || stock < 0)) {
  return res.status(400).json({ error: "Некоректний залишок" });
}

let specsUpdate = undefined;
if (Object.prototype.hasOwnProperty.call(req.body, "specs")) {
  try {
    specsUpdate = parseSpecsInput(req.body.specs);
  } catch {
    return res.status(400).json({ error: "Характеристики (specs) мають бути валідним JSON" });
  }
}

const fields = [];
const params = { id };

if (category_id != null) {
  fields.push("category_id = :category_id");
  params.category_id = category_id;
}
if (name != null) {
  fields.push("name = :name");
  params.name = name;
}
if (slug != null) {
  fields.push("slug = :slug");
  params.slug = slug;
}
if (description != null) {
  fields.push("description = :description");
  params.description = description;
}
if (price != null) {
  fields.push("price = :price");
  params.price = price;
}
if (stock != null) {
  fields.push("stock = :stock");
  params.stock = stock;
}
if (image_url != null) {
  fields.push("image_url = :image_url");
  params.image_url = image_url;
}
if (specsUpdate !== undefined) {
  fields.push("specs = :specs");
  params.specs = specsUpdate == null ? null : JSON.stringify(specsUpdate);
}

if (!fields.length) {
  return res.status(400).json({ error: "Немає полів для оновлення" });
}

const pool = getPool();
try {
  const [r] = await pool.query(`UPDATE products SET ${fields.join(", ")} WHERE id = :id`, params);
  if (r.affectedRows === 0) {
    return res.status(404).json({ error: "Товар не знайдено" });
  }
  res.json({ ok: true });
} catch (e) {
  if (e.code === "ER_DUP_ENTRY") {
    return res.status(409).json({ error: "Товар з таким slug вже є" });
  }
```

```
    console.error(e);
    res.status(500).json({ error: "Не вдалося оновити товар" });
  }
});

router.delete("/:id", async (req, res) => {
  const id = Number(req.params.id);
  if (!id) {
    return res.status(400).json({ error: "Некоректний id" });
  }

  const pool = getPool();
  const [existing] = await pool.query(`SELECT id, is_active FROM products WHERE id = :id LIMIT 1`, { id });
  if (!existing[0]) {
    return res.status(404).json({ error: "Товар не знайдено" });
  }

  if (Number(existing[0].is_active) === 0) {
    await pool.query(`DELETE FROM cart_items WHERE product_id = :id`, { id });
    return res.json({ ok: true, archived: true, alreadyArchived: true });
  }

  try {
    const [r] = await pool.query(`DELETE FROM products WHERE id = :id`, { id });
    if (r.affectedRows === 1) {
      return res.json({ ok: true });
    }
    return res.status(404).json({ error: "Товар не знайдено" });
  } catch (e) {
    if (e.errno !== 1451 && e.code !== "ER_ROW_IS_REFERENCED_2") {
      console.error(e);
      return res.status(500).json({ error: "Не вдалося видалити товар" });
    }
  }

  const conn = await pool.getConnection();
  try {
    await conn.beginTransaction();
    const [u] = await conn.query(
      `UPDATE products SET is_active = 0, slug = CONCAT(LEFT(slug, 170), '-a', id) WHERE id = :id AND
is_active = 1`,
      { id }
    );
    await conn.query(`DELETE FROM cart_items WHERE product_id = :id`, { id });
    await conn.commit();
    return res.json({ ok: true, archived: true, slugChanged: u.affectedRows === 1 });
  } catch (e) {
    await conn.rollback().catch(() => {});
    console.error(e);
    return res.status(500).json({ error: "Не вдалося зняти товар з продажу" });
  } finally {
    conn.release();
  }
});

module.exports = router;

const path = require("path");
const fs = require("fs");
const multer = require("multer");

const uploadDir = path.join(__dirname, "../../uploads");
fs.mkdirSync(uploadDir, { recursive: true });

const storage = multer.diskStorage({
  destination: (_req, _file, cb) => cb(null, uploadDir),
  filename: (_req, file, cb) => {
    const ext = path.extname(file.originalname || "").toLowerCase().slice(0, 8);
    const name = `${Date.now()}-${Math.random().toString(36).slice(2, 10)}${ext || ".bin"}`;
    cb(null, name);
  },
});

const upload = multer({
  storage,
  limits: { fileSize: 3 * 1024 * 1024 },
  fileFilter: (_req, file, cb) => {
    const ok = ["image/jpeg", "image/png", "image/webp", "image/gif"].includes(file.mimetype);
    if (!ok) {
```

```
      cb(new Error("Лише зображення: JPEG, PNG, WebP або GIF"));
      return;
    }
    cb(null, true);
  },
});

function uploadAdminImage(req, res) {
  upload.single("file")(req, res, (err) => {
    if (err) {
      return res.status(400).json({ error: err.message || "Помилка завантаження" });
    }
    if (!req.file) {
      return res.status(400).json({ error: "Файл не отримано (поле file)" });
    }
    const port = Number(process.env.PORT) || 3001;
    const base = (process.env.PUBLIC_API_URL || "").replace(/\/$/, "") || `http://127.0.0.1:${port}`;
    const url = `${base}/uploads/${req.file.filename}`;
    res.json({ url });
  });
}

module.exports = { uploadAdminImage };

const express = require("express");
const bcrypt = require("bcryptjs");
const { getPool } = require("../db");
const { COOKIE_NAME, signToken } = require("../lib/authToken");
const { requireAuth } = require("../middleware/auth");

const router = express.Router();

const COOKIE_OPTS = {
  httpOnly: true,
  sameSite: "lax",
  path: "/",
  maxAge: 7 * 24 * 60 * 60 * 1000,
};

function isEmail(s) {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(String(s || "").trim());
}

router.post("/register", async (req, res) => {
  const email = String(req.body?.email || "").trim().toLowerCase();
  const password = String(req.body?.password || "");
  const full_name = String(req.body?.full_name || "").trim();
  const phone = String(req.body?.phone || "").trim();

  if (!isEmail(email)) {
    return res.status(400).json({ error: "Некоректний email" });
  }
  if (password.length < 6) {
    return res.status(400).json({ error: "Пароль мінімум 6 символів" });
  }
  if (full_name.length < 2) {
    return res.status(400).json({ error: "Вкажіть ім’я (мінімум 2 символи)" });
  }

  const hash = bcrypt.hashSync(password, 10);
  const pool = getPool();
  try {
    const [r] = await pool.query(
      `INSERT INTO users (email, password_hash, full_name, phone, role) VALUES
(:email, :hash, :full_name, :phone, 'customer')`,
      { email, hash, full_name, phone }
    );
    const id = r.insertId;
    const token = signToken({ sub: String(id) });
    res.cookie(COOKIE_NAME, token, COOKIE_OPTS);
    return res.status(201).json({
      user: { id, email, full_name, phone, role: "customer" },
    });
  } catch (e) {
    if (e.code === "ER_DUP_ENTRY") {
      return res.status(409).json({ error: "Користувач з таким email вже є" });
    }
    console.error(e);
    return res.status(500).json({ error: "Помилка сервера" });
```

```
  }
});

router.post("/login", async (req, res) => {
  const email = String(req.body?.email || "").trim().toLowerCase();
  const password = String(req.body?.password || "");
  if (!email || !password) {
    return res.status(400).json({ error: "Email і пароль обов’язкові" });
  }

  const pool = getPool();
  const [rows] = await pool.query(
    `SELECT id, email, password_hash, full_name, phone, role FROM users WHERE email = :email LIMIT 1`,
    { email }
  );
  const user = rows[0];
  if (!user || !bcrypt.compareSync(password, user.password_hash)) {
    return res.status(401).json({ error: "Невірний email або пароль" });
  }

  const token = signToken({ sub: String(user.id) });
  res.cookie(COOKIE_NAME, token, COOKIE_OPTS);
  return res.json({
    user: {
      id: user.id,
      email: user.email,
      full_name: user.full_name,
      phone: user.phone,
      role: user.role,
    },
  });
});

router.post("/logout", (_req, res) => {
  res.clearCookie(COOKIE_NAME, { path: "/", httpOnly: true, sameSite: "lax" });
  res.json({ ok: true });
});

router.get("/me", requireAuth, async (req, res) => {
  const pool = getPool();
  const [rows] = await pool.query(
    `SELECT id, email, full_name, phone, role FROM users WHERE id = :id LIMIT 1`,
    { id: req.userId }
  );
  const user = rows[0];
  if (!user) {
    res.clearCookie(COOKIE_NAME, { path: "/", httpOnly: true, sameSite: "lax" });
    return res.status(401).json({ error: "Користувача не знайдено" });
  }
  res.json({
    user: {
      id: user.id,
      email: user.email,
      full_name: user.full_name,
      phone: user.phone,
      role: user.role,
    },
  });
});

router.patch("/profile", requireAuth, async (req, res) => {
  const full_name = String(req.body?.full_name ?? "").trim();
  const phone = String(req.body?.phone ?? "").trim();

  if (full_name.length < 2) {
    return res.status(400).json({ error: "Ім’я мінімум 2 символи" });
  }
  if (full_name.length > 120) {
    return res.status(400).json({ error: "Ім’я занадто довге" });
  }
  if (phone.length > 40) {
    return res.status(400).json({ error: "Телефон занадто довгий" });
  }

  const pool = getPool();
  await pool.query(
    `UPDATE users SET full_name = :full_name, phone = :phone WHERE id = :id`,
    { full_name, phone, id: req.userId }
  );
```

```
  const [rows] = await pool.query(
    `SELECT id, email, full_name, phone, role FROM users WHERE id = :id LIMIT 1`,
    { id: req.userId }
  );
  const user = rows[0];
  res.json({
    user: {
      id: user.id,
      email: user.email,
      full_name: user.full_name,
      phone: user.phone,
      role: user.role,
    },
  });
});

module.exports = router;

const express = require("express");
const { getPool } = require("../db");
const { requireAuth } = require("../middleware/auth");

const router = express.Router();

router.get("/", requireAuth, async (req, res) => {
  const pool = getPool();
  const [rows] = await pool.query(
    `SELECT ci.product_id, ci.quantity,
            p.name, p.slug, p.price, p.stock, p.image_url
     FROM cart_items ci
     INNER JOIN products p ON p.id = ci.product_id AND p.is_active = 1
     WHERE ci.user_id = :uid
     ORDER BY p.name ASC`,
    { uid: req.userId }
  );

  let totalPrice = 0;
  let totalQty = 0;
  const items = rows.map((row) => {
    const price = Number(row.price);
    const quantity = Number(row.quantity);
    const line = price * quantity;
    totalPrice += line;
    totalQty += quantity;
    return {
      product_id: row.product_id,
      quantity,
      name: row.name,
      slug: row.slug,
      price,
      stock: Number(row.stock),
      image_url: row.image_url,
      line_total: line,
    };
  });

  res.json({ items, totalQty, totalPrice });
});

router.post("/items", requireAuth, async (req, res) => {
  const productId = Number(req.body?.product_id);
  const addQty = Math.min(99, Math.max(1, parseInt(req.body?.quantity, 10) || 1));
  const fromCatalog = Boolean(req.body?.from_catalog);

  if (!productId) {
    return res.status(400).json({ error: "Некоректний товар" });
  }

  const pool = getPool();
  const [products] = await pool.query(
    `SELECT id, stock FROM products WHERE id = :id AND is_active = 1 LIMIT 1`,
    { id: productId }
  );
  const product = products[0];
  if (!product) {
    return res.status(404).json({ error: "Товар не знайдено" });
  }
```

```
  const [cartRows] = await pool.query(
    `SELECT quantity FROM cart_items WHERE user_id = :uid AND product_id = :pid LIMIT 1`,
    { uid: req.userId, pid: productId }
  );
  const existing = cartRows[0];

  if (fromCatalog) {
    if (existing) {
      return res.json({ ok: true, alreadyInCart: true });
    }
    if (Number(product.stock) < 1) {
      return res.status(400).json({ error: "Немає в наявності" });
    }
    await pool.query(
      `INSERT INTO cart_items (user_id, product_id, quantity) VALUES (:uid, :pid, 1)`,
      { uid: req.userId, pid: productId }
    );
    return res.status(201).json({ ok: true });
  }

  const current = existing ? Number(existing.quantity) : 0;
  if (current + addQty > Number(product.stock)) {
    return res.status(400).json({ error: "Недостатньо товару на складі" });
  }

  await pool.query(
    `INSERT INTO cart_items (user_id, product_id, quantity) VALUES (:uid, :pid, :qty)
     ON DUPLICATE KEY UPDATE quantity = quantity + :inc`,
    { uid: req.userId, pid: productId, qty: addQty, inc: addQty }
  );

  res.status(201).json({ ok: true });
});

router.patch("/items/:productId", requireAuth, async (req, res) => {
  const productId = Number(req.params.productId);
  const qty = parseInt(req.body?.quantity, 10);
  if (!productId || Number.isNaN(qty) || qty < 1 || qty > 99) {
    return res.status(400).json({ error: "Некоректна кількість" });
  }

  const pool = getPool();
  const [products] = await pool.query(`SELECT stock FROM products WHERE id = :id AND is_active = 1 LIMIT 1`,
{
    id: productId,
  });
  const product = products[0];
  if (!product) {
    return res.status(404).json({ error: "Товар не знайдено" });
  }
  if (qty > Number(product.stock)) {
    return res.status(400).json({ error: "Недостатньо на складі" });
  }

  const [r] = await pool.query(
    `UPDATE cart_items SET quantity = :qty WHERE user_id = :uid AND product_id = :pid`,
    { qty, uid: req.userId, pid: productId }
  );
  if (r.affectedRows === 0) {
    return res.status(404).json({ error: "Позиції немає в кошику" });
  }

  res.json({ ok: true });
});

router.delete("/items/:productId", requireAuth, async (req, res) => {
  const productId = Number(req.params.productId);
  if (!productId) {
    return res.status(400).json({ error: "Некоректний товар" });
  }
  const pool = getPool();
  await pool.query(`DELETE FROM cart_items WHERE user_id = :uid AND product_id = :pid`, {
    uid: req.userId,
    pid: productId,
  });
  res.json({ ok: true });
});

module.exports = router;
```

```
const express = require("express");
const { getPool } = require("../db");

const router = express.Router();

function parseSpecs(raw) {
  if (raw == null) return null;
  try {
    return typeof raw === "string" ? JSON.parse(raw) : raw;
  } catch {
    return null;
  }
}

router.get("/categories", async (_req, res) => {
  try {
    const pool = getPool();
    const [rows] = await pool.query(
      `SELECT id, name, slug, sort_order FROM categories ORDER BY sort_order ASC, id ASC`
    );
    res.json({ categories: rows });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити категорії" });
  }
});

router.get("/products", async (req, res) => {
  const categorySlug = typeof req.query.category === "string" ? req.query.category.trim() : "";
  const searchRaw = typeof req.query.search === "string" ? req.query.search.trim() : "";
  const search = searchRaw.slice(0, 120);

  const page = Math.max(1, parseInt(req.query.page, 10) || 1);
  const limit = Math.min(50, Math.max(1, parseInt(req.query.limit, 10) || 12));
  const offset = (page - 1) * limit;

  let where = "p.is_active = 1";
  const params = {};
  if (categorySlug) {
    where += " AND c.slug = :categorySlug";
    params.categorySlug = categorySlug;
  }
  if (search) {
    where += " AND p.name LIKE :searchLike";
    params.searchLike = `%${search}%`;
  }

  try {
    const pool = getPool();
    const [countRows] = await pool.query(
      `SELECT COUNT(*) AS n
       FROM products p
       INNER JOIN categories c ON c.id = p.category_id
       WHERE ${where}`,
      params
    );
    const total = Number(countRows[0]?.n || 0);
    const pages = Math.max(1, Math.ceil(total / limit));

    const [items] = await pool.query(
      `SELECT p.id, p.name, p.slug, p.price, p.stock, p.image_url, p.specs,
              c.slug AS category_slug, c.name AS category_name
       FROM products p
       INNER JOIN categories c ON c.id = p.category_id
       WHERE ${where}
       ORDER BY p.id DESC
       LIMIT ${Number(limit)} OFFSET ${Number(offset)}`,
      params
    );

    const normalized = items.map((row) => ({
      ...row,
      price: Number(row.price),
      specs: parseSpecs(row.specs),
    }));

    res.json({
      items: normalized,
```

```
      total,
      page,
      limit,
      pages,
    });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити товари" });
  }
});

router.get("/products/:slug", async (req, res) => {
  const slug = String(req.params.slug || "").trim();
  if (!slug || slug.length > 200) {
    return res.status(400).json({ error: "Некоректний slug" });
  }

  try {
    const pool = getPool();
    const [rows] = await pool.query(
      `SELECT p.id, p.name, p.slug, p.description, p.price, p.stock, p.image_url, p.specs, p.created_at,
              c.id AS category_id, c.slug AS category_slug, c.name AS category_name
       FROM products p
       INNER JOIN categories c ON c.id = p.category_id
       WHERE p.slug = :slug AND p.is_active = 1
       LIMIT 1`,
      { slug }
    );
    const row = rows[0];
    if (!row) {
      return res.status(404).json({ error: "Товар не знайдено" });
    }

    const product = {
      id: row.id,
      name: row.name,
      slug: row.slug,
      description: row.description,
      price: Number(row.price),
      stock: Number(row.stock),
      image_url: row.image_url,
      specs: parseSpecs(row.specs),
      created_at: row.created_at,
      category_id: row.category_id,
      category_slug: row.category_slug,
      category_name: row.category_name,
    };

    res.json({ product });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити товар" });
  }
});

module.exports = router;

const express = require("express");
const { getPool } = require("../db");
const { requireAuth } = require("../middleware/auth");
const { isAllowedPayment, isAllowedShipping, PAYMENT_METHODS, SHIPPING_METHODS } =
require("../lib/orderOptions");
const { deliveryAddressIsPlausible } = require("../lib/validateAddress");

const router = express.Router();

router.get("/form-options", (_req, res) => {
  res.json({
    payment_methods: PAYMENT_METHODS,
    shipping_methods: SHIPPING_METHODS,
  });
});

router.get("/", requireAuth, async (req, res) => {
  try {
    const pool = getPool();
    const [rows] = await pool.query(
      `SELECT o.id, o.status, o.total, o.created_at, o.delivery_address, o.payment_method, o.shipping_method,
              (SELECT COUNT(*) FROM order_items oi WHERE oi.order_id = o.id) AS item_count
```

```
       FROM orders o
       WHERE o.user_id = :uid
       ORDER BY o.created_at DESC`,
      { uid: req.userId }
    );
    const orders = rows.map((r) => ({
      id: r.id,
      status: r.status,
      total: Number(r.total),
      created_at: r.created_at,
      delivery_address: r.delivery_address,
      payment_method: r.payment_method,
      shipping_method: r.shipping_method,
      item_count: Number(r.item_count),
    }));
    res.json({ orders });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити замовлення" });
  }
});

router.get("/:id", requireAuth, async (req, res) => {
  const id = Number(req.params.id);
  if (!id) {
    return res.status(400).json({ error: "Некоректний id" });
  }
  try {
    const pool = getPool();
    const [orders] = await pool.query(
      `SELECT id, user_id, status, total, created_at, delivery_address, payment_method, shipping_method
       FROM orders WHERE id = :id LIMIT 1`,
      { id }
    );
    const order = orders[0];
    if (!order || Number(order.user_id) !== req.userId) {
      return res.status(404).json({ error: "Замовлення не знайдено" });
    }

    const [items] = await pool.query(
      `SELECT oi.product_id, oi.quantity, oi.price_at_purchase, p.name, p.slug
       FROM order_items oi
       INNER JOIN products p ON p.id = oi.product_id
       WHERE oi.order_id = :oid
       ORDER BY oi.id ASC`,
      { oid: id }
    );

    res.json({
      order: {
        id: order.id,
        status: order.status,
        total: Number(order.total),
        created_at: order.created_at,
        delivery_address: order.delivery_address,
        payment_method: order.payment_method,
        shipping_method: order.shipping_method,
        items: items.map((r) => ({
          product_id: r.product_id,
          name: r.name,
          slug: r.slug,
          quantity: Number(r.quantity),
          price_at_purchase: Number(r.price_at_purchase),
          line_total: Number(r.quantity) * Number(r.price_at_purchase),
        })),
      },
    });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: "Не вдалося завантажити замовлення" });
  }
});

router.post("/", requireAuth, async (req, res) => {
  const delivery_address = String(req.body?.delivery_address || "").trim();
  const payment_method = String(req.body?.payment_method || "").trim();
  const shipping_method = String(req.body?.shipping_method || "").trim();

  if (delivery_address.length < 10) {
```

```
    return res.status(400).json({ error: "Адреса доставки мінімум 10 символів" });
  }
  if (delivery_address.length > 500) {
    return res.status(400).json({ error: "Адреса занадто довга" });
  }
  if (!deliveryAddressIsPlausible(delivery_address)) {
    return res.status(400).json({
      error: "Вкажіть змістовну адресу (місто, відділення або вулиця, будинок – мінімум 5 літер чи цифр загалом)",
    });
  }
  if (!isAllowedPayment(payment_method)) {
    return res.status(400).json({ error: "Оберіть спосіб оплати зі списку" });
  }
  if (!isAllowedShipping(shipping_method)) {
    return res.status(400).json({ error: "Оберіть спосіб доставки зі списку" });
  }

  const pool = getPool();
  let conn;
  try {
    conn = await pool.getConnection();
    await conn.beginTransaction();

    const [lines] = await conn.query(
      `SELECT ci.product_id, ci.quantity, p.price, p.stock
       FROM cart_items ci
       INNER JOIN products p ON p.id = ci.product_id AND p.is_active = 1
       WHERE ci.user_id = :uid`,
      { uid: req.userId }
    );

    if (!lines.length) {
      await conn.rollback();
      return res.status(400).json({ error: "Кошик порожній" });
    }

    let total = 0;
    for (const row of lines) {
      const qty = Number(row.quantity);
      const stock = Number(row.stock);
      const price = Number(row.price);
      if (qty > stock || qty < 1) {
        await conn.rollback();
        return res.status(400).json({ error: "Недостатньо товару на складі для оформлення" });
      }
      total += price * qty;
    }

    const [ins] = await conn.query(
      `INSERT INTO orders (user_id, status, delivery_address, payment_method, shipping_method, total)
       VALUES (:uid, 'pending', :addr, :pay, :ship, :total)`,
      {
        uid: req.userId,
        addr: delivery_address,
        pay: payment_method,
        ship: shipping_method,
        total,
      }
    );
    const orderId = ins.insertId;

    for (const row of lines) {
      const qty = Number(row.quantity);
      const price = Number(row.price);
      const pid = row.product_id;

      await conn.query(
        `INSERT INTO order_items (order_id, product_id, quantity, price_at_purchase)
         VALUES (:oid, :pid, :qty, :price)`,
        { oid: orderId, pid, qty, price }
      );

      const [upd] = await conn.query(
        `UPDATE products SET stock = stock - :q WHERE id = :pid AND stock >= :q`,
        { q: qty, pid }
      );
      if (upd.affectedRows !== 1) {
        await conn.rollback();
```

```
        return res.status(400).json({ error: "Не вдалося зарезервувати товар (склад)" });
      }
    }

    await conn.query(`DELETE FROM cart_items WHERE user_id = :uid`, { uid: req.userId });
    await conn.commit();

    res.status(201).json({
      order: {
        id: orderId,
        status: "pending",
        total,
      },
    });
  } catch (e) {
    if (conn) await conn.rollback().catch(() => {});
    console.error(e);
    res.status(500).json({ error: "Не вдалося оформити замовлення" });
  } finally {
    if (conn) conn.release();
  }
});

module.exports = router;
```

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      "/api": { target: "http://127.0.0.1:3001", changeOrigin: true },
      "/uploads": { target: "http://127.0.0.1:3001", changeOrigin: true },
    },
  },
});
```

```
/** @type {import('tailwindcss').Config} */
export default {
  content: ["./index.html", "./src/**/*.{js,jsx}"],
  theme: {
    extend: {
      colors: {
        brand: {
          orange: "#e85d04",
          "orange-hover": "#cf5304",
          green: "#76b82a",
          "green-hover": "#689d25",
          ink: "#1a1a1a",
          muted: "#6b7280",
          surface: "#f6f7f8",
        },
      },
    },
  },
  plugins: [],
};
```

```
export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

```
<!doctype html>
<html lang="uk">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>PC Shop</title>
  </head>
  <body class="bg-brand-surface text-brand-ink antialiased">
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

```
import { Navigate, Route, Routes } from "react-router-dom";
import Layout from "./components/Layout.jsx";
import AdminProducts from "./pages/AdminProducts.jsx";
import Cart from "./pages/Cart.jsx";
import Checkout from "./pages/Checkout.jsx";
import Configurator from "./pages/Configurator.jsx";
import Home from "./pages/Home.jsx";
import Login from "./pages/Login.jsx";
import OrderDetail from "./pages/OrderDetail.jsx";
import Orders from "./pages/Orders.jsx";
import Product from "./pages/Product.jsx";
import Profile from "./pages/Profile.jsx";
import Register from "./pages/Register.jsx";

export default function App() {
  return (
    <Routes>
      <Route element={<Layout />}>
        <Route path="/" element={<Home />} />
        <Route path="/product/:slug" element={<Product />} />
        <Route path="/configurator" element={<Configurator />} />
        <Route path="/cart" element={<Cart />} />
        <Route path="/checkout" element={<Checkout />} />
        <Route path="/orders" element={<Orders />} />
        <Route path="/orders/:id" element={<OrderDetail />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/admin/products" element={<AdminProducts />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="*" element={<Navigate to="/" replace />} />
      </Route>
    </Routes>
  );
}

export async function api(path, options = {}) {
  const headers = { "Content-Type": "application/json", ...options.headers };
  const res = await fetch(path, { credentials: "include", ...options, headers });
  const data = await res.json().catch(() => ({}));
  if (!res.ok) {
    const msg = data.error || res.statusText || "Помилка запиту";
    const err = new Error(msg);
    err.status = res.status;
    throw err;
  }
  return data;
}

import { useEffect, useState } from "react";
import { Link, Outlet, useLocation } from "react-router-dom";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";
import { CART_UPDATE_EVENT } from "../lib/cartEvents.js";

export default function Layout() {
  const { user, loading, logout } = useAuth();
  const location = useLocation();
  const [cartCount, setCartCount] = useState(null);

  useEffect(() => {
    if (!user) {
      setCartCount(null);
      return undefined;
    }

    let cancelled = false;
    async function loadCount() {
      try {
        const d = await api("/api/cart");
        if (!cancelled) setCartCount(d.items?.length ?? 0);
      } catch {
        if (!cancelled) setCartCount(0);
      }
    }

    loadCount();
    const onUpdate = () => {
      loadCount();
```

```
  };
  window.addEventListener(CART_UPDATE_EVENT, onUpdate);
  return () => {
    cancelled = true;
    window.removeEventListener(CART_UPDATE_EVENT, onUpdate);
  };
}, [user, location.pathname]);

return (
  <div className="min-h-screen flex flex-col">
    <header className="bg-brand-orange text-white shadow">
      <div className="mx-auto max-w-5xl px-4 py-3 flex flex-wrap items-center justify-between gap-3">
        <div className="flex items-center gap-5">
          <Link to="/" className="font-semibold text-lg hover:opacity-90">
            PC Shop
          </Link>
          <nav className="flex flex-wrap items-center gap-3 text-sm opacity-95">
            <Link to="/" className="hover:underline">
              Каталог
            </Link>
            <Link to="/configurator" className="hover:underline">
              Конфігуратор
            </Link>
            {user && (
              <>
                <Link to="/cart" className="hover:underline">
                  Кошик{cartCount != null && cartCount > 0 ? ` (${cartCount})` : ""}
                </Link>
                <Link to="/orders" className="hover:underline">
                  Замовлення
                </Link>
                <Link to="/profile" className="hover:underline">
                  Профіль
                </Link>
                {user.role === "admin" && (
                  <Link to="/admin/products" className="hover:underline font-medium">
                    Адмін
                  </Link>
                )}
              </>
            )}
          </nav>
        </div>
        <nav className="flex items-center gap-3 text-sm">
          {!loading && user && (
            <>
              <span className="opacity-90 truncate max-w-[200px]" title={user.email}>
                {user.full_name || user.email}
              </span>
              <button
                type="button"
                onClick={() => logout()}
                className="rounded bg-white/15 px-3 py-1 hover:bg-white/25"
              >
                Вийти
              </button>
            </>
          )}
          {!loading && !user && (
            <>
              <Link to="/login" className="hover:underline">
                Вхід
              </Link>
              <Link
                to="/register"
                className="rounded bg-white/15 px-3 py-1 hover:bg-white/25 font-medium"
              >
                Реєстрація
              </Link>
            </>
          )}
        </nav>
      </div>
    </header>

    <main className="mx-auto flex w-full min-h-0 max-w-5xl flex-1 flex-col px-4 py-8 sm:px-5 sm:py-10">
      <Outlet />
    </main>
```

```
      <footer className="border-t border-gray-200 bg-gray-50/80 py-5 text-center text-xs text-brand-muted">
        PC Shop
      </footer>
    </div>
  );
}

import { createContext, useCallback, useContext, useEffect, useMemo, useState } from "react";
import { api } from "../api";

const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const refresh = useCallback(async () => {
    try {
      const data = await api("/api/auth/me");
      setUser(data.user);
    } catch {
      setUser(null);
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    refresh();
  }, [refresh]);

  const login = useCallback(async (body) => {
    const data = await api("/api/auth/login", { method: "POST", body: JSON.stringify(body) });
    setUser(data.user);
    return data.user;
  }, []);

  const register = useCallback(async (body) => {
    const data = await api("/api/auth/register", { method: "POST", body: JSON.stringify(body) });
    setUser(data.user);
    return data.user;
  }, []);

  const logout = useCallback(async () => {
    await api("/api/auth/logout", { method: "POST", body: "{}" });
    setUser(null);
  }, []);

  const updateProfile = useCallback(async (body) => {
    const data = await api("/api/auth/profile", { method: "PATCH", body: JSON.stringify(body) });
    setUser(data.user);
    return data.user;
  }, []);

  const value = useMemo(
    () => ({ user, loading, login, register, logout, refresh, updateProfile }),
    [user, loading, login, register, logout, refresh, updateProfile]
  );

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

export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) throw new Error("useAuth поза AuthProvider");
  return ctx;
}

import { useCallback, useEffect, useState } from "react";
import toast from "react-hot-toast";
import { api } from "../api";
import { CART_UPDATE_EVENT } from "../lib/cartEvents.js";

export function useInCart(user) {
  const [inCartIds, setInCartIds] = useState(() => new Set());
  const [addingToCartId, setAddingToCartId] = useState(null);

  useEffect(() => {
    if (!user) {
```

```
      setInCartIds(new Set());
      return undefined;
    }
    let cancelled = false;
    async function loadCartIds() {
      try {
        const d = await api("/api/cart");
        if (!cancelled) {
          setInCartIds(new Set((d.items || []).map((i) => i.product_id)));
        }
      } catch {
        if (!cancelled) setInCartIds(new Set());
      }
    }
    loadCartIds();
    const onUpdate = () => loadCartIds();
    window.addEventListener(CART_UPDATE_EVENT, onUpdate);
    return () => {
      cancelled = true;
      window.removeEventListener(CART_UPDATE_EVENT, onUpdate);
    };
  }, [user]);

  const addToCart = useCallback(async (productId) => {
    setAddingToCartId(productId);
    try {
      const data = await api("/api/cart/items", {
        method: "POST",
        body: JSON.stringify({ product_id: productId, quantity: 1, from_catalog: true }),
      });
      if (data.alreadyInCart) {
        toast("Цей товар уже у кошику – змініть кількість у кошику", { icon: "ℹ️" });
      } else {
        toast.success("Товар додано в кошик");
      }
      setInCartIds((prev) => new Set(prev).add(productId));
      window.dispatchEvent(new Event(CART_UPDATE_EVENT));
    } catch (e) {
      toast.error(e.message || "Не вдалося додати в кошик");
    } finally {
      setAddingToCartId(null);
    }
  }, []);

  return { inCartIds, addingToCartId, addToCart };
}

export const CART_UPDATE_EVENT = "pcshop-cart-update";

const LEGACY = {
  items: "pcshop_configurator_items",
  a: "pcshop_configurator_slot_a",
  b: "pcshop_configurator_slot_b",
};

const PREFIX = {
  items: "pcshop_configurator_items_",
  a: "pcshop_configurator_slot_a_",
  b: "pcshop_configurator_slot_b_",
};

function suffixFromUser(user) {
  if (user == null) return "guest";
  const id = user.id;
  if (id === null || id === undefined || id === "") return "guest";
  const n = Number(id);
  return Number.isFinite(n) ? `u${n}` : `u${String(id)}`;
}

function keyItems(suf) {
  return `${PREFIX.items}${suf}`;
}
function keySlot(slot, suf) {
  return slot === "b" ? `${PREFIX.b}${suf}` : `${PREFIX.a}${suf}`;
}

function sameId(a, b) {
  return a != null && b != null && String(a) === String(b);
}
```

```
function normalizeItemId(id) {
  if (id === null || id === undefined) return id;
  const n = Number(id);
  return id !== "" && String(id).trim() !== "" && Number.isFinite(n) ? n : id;
}

function readJson(key, fallback) {
  try {
    const raw = localStorage.getItem(key);
    if (!raw) return fallback;
    return JSON.parse(raw);
  } catch {
    return fallback;
  }
}

function readItemsWithMigrate(suf) {
  const k = keyItems(suf);
  const raw = readJson(k, null);
  const arr = Array.isArray(raw) ? raw : null;
  if (suf === "guest" && (arr == null || arr.length === 0)) {
    const legacy = readJson(LEGACY.items, null);
    if (Array.isArray(legacy) && legacy.length) {
      try {
        localStorage.setItem(k, JSON.stringify(legacy));
        localStorage.removeItem(LEGACY.items);
      } catch {
        // ignore
      }
      return legacy;
    }
  }
  return Array.isArray(arr) ? arr : [];
}

function readSlotWithMigrate(slot, suf) {
  const k = keySlot(slot, suf);
  const raw = readJson(k, null);
  const arr = Array.isArray(raw) ? raw : null;
  if (suf === "guest" && (arr == null || arr.length === 0) && (slot === "a" || slot === "b")) {
    const legKey = slot === "a" ? LEGACY.a : LEGACY.b;
    const legacy = readJson(legKey, null);
    if (Array.isArray(legacy) && legacy.length) {
      try {
        localStorage.setItem(k, JSON.stringify(legacy));
        localStorage.removeItem(legKey);
      } catch {
        // ignore
      }
      return legacy;
    }
  }
  return Array.isArray(arr) ? arr : [];
}

export function getConfiguratorItems(user) {
  return readItemsWithMigrate(suffixFromUser(user));
}

export function setConfiguratorItems(items, user) {
  localStorage.setItem(keyItems(suffixFromUser(user)), JSON.stringify(items));
}

function normalizeItem(item) {
  const { id, name, price, category_slug, specs } = item;
  const o = {
    id: normalizeItemId(id),
    name,
    price,
    category_slug,
  };
  if (specs && typeof specs === "object" && !Array.isArray(specs)) o.specs = specs;
  return o;
}

export function addToConfiguratorDraft(item, user) {
  const n = normalizeItem(item);
  const list = getConfiguratorItems(user).filter((x) => !sameId(x.id, n.id));
```

```
  list.push(n);
  setConfiguratorItems(list, user);
}

export function removeConfiguratorItem(id, user) {
  setConfiguratorItems(
    getConfiguratorItems(user).filter((x) => !sameId(x.id, id)),
    user
  );
}

export function clearConfigurator(user) {
  try {
    localStorage.removeItem(keyItems(suffixFromUser(user)));
  } catch {
    // ignore
  }
}

export function getSlotItems(slot, user) {
  return readSlotWithMigrate(slot, suffixFromUser(user));
}

export function setSlotItems(slot, items, user) {
  localStorage.setItem(keySlot(slot, suffixFromUser(user)), JSON.stringify(items));
}

export function isSameBuild(itemsA, itemsB) {
  if (!Array.isArray(itemsA) || !Array.isArray(itemsB)) return false;
  if (itemsA.length === 0 && itemsB.length === 0) return true;
  if (itemsA.length !== itemsB.length) return false;
  const snap = (items) =>
    [...items]
      .map((x) => ({
        id: normalizeItemId(x.id),
        name: String(x.name || ""),
        price: Number(x.price) || 0,
        category_slug: String(x.category_slug || ""),
        specs: x.specs && typeof x.specs === "object" ? x.specs : null,
      }))
      .sort((a, b) => String(a.id).localeCompare(String(b.id), undefined, { numeric: true }));
  return JSON.stringify(snap(itemsA)) === JSON.stringify(snap(itemsB));
}

export function tryCopyCurrentToSlot(slot, user) {
  const other = slot === "b" ? "a" : "b";
  const current = getConfiguratorItems(user);
  const inOther = getSlotItems(other, user);
  if (inOther.length > 0 && isSameBuild(current, inOther)) {
    return false;
  }
  setSlotItems(slot, current, user);
  return true;
}

export function clearSlot(slot, user) {
  try {
    localStorage.removeItem(keySlot(slot, suffixFromUser(user)));
  } catch {
    // ignore
  }
}

import { isSameBuild } from "./configurator.js";

const CAT = {
  CPU: "cpu",
  MB: "motherboard",
  RAM: "ram",
  GPU: "gpu",
  STORAGE: "storage",
  PSU: "psu",
};

function firstByCategory(items, slug) {
  return items.find((x) => x.category_slug === slug) || null;
}

export function checkCompatibility(items) {
```

```
  const pos = [];
  const neg = [];
  const cpu = firstByCategory(items, CAT.CPU);
  const mb = firstByCategory(items, CAT.MB);
  const ram = firstByCategory(items, CAT.RAM);

  if (cpu && mb) {
    const cSock = cpu.specs && cpu.specs.socket;
    const mSock = mb.specs && mb.specs.socket;
    if (cSock && mSock) {
      if (String(cSock) !== String(mSock)) {
        neg.push("Процесор і материнська плата не підходять одна до одної (різна «посадка»).");
      } else {
        pos.push("Процесор і плата підходять одна до одної – в одній збірці вони можуть разом.");
      }
    } else {
      pos.push("У карток товарів мало деталей – автоперевірка неповна, перегляньте опис на сайті.");
    }
  } else if (cpu || mb) {
    pos.push(cpu ? "Щоб перевірити пару «процесор + плата», додайте в список материнську плату." : "Щоб
перевірити, додайте в список процесор.");
  }

  if (ram && mb) {
    const rT = ram.specs && ram.specs.ramType;
    const mT = mb.specs && mb.specs.ramType;
    if (rT && mT) {
      if (String(rT) !== String(mT)) {
        neg.push("Пам’ять не пасує до плати: у товару плати й у пам’яті має бути один і той самий тип, що в
картці товару (подивіться в опис).");
      } else {
        pos.push("Пам’ять і плата: один тип – підходить разом.");
      }
    } else {
      pos.push("Типу пам’яті в картці не видно – підказка з пам’яттю обмежена.");
    }
  } else {
    if (ram && !mb) pos.push("Щоб перевірити пару «пам’ять + плата», додайте в список материнську плату.");
    if (mb && !ram) pos.push("Щоб перевірити пам’ять, додайте в список модулі ОЗП.");
  }

  if (!cpu && !mb && !ram) {
    return {
      ok: true,
      level: "info",
      lines: ["Щоб отримати поради щодо сумісності, додайте хоча б процесор, материнську плату й модулі
пам’яті."],
    };
  }

  if (neg.length) {
    return { ok: false, level: "warn", lines: [...neg, ...pos] };
  }

  const hasOkHint = pos.some((l) => l.includes("в одній збірці") || l.includes("підходить разом"));
  return {
    ok: true,
    level: hasOkHint ? "ok" : "info",
    lines: pos.length ? pos : ["Додайте процесор, материнську плату й пам’ять, щоб з’явились підказки."],
  };
}

function metricsFromItems(items) {
  const cpu = firstByCategory(items, CAT.CPU);
  const rams = items.filter((i) => i.category_slug === CAT.RAM);
  const gpus = items.filter((i) => i.category_slug === CAT.GPU);
  const storage = items.filter((i) => i.category_slug === CAT.STORAGE);

  const vram = Math.max(0, ...gpus.map((g) => Number(g.specs && g.specs.vramGb) || 0));
  const ramGb = rams.reduce((s, r) => s + (Number(r.specs && r.specs.sizeGb) || 0), 0);
  const ramMhz = Math.max(0, ...rams.map((r) => Number(r.specs && r.specs.speedMhz) || 0));
  const cpuTdp = Number((cpu && cpu.specs && cpu.specs.tdpW) || 0);
  const hasNvme = storage.some(
    (s) => String((s.specs && s.specs.interface) || "").toLowerCase().includes("nvme")
  );
  const total = items.reduce((s, i) => s + Number(i.price || 0), 0);

  return { vram, ramGb, ramMhz, cpuTdp, hasNvme, total };
}
```

```
function scores(m) {
  return {
    games: m.vram * 120 + m.ramMhz * 0.15 + m.cpuTdp * 0.4,
    dev: m.ramGb * 22 + m.cpuTdp * 0.2 + (m.hasNvme ? 80 : 0),
    video: m.ramGb * 18 + m.vram * 100 + (m.hasNvme ? 50 : 0),
  };
}

export function compareBuilds(itemsA, itemsB) {
  const empty = !itemsA.length && !itemsB.length;
  if (empty) {
    return { title: "Порівняння", body: ["Заповніть A і B кнопками «Поточне → A / B»."] };
  }
  if (!itemsA.length) {
    return { title: "Порівняння", body: ["Слот A порожній."] };
  }
  if (!itemsB.length) {
    return { title: "Порівняння", body: ["Слот B порожній."] };
  }

  if (isSameBuild(itemsA, itemsB)) {
    return {
      title: "Порівняння",
      body: ["A і B однакові. Змініть одну зі збірок, щоб порівняти."],
    };
  }

  const ma = metricsFromItems(itemsA);
  const mb = metricsFromItems(itemsB);
  const sa = scores(ma);
  const sb = scores(mb);
  const body = [`A: ${ma.total.toFixed(0)} грн · B: ${mb.total.toFixed(0)} грн`];

  function line(key, forWhat) {
    const d = sa[key] - sb[key];
    if (Math.abs(d) < 1) {
      return `Для ${forWhat} – варіанти близькі.`;
    }
    const w = d > 0 ? "A" : "B";
    return `Варіант ${w} краще підходить для ${forWhat}.`;
  }

  body.push(line("games", "ігор"));
  body.push(line("dev", "програмування"));
  body.push(line("video", "відео та монтажу"));

  if (mb.total < ma.total) {
    body.push("B дешевший – варіант, коли важлива ціна чи вистачає такої простішої збірки.");
  } else if (ma.total < mb.total) {
    body.push("A дешевший – варіант, коли важлива ціна чи вистачає такої простішої збірки.");
  }

  return { title: "Порівняння A і B", body };
}

export const ORDER_STATUS_UA = {
  pending: "Очікує підтвердження",
  processing: "В обробці",
  shipped: "Відправлено",
  delivered: "Доставлено",
  cancelled: "Скасовано",
};

export function orderStatusLabel(status) {
  return ORDER_STATUS_UA[status] || status;
}

const MAP = {
  cpu: "Процесор",
  motherboard: "Материнська плата",
  ram: "Пам'ять",
  gpu: "Відеокарта",
  storage: "Накопичувач",
  psu: "Блок живлення",
};

export function partCategoryLabel(slug) {
  if (!slug) return "";
```

```
  return MAP[slug] || slug;
}

export function deliveryAddressIsPlausible(s) {
  const t = String(s || "").trim();
  const meaningful = t.match(/\p{L}|\p{N}/gu);
  return meaningful != null && meaningful.length >= 5;
}

import { useCallback, useEffect, useState } from "react";
import toast from "react-hot-toast";
import { Link } from "react-router-dom";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

function buildMockProductFields(category) {
  const idPart = Date.now().toString(36);
  const slugKey = category?.slug;

  const bySlug = {
    cpu: {
      name: "AMD Ryzen 5 7600",
      slug: `amd-ryzen-5-7600-${idPart}`,
      description:
        "6 ядер / 12 потоків, сокет AM5, вбудована графіка Radeon RDNA2. У комплекті – кулер AMD Wraith
Stealth.",
      price: "8299",
      stock: "14",
      image_url: "https://placehold.co/480x360/e85d04/ffffff?text=Ryzen+7600",
      specs: {
        socket: "AM5",
        tdpW: 65,
      },
    },
    motherboard: {
      name: "ASUS B650M-PLUS WIFI",
      slug: `asus-b650m-plus-wifi-${idPart}`,
      description: "Материнська плата mATX, сокет AM5, DDR5, Wi-Fi 6E, PCIe 4.0.",
      price: "6299",
      stock: "11",
      image_url: "https://placehold.co/480x360/2e7d32/ffffff?text=B650M",
      specs: {
        socket: "AM5",
        ramType: "DDR5",
        formFactor: "mATX",
      },
    },
    ram: {
      name: "Kingston Fury Beast 32GB DDR5 5600",
      slug: `kingston-fury-beast-32-ddr5-5600-${idPart}`,
      description: "Комплект 2×16 ГБ, профілі AMD EXPO та Intel XMP, швидкість 5600 МГц.",
      price: "3699",
      stock: "22",
      image_url: "https://placehold.co/480x360/6a1b9a/ffffff?text=DDR5+32",
      specs: {
        ramType: "DDR5",
        sizeGb: 32,
        speedMhz: 5600,
      },
    },
    gpu: {
      name: "NVIDIA GeForce RTX 4060 8GB",
      slug: `nvidia-rtx-4060-8gb-${idPart}`,
      description: "8 ГБ GDDR6, енергоспоживання до 115 Вт. Підходить для ігор у Full HD на високих
налаштуваннях.",
      price: "11299",
      stock: "7",
      image_url: "https://placehold.co/480x360/222222/76b900?text=RTX+4060",
      specs: {
        vramGb: 8,
        tdpW: 115,
      },
    },
    storage: {
```

```
      name: "Samsung 990 PRO 1TB NVMe",
      slug: `samsung-990-pro-1tb-${idPart}`,
      description: "SSD M.2 NVMe PCIe 4.0, до 7450 МБ/с читання, ємність 1 ТБ.",
      price: "5499",
      stock: "19",
      image_url: "https://placehold.co/480x360/1428a0/ffffff?text=990+PRO",
      specs: {
        interface: "NVMe",
        capacityGb: 1000,
      },
    },
    psu: {
      name: "DeepCool PK750D 750W",
      slug: `deepcool-pk750d-750w-${idPart}`,
      description: "Блок живлення 750 Вт, сертифікат 80+ Bronze, надійна платформа для середнього класу ПК.",
      price: "2499",
      stock: "16",
      image_url: "https://placehold.co/480x360/455a64/ffffff?text=750W",
      specs: {
        wattage: 750,
        cert: "80+ Bronze",
      },
    },
  };

  const base = bySlug[slugKey] || bySlug.cpu;
  return {
    category_id: category?.id != null ? String(category.id) : "",
    name: base.name,
    slug: base.slug,
    description: base.description,
    price: base.price,
    stock: base.stock,
    image_url: base.image_url,
    specsText: JSON.stringify(base.specs, null, 2),
  };
}

async function uploadImage(file) {
  const fd = new FormData();
  fd.append("file", file);
  const res = await fetch("/api/admin/upload", {
    method: "POST",
    credentials: "include",
    body: fd,
  });
  const data = await res.json().catch(() => ({}));
  if (!res.ok) {
    throw new Error(data.error || res.statusText);
  }
  return data.url;
}

export default function AdminProducts() {
  const { user, loading } = useAuth();
  const [categories, setCategories] = useState([]);
  const [items, setItems] = useState([]);
  const [listLoading, setListLoading] = useState(true);
  const [editingId, setEditingId] = useState(null);
  const [category_id, setCategoryId] = useState("");
  const [name, setName] = useState("");
  const [slug, setSlug] = useState("");
  const [description, setDescription] = useState("");
  const [price, setPrice] = useState("");
  const [stock, setStock] = useState("");
  const [image_url, setImageUrl] = useState("");
  const [specsText, setSpecsText] = useState("{}");
  const [busy, setBusy] = useState(false);
  const [uploading, setUploading] = useState(false);

  const loadList = useCallback(async () => {
    setListLoading(true);
    try {
      const [catData, prodData] = await Promise.all([api("/api/catalog/categories"),
api("/api/admin/products")]);
      setCategories(catData.categories || []);
      setItems(prodData.items || []);
    } catch (e) {
      toast.error(e.message || "Помилка завантаження");
```

```
  } finally {
    setListLoading(false);
  }
}, []);

useEffect(() => {
  if (!user || user.role !== "admin") return;
  loadList();
}, [user, loadList]);

const applyNewProductTemplate = useCallback((category) => {
  if (!category) return;
  const m = buildMockProductFields(category);
  setCategoryId(m.category_id);
  setName(m.name);
  setSlug(m.slug);
  setDescription(m.description);
  setPrice(m.price);
  setStock(m.stock);
  setImageUrl(m.image_url);
  setSpecsText(m.specsText);
}, []);

const resetForm = useCallback(() => {
  setEditingId(null);
  if (!categories.length) {
    setCategoryId("");
    setName("");
    setSlug("");
    setDescription("");
    setPrice("");
    setStock("");
    setImageUrl("");
    setSpecsText("{}");
    return;
  }
  applyNewProductTemplate(categories[0]);
}, [categories, applyNewProductTemplate]);

useEffect(() => {
  if (!categories.length || editingId != null) return;
  resetForm();
}, [categories, editingId, resetForm]);

function startEdit(row) {
  setEditingId(row.id);
  setCategoryId(String(row.category_id));
  setName(row.name);
  setSlug(row.slug);
  setDescription(row.description);
  setPrice(String(row.price));
  setStock(String(row.stock));
  setImageUrl(row.image_url || "");
  setSpecsText(row.specs ? JSON.stringify(row.specs, null, 2) : "{}");
  window.scrollTo({ top: 0, behavior: "smooth" });
}

async function onSubmit(e) {
  e.preventDefault();
  let specsPayload = null;
  const trimmed = specsText.trim();
  if (trimmed && trimmed !== "{}") {
    try {
      specsPayload = JSON.parse(trimmed);
    } catch {
      toast.error("Поле «Характеристики» має бути валідним JSON");
      return;
    }
  }

  setBusy(true);
  try {
    if (editingId) {
      await api(`/api/admin/products/${editingId}`, {
        method: "PATCH",
        body: JSON.stringify({
          category_id: Number(category_id),
          name: name.trim(),
          slug: slug.trim().toLowerCase(),
```

```
            description: description.trim(),
            price: Number(price),
            stock: parseInt(stock, 10),
            image_url: image_url.trim(),
            specs: specsPayload,
          }),
        });
        toast.success("Товар оновлено");
      } else {
        await api("/api/admin/products", {
          method: "POST",
          body: JSON.stringify({
            category_id: Number(category_id),
            name: name.trim(),
            slug: slug.trim().toLowerCase(),
            description: description.trim(),
            price: Number(price),
            stock: parseInt(stock, 10),
            image_url: image_url.trim(),
            specs: specsPayload,
          }),
        });
        toast.success("Товар створено");
      }
      await loadList();
      resetForm();
    } catch (err) {
      toast.error(err.message || "Помилка");
    } finally {
      setBusy(false);
    }
  }

  async function onPickFile(e) {
    const file = e.target.files?.[0];
    e.target.value = "";
    if (!file) return;
    setUploading(true);
    try {
      const url = await uploadImage(file);
      setImageUrl(url);
      toast.success("Зображення завантажено");
    } catch (err) {
      toast.error(err.message || "Не вдалося завантажити");
    } finally {
      setUploading(false);
    }
  }

  async function onDelete(id) {
    if (!window.confirm("Видалити товар з каталогу? Якщо він уже купувався, він буде лише знятий з
продажу.")) return;
    try {
      const data = await api(`/api/admin/products/${id}`, { method: "DELETE" });
      if (data.archived) {
        if (data.alreadyArchived) {
          toast.success("Товар уже знято з продажу");
        } else {
          toast.success("Знято з продажу (товар був у замовленнях – рядок збережено для історії)");
        }
      } else {
        toast.success("Видалено");
      }
      await loadList();
      if (editingId === id) resetForm();
    } catch (err) {
      toast.error(err.message || "Помилка");
    }
  }

  if (!loading && (!user || user.role !== "admin")) {
    return (
      <div>
        <h1 className="text-xl font-semibold text-brand-ink mb-2">Доступ заборонено</h1>
        <p className="text-brand-muted mb-4">Ця сторінка лише для адміністратора.</p>
        <Link to="/" className="text-brand-orange font-medium hover:underline">
          На головну
        </Link>
      </div>
```

```
    );
  }

  if (loading || listLoading) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  return (
    <div>
      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Адмін: товари</h1>
      <p className="text-sm text-brand-muted mb-6">Створення, редагування, видалення та завантаження
зображення.</p>

      <div className="rounded-lg border border-gray-200 bg-white p-5 shadow-sm mb-10">
        <h2 className="text-lg font-semibold text-brand-ink mb-4">{editingId ? `Редагування #${editingId}` :
"Новий товар"}</h2>
        <form onSubmit={onSubmit} className="max-w-2xl space-y-3">
          <div className="grid gap-3 sm:grid-cols-2">
            <div>
              <label className="block text-xs text-brand-muted mb-1">Категорія *</label>
              <select
                required
                value={category_id}
                onChange={(e) => {
                  const v = e.target.value;
                  setCategoryId(v);
                  if (editingId != null) return;
                  const cat = categories.find((c) => String(c.id) === v);
                  if (cat) applyNewProductTemplate(cat);
                }}
                className="w-full rounded border border-gray-300 px-3 py-2 text-sm"
              >
                {categories.map((c) => (
                  <option key={c.id} value={c.id}>
                    {c.name}
                  </option>
                ))}
              </select>
            </div>
            <div>
              <label className="block text-xs text-brand-muted mb-1">Slug * (латиниця, дефіс)</label>
              <input
                required
                value={slug}
                onChange={(e) => setSlug(e.target.value)}
                className="w-full rounded border border-gray-300 px-3 py-2 text-sm font-mono"
                placeholder="my-product-slug"
              />
            </div>
          </div>
          <div>
            <label className="block text-xs text-brand-muted mb-1">Назва *</label>
            <input required value={name} onChange={(e) => setName(e.target.value)} className="w-full rounded
border border-gray-300 px-3 py-2 text-sm" />
          </div>
          <div>
            <label className="block text-xs text-brand-muted mb-1">Опис *</label>
            <textarea
              required
              rows={3}
              value={description}
              onChange={(e) => setDescription(e.target.value)}
              className="w-full rounded border border-gray-300 px-3 py-2 text-sm"
            />
          </div>
          <div className="grid gap-3 sm:grid-cols-2">
            <div>
              <label className="block text-xs text-brand-muted mb-1">Ціна (UAH) *</label>
              <input required type="number" min="0" step="0.01" value={price} onChange={(e) =>
setPrice(e.target.value)} className="w-full rounded border border-gray-300 px-3 py-2 text-sm" />
            </div>
            <div>
              <label className="block text-xs text-brand-muted mb-1">Залишок *</label>
              <input required type="number" min="0" value={stock} onChange={(e) => setStock(e.target.value)}
className="w-full rounded border border-gray-300 px-3 py-2 text-sm" />
            </div>
          </div>
          <div>
            <label className="block text-xs text-brand-muted mb-1">URL зображення</label>
```

```
            <div className="flex flex-wrap gap-2">
              <input
                value={image_url}
                onChange={(e) => setImageUrl(e.target.value)}
                className="min-w-[200px] flex-1 rounded border border-gray-300 px-3 py-2 text-sm"
                placeholder="https://… або завантажте файл"
              />
              <label className="inline-flex items-center justify-center rounded border border-gray-300 px-3
py-2 text-sm cursor-pointer hover:bg-gray-50">
                {uploading ? "…" : "Файл"}
                <input type="file" accept="image/jpeg,image/png,image/webp,image/gif" className="hidden"
onChange={onPickFile} disabled={uploading} />
              </label>
            </div>
            {image_url && (
              <div className="mt-2 h-24 w-32 rounded border border-gray-200 overflow-hidden bg-gray-50">
                <img src={image_url} alt="" className="h-full w-full object-cover" />
              </div>
            )}
          </div>
          <div>
            <label className="block text-xs text-brand-muted mb-1">Характеристики (JSON)</label>
            <textarea rows={10} value={specsText} onChange={(e) => setSpecsText(e.target.value)}
className="w-full rounded border border-gray-300 px-3 py-2 text-sm font-mono text-xs" />
          </div>
          <div className="flex flex-wrap gap-2 pt-2">
            <button
              type="submit"
              disabled={busy}
              className="rounded bg-brand-orange px-4 py-2 text-sm font-medium text-white hover:bg-brand-
orange-hover disabled:opacity-50"
            >
              {busy ? "Збереження…" : editingId ? "Зберегти зміни" : "Створити товар"}
            </button>
            {editingId && (
              <button type="button" onClick={resetForm} className="rounded border border-gray-300 px-4 py-2
text-sm hover:bg-gray-50">
                Скасувати редагування
              </button>
            )}
          </div>
        </form>
      </div>

      <h2 className="text-lg font-semibold text-brand-ink mb-3">Список товарів</h2>
      <div className="overflow-x-auto rounded-lg border border-gray-200 bg-white shadow-sm">
        <table className="w-full text-sm">
          <thead className="bg-gray-50 text-left text-xs text-brand-muted uppercase">
            <tr>
              <th className="px-3 py-2">ID</th>
              <th className="px-3 py-2">Назва</th>
              <th className="px-3 py-2">Категорія</th>
              <th className="px-3 py-2">Ціна</th>
              <th className="px-3 py-2">Склад</th>
              <th className="px-3 py-2 w-28" />
            </tr>
          </thead>
          <tbody className="divide-y divide-gray-100">
            {items.map((row) => (
              <tr key={row.id}>
                <td className="px-3 py-2 font-mono text-xs">{row.id}</td>
                <td className="px-3 py-2">
                  <div className="flex flex-wrap items-center gap-2">
                    {row.is_active === false ? (
                      <span className="text-brand-muted" title={row.slug}>
                        {row.name}
                      </span>
                    ) : (
                      <Link to={`/product/${row.slug}`} className="text-brand-orange hover:underline">
                        {row.name}
                      </Link>
                    )}
                    {row.is_active === false && (
                      <span className="rounded bg-gray-200 px-1.5 py-0.5 text-[10px] font-medium uppercase
text-brand-muted">
                        знято
                      </span>
                    )}
                  </div>
```

```
                </td>
                <td className="px-3 py-2 text-brand-muted">{row.category_name}</td>
                <td className="px-3 py-2 whitespace-nowrap">{money(row.price)}</td>
                <td className="px-3 py-2">{row.stock}</td>
                <td className="px-3 py-2 whitespace-nowrap">
                  <button type="button" onClick={() => startEdit(row)} className="text-brand-orange text-xs
hover:underline mr-2">
                    Змінити
                  </button>
                  <button type="button" onClick={() => onDelete(row.id)} className="text-red-600 text-xs
hover:underline">
                    Видалити
                  </button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
}

import { useCallback, useEffect, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
import { api } from "../api";
import { CART_UPDATE_EVENT } from "../lib/cartEvents.js";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

export default function Cart() {
  const navigate = useNavigate();
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [updatingId, setUpdatingId] = useState(null);

  const load = useCallback(async () => {
    setLoading(true);
    try {
      const d = await api("/api/cart");
      setData(d);
    } catch (e) {
      toast.error(e.message || "Не вдалося завантажити кошик");
      setData({ items: [], totalQty: 0, totalPrice: 0 });
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    load().then(() => window.dispatchEvent(new Event(CART_UPDATE_EVENT)));
  }, [load]);

  const setQty = useCallback(
    async (productId, quantity) => {
      setUpdatingId(productId);
      try {
        await api(`/api/cart/items/${productId}`, {
          method: "PATCH",
          body: JSON.stringify({ quantity }),
        });
        await load();
        window.dispatchEvent(new Event(CART_UPDATE_EVENT));
      } catch (e) {
        toast.error(e.message || "Помилка");
      } finally {
        setUpdatingId(null);
      }
    },
    [load]
  );

  const remove = useCallback(
    async (productId) => {
      setUpdatingId(productId);
```

```
      try {
        await api(`/api/cart/items/${productId}`, { method: "DELETE" });
        await load();
        window.dispatchEvent(new Event(CART_UPDATE_EVENT));
        toast.success("Прибрано з кошика");
      } catch (e) {
        toast.error(e.message || "Помилка");
      } finally {
        setUpdatingId(null);
      }
    },
    [load]
  );

  if (loading && !data) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  const items = data?.items || [];

  return (
    <div>
      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Кошик</h1>
      <p className="text-sm text-brand-muted mb-6">Зміна кількості та підсумок. У картці товару – залишок на
складі.</p>

      {items.length === 0 ? (
        <p className="text-brand-muted mb-4">
          Порожньо. Перейдіть у{" "}
          <Link to="/" className="text-brand-orange font-medium hover:underline">
            каталог
          </Link>
          .
        </p>
      ) : (
        <>
          <div className="rounded-lg border border-gray-200 bg-white shadow-sm overflow-hidden">
            <table className="w-full text-sm">
              <thead className="bg-gray-50 text-left text-xs text-brand-muted uppercase tracking-wide">
                <tr>
                  <th className="px-3 py-2">Товар</th>
                  <th className="px-3 py-2 hidden sm:table-cell">На складі</th>
                  <th className="px-3 py-2 w-32">К-сть</th>
                  <th className="px-3 py-2 text-right">Сума</th>
                  <th className="px-2 py-2 w-10" />
                </tr>
              </thead>
              <tbody className="divide-y divide-gray-100">
                {items.map((row) => (
                  <tr key={row.product_id}>
                    <td className="px-3 py-3">
                      <Link
                        to={`/product/${row.slug}`}
                        className="font-medium text-brand-ink hover:text-brand-orange hover:underline"
                      >
                        {row.name}
                      </Link>
                      <p className="text-xs text-brand-muted sm:hidden">На складі: {row.stock}</p>
                    </td>
                    <td className="px-3 py-3 hidden sm:table-cell text-brand-muted">{row.stock}</td>
                    <td className="px-3 py-3">
                      <div className="flex items-center gap-1">
                        <button
                          type="button"
                          disabled={updatingId === row.product_id || row.quantity <= 1}
                          onClick={() => setQty(row.product_id, row.quantity - 1)}
                          className="h-8 w-8 rounded border border-gray-300 text-lg leading-none hover:bg-
gray-50 disabled:opacity-40"
                        >
                          −
                        </button>
                        <span className="min-w-[2rem] text-center font-medium">{row.quantity}</span>
                        <button
                          type="button"
                          disabled={updatingId === row.product_id || row.quantity >= row.stock}
                          onClick={() => setQty(row.product_id, row.quantity + 1)}
                          className="h-8 w-8 rounded border border-gray-300 text-lg leading-none hover:bg-
gray-50 disabled:opacity-40"
                        >
```

```
                        +
                      </button>
                    </div>
                  </td>
                  <td className="px-3 py-3 text-right font-medium whitespace-
nowrap">{money(row.line_total)}</td>
                  <td className="px-2 py-3">
                    <button
                      type="button"
                      disabled={updatingId === row.product_id}
                      onClick={() => remove(row.product_id)}
                      className="text-red-600 text-xs hover:underline disabled:opacity-40"
                    >
                      Видалити
                    </button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
        <p className="mt-4 text-base">
          Разом: <span className="font-semibold">{money(data?.totalPrice || 0)}</span>
          <span className="text-brand-muted text-sm ml-2">({data?.totalQty} шт.)</span>
        </p>
        <div className="mt-4">
          <button
            type="button"
            onClick={() => navigate("/checkout")}
            className="rounded bg-brand-orange px-5 py-2.5 text-sm font-medium text-white hover:bg-brand-
orange-hover"
          >
            Оформити замовлення
          </button>
        </div>
      </>
    )}
  </div>
);
}

import { useCallback, useEffect, useState } from "react";
import toast from "react-hot-toast";
import { Link, Navigate, useNavigate } from "react-router-dom";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";
import { CART_UPDATE_EVENT } from "../lib/cartEvents.js";
import { deliveryAddressIsPlausible } from "../lib/validateAddress.js";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

export default function Checkout() {
  const { user, loading } = useAuth();
  const navigate = useNavigate();
  const [cart, setCart] = useState(null);
  const [options, setOptions] = useState({ payment_methods: [], shipping_methods: [] });
  const [delivery_address, setDeliveryAddress] = useState("");
  const [payment_method, setPaymentMethod] = useState("");
  const [shipping_method, setShippingMethod] = useState("");
  const [busy, setBusy] = useState(false);

  const load = useCallback(async () => {
    const [cartData, optData] = await Promise.all([api("/api/cart"), api("/api/orders/form-options")]);
    setCart(cartData);
    setOptions(optData);
    setPaymentMethod((prev) => prev || optData.payment_methods?.[0] || "");
    setShippingMethod((prev) => prev || optData.shipping_methods?.[0] || "");
  }, []);

  useEffect(() => {
    if (!user) return;
    (async () => {
      try {
        await load();
      } catch (e) {
        toast.error(e.message || "Помилка");
```

```
      }
    })();
  }, [user, load]);

  async function onSubmit(e) {
    e.preventDefault();
    setBusy(true);
    try {
      const { order } = await api("/api/orders", {
        method: "POST",
        body: JSON.stringify({ delivery_address, payment_method, shipping_method }),
      });
      window.dispatchEvent(new Event(CART_UPDATE_EVENT));
      toast.success("Замовлення оформлено");
      navigate(`/orders/${order.id}`);
    } catch (err) {
      toast.error(err.message || "Помилка оформлення");
    } finally {
      setBusy(false);
    }
  }

  if (!loading && !user) {
    return <Navigate to="/login" replace state={{ from: "/checkout" }} />;
  }

  if (loading || !cart) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  if (!cart.items?.length) {
    return (
      <div>
        <p className="text-brand-muted mb-4">Кошик порожній – додайте товари перед оформленням.</p>
        <Link to="/cart" className="text-brand-orange font-medium hover:underline">
          До кошика
        </Link>
      </div>
    );
  }

  return (
    <div>
      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Оформлення замовлення</h1>
      <p className="text-sm text-brand-muted mb-6">
        Разом до сплати: <span className="font-semibold text-brand-ink">{money(cart.totalPrice)}</span>
({cart.totalQty}{" "}
        шт.)
      </p>

      <form onSubmit={onSubmit} className="max-w-xl space-y-4 rounded-lg border border-gray-200 bg-white p-5
shadow-sm">
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Адреса доставки *</label>
          <textarea
            required
            minLength={10}
            maxLength={500}
            rows={4}
            value={delivery_address}
            onChange={(e) => setDeliveryAddress(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            placeholder="Місто, відділення / вулиця, будинок, квартира…"
          />
          {delivery_address.trim().length >= 10 && !deliveryAddressIsPlausible(delivery_address) && (
            <p className="text-xs text-red-600 mt-1">
              Невірний формат вводу. Потрібні літери або цифри (наприклад, місто, номер відділення).
            </p>
          )}
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Оплата *</label>
          <select
            required
            value={payment_method}
            onChange={(e) => setPaymentMethod(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
```

```
          >
            {options.payment_methods.map((p) => (
              <option key={p} value={p}>
                {p}
              </option>
            ))}
          </select>
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Доставка *</label>
          <select
            required
            value={shipping_method}
            onChange={(e) => setShippingMethod(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
          >
            {options.shipping_methods.map((s) => (
              <option key={s} value={s}>
                {s}
              </option>
            ))}
          </select>
        </div>
        <div className="flex flex-wrap gap-3 pt-2">
          <button
            type="submit"
            disabled={busy || delivery_address.trim().length < 10
|| !deliveryAddressIsPlausible(delivery_address)}
            className="rounded bg-brand-orange px-5 py-2.5 text-sm font-medium text-white hover:bg-brand-
orange-hover disabled:opacity-50"
          >
            {busy ? "Відправка…" : "Підтвердити замовлення"}
          </button>
          <Link to="/cart" className="inline-flex items-center rounded border border-gray-300 px-5 py-2.5
text-sm hover:bg-gray-50">
            Назад до кошика
          </Link>
        </div>
      </form>
    </div>
  );
}

import { useState } from "react";
import toast from "react-hot-toast";
import { Link } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import {
  clearConfigurator,
  clearSlot,
  getConfiguratorItems,
  getSlotItems,
  removeConfiguratorItem,
  tryCopyCurrentToSlot,
} from "../lib/configurator";
import { checkCompatibility, compareBuilds } from "../lib/configuratorAnalysis";
import { partCategoryLabel } from "../lib/partCategoryLabel.js";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

function sumItems(items) {
  return items.reduce((s, x) => s + Number(x.price || 0), 0);
}

function panelClass(level) {
  if (level === "ok") return "border-emerald-200 bg-emerald-50";
  if (level === "warn") return "border-amber-200 bg-amber-50";
  return "border-gray-200 bg-gray-50";
}

export default function Configurator() {
  const { user } = useAuth();
  const [, setRenderTick] = useState(0);
  const refresh = () => setRenderTick((n) => n + 1);
```

```
const items = getConfiguratorItems(user);
const slotA = getSlotItems("a", user);
const slotB = getSlotItems("b", user);

const total = sumItems(items);
const compat = checkCompatibility(items);
const compatA = checkCompatibility(slotA);
const compatB = checkCompatibility(slotB);
const comparison = compareBuilds(slotA, slotB);

return (
  <div>
    <h1 className="text-2xl font-semibold text-brand-ink mb-2">Конфігуратор ПК</h1>
    <p className="text-sm text-brand-muted mb-6 max-w-2xl">
      Підказки щодо поєднання процесора, плати та
      пам’яті. Щоб порівняти два варіанти – збережіть чернетку в A, змініть набір у каталозі, потім у B.
    </p>

    {items.length === 0 ? (
      <p className="text-brand-muted mb-4">
        Порожньо. Додайте деталі з{" "}
        <Link to="/" className="text-brand-orange font-medium hover:underline">
          каталогу
        </Link>
        .
      </p>
    ) : (
      <ul className="rounded-lg border border-gray-200 bg-white divide-y divide-gray-100 shadow-sm mb-4">
        {items.map((row) => (
          <li
            key={`${row.id}-${row.category_slug}`}
            className="flex flex-wrap items-center justify-between gap-3 px-4 py-3"
          >
            <div>
              <p className="font-medium text-brand-ink">{row.name}</p>
              <p className="text-xs text-brand-muted">{partCategoryLabel(row.category_slug)}</p>
            </div>
            <div className="flex items-center gap-3">
              <span className="text-sm font-semibold text-brand-ink">{money(row.price)}</span>
              <button
                type="button"
                onClick={() => {
                  removeConfiguratorItem(row.id, user);
                  refresh();
                }}
                className="text-sm text-red-600 hover:underline"
              >
                Прибрати
              </button>
            </div>
          </li>
        ))}
      </ul>
    )}

    <p className="text-sm mb-4 text-brand-ink">
      Всього: <span className="font-semibold tabular-nums">{money(total)}</span>
    </p>

    <div className={`rounded-lg border p-4 mb-6 text-sm ${panelClass(compat.level)}`}>
      <p className="font-medium text-brand-ink mb-2">Сумісність</p>
      <ul className="list-disc pl-5 space-y-1 text-brand-ink">
        {compat.lines.map((l, i) => (
          <li key={i}>{l}</li>
        ))}
      </ul>
    </div>

    {items.length > 0 && (
      <div className="flex flex-wrap gap-2 mb-8">
        <button
          type="button"
          onClick={() => {
            if (!tryCopyCurrentToSlot("a", user)) {
              toast.error("Варіанти A і B мають відрізнятися. Очистіть B або змініть чернетку.");
              return;
            }
            refresh();
            toast.success("Скопійовано в збірку A");
```

```
            }}
            className="rounded border border-brand-green bg-white px-3 py-2 text-sm text-brand-green
hover:bg-green-50"
          >
            Поточне → A
          </button>
          <button
            type="button"
            onClick={() => {
              if (!tryCopyCurrentToSlot("b", user)) {
                toast.error("Варіанти A і B мають відрізнятися. Очистіть A або змініть чернетку.");
                return;
              }
              refresh();
              toast.success("Скопійовано в збірку B");
            }}
            className="rounded border border-brand-green bg-white px-3 py-2 text-sm text-brand-green
hover:bg-green-50"
          >
            Поточне → B
          </button>
          <button
            type="button"
            onClick={() => {
              clearConfigurator(user);
              refresh();
            }}
            className="rounded border border-gray-300 px-3 py-2 text-sm hover:bg-gray-50"
          >
            Очистити поточне
          </button>
        </div>
      )}

      <section className="mb-8">
        <h2 className="text-lg font-semibold text-brand-ink mb-3">Збірки для порівняння</h2>
        <div className="grid gap-4 md:grid-cols-2">
          {["a", "b"].map((slot) => {
            const data = slot === "a" ? slotA : slotB;
            const c = slot === "a" ? compatA : compatB;
            const t = sumItems(data);
            return (
              <div
                key={slot}
                className={`rounded-lg border p-4 text-sm ${
                  slot === "a" ? "border-blue-200 bg-blue-50/50" : "border-violet-200 bg-violet-50/50"
                }`}
              >
                <p className="font-medium text-brand-ink mb-1">Збірка {slot.toUpperCase()}</p>
                <p className="text-xs text-brand-muted mb-2">
                  {data.length} поз. · {money(t)}
                </p>
                {data.length > 0 && (
                  <p className="text-xs text-brand-ink mb-2 line-clamp-3" title={data.map((d) =>
d.name).join(", ")}>
                    {data.map((d) => d.name).join(" · ")}
                  </p>
                )}
                {data.length === 0 ? (
                  <p className="text-brand-muted">Порожньо. Скопіюйте чернетку кнопкою «Поточне →
{slot.toUpperCase()}».</p>
                ) : (
                  <ul className="text-xs space-y-1 text-brand-ink mb-2 border-t border-gray-200/80 pt-2">
                    {c.lines.map((l, i) => (
                      <li key={i}>
                        {l}
                      </li>
                    ))}
                  </ul>
                )}
                <button
                  type="button"
                  onClick={() => {
                    clearSlot(slot, user);
                    refresh();
                    toast.success(`Збірка ${slot.toUpperCase()} очищена`);
                  }}
                  className="text-xs text-red-600 hover:underline"
                >
```

```
                Очистити {slot.toUpperCase()}
              </button>
            </div>
          );
        })}
      </div>
    </section>

    <section className="rounded-lg border border-gray-200 bg-white p-5 shadow-sm">
      <h2 className="text-lg font-semibold text-brand-ink mb-3">{comparison.title}</h2>
      <ul className="text-sm text-brand-ink space-y-2 list-disc pl-5 marker:text-brand-orange">
        {comparison.body.map((line, i) => (
          <li key={i} className="leading-relaxed pl-0.5">
            {line}
          </li>
        ))}
      </ul>
    </section>
  </div>
 );
}

import { useCallback, useEffect, useMemo, useState } from "react";
import toast from "react-hot-toast";
import { Link, useSearchParams } from "react-router-dom";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";
import { useInCart } from "../hooks/useInCart.js";
import { addToConfiguratorDraft, getConfiguratorItems } from "../lib/configurator";

function money(n) {
 return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

export default function Home() {
 const { user } = useAuth();
 const { inCartIds, addingToCartId, addToCart } = useInCart(user);
 const [params, setParams] = useSearchParams();
 const category = params.get("category") || "";
 const search = params.get("search") || "";
 const page = Math.max(1, parseInt(params.get("page"), 10) || 1);

 const [categories, setCategories] = useState([]);
 const [catalog, setCatalog] = useState({ items: [], total: 0, pages: 1, page: 1, limit: 12 });
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState("");
 const [searchInput, setSearchInput] = useState(search);
 const [cfgBump, setCfgBump] = useState(0);

 const configCount = useMemo(() => getConfiguratorItems(user).length, [cfgBump, user?.id]);

 useEffect(() => {
   setSearchInput(search);
 }, [search]);

 useEffect(() => {
   let cancelled = false;
   (async () => {
     try {
       const data = await api("/api/catalog/categories");
       if (!cancelled) setCategories(data.categories || []);
     } catch {
       if (!cancelled) setCategories([]);
     }
   })();
   return () => {
     cancelled = true;
   };
 }, []);

 useEffect(() => {
   let cancelled = false;
   setLoading(true);
   setError("");
   const q = new URLSearchParams();
   if (category) q.set("category", category);
   if (search) q.set("search", search);
   q.set("page", String(page));
```

```
  q.set("limit", "12");

  (async () => {
    try {
      const data = await api(`/api/catalog/products?${q.toString()}`);
      if (cancelled) return;
      setCatalog(data);
      if (page > data.pages && data.pages >= 1) {
        setParams(
          (prev) => {
            const next = new URLSearchParams(prev);
            next.set("page", String(data.pages));
            return next;
          },
          { replace: true }
        );
      }
    } catch (e) {
      if (!cancelled) setError(e.message || "Помилка завантаження");
    } finally {
      if (!cancelled) setLoading(false);
    }
  })();
  return () => {
    cancelled = true;
  };
}, [category, search, page, setParams]);

const setCategory = useCallback(
  (slug) => {
    const next = new URLSearchParams(params);
    if (slug) next.set("category", slug);
    else next.delete("category");
    next.set("page", "1");
    setParams(next);
  },
  [params, setParams]
);

const applySearch = useCallback(
  (e) => {
    e?.preventDefault?.();
    const next = new URLSearchParams(params);
    const q = searchInput.trim();
    if (q) next.set("search", q);
    else next.delete("search");
    next.set("page", "1");
    setParams(next);
  },
  [params, searchInput, setParams]
);

const goPage = useCallback(
  (p) => {
    const next = new URLSearchParams(params);
    next.set("page", String(p));
    setParams(next);
  },
  [params, setParams]
);

const addToCfg = useCallback(
  (item) => {
    addToConfiguratorDraft(item, user);
    setCfgBump((x) => x + 1);
    toast.success("Додано в конфігуратор");
  },
  [user]
);

return (
  <>
    <div className="flex flex-col sm:flex-row sm:items-start sm:justify-between gap-4 mb-6">
      <div>
        <h1 className="text-2xl font-semibold text-brand-ink mb-1">Каталог</h1>
      </div>
      {configCount > 0 && (
        <Link
          to="/configurator"
```

```
            className="inline-flex items-center justify-center rounded bg-brand-green px-4 py-2 text-sm font-
medium text-white hover:bg-brand-green-hover shrink-0"
          >
            Конфігуратор ({configCount})
          </Link>
        )}
      </div>

      <div className="rounded-lg border border-gray-200 bg-white p-4 shadow-sm mb-6">
        <p className="text-xs font-medium text-brand-muted mb-2">Категорія</p>
        <div className="flex flex-wrap gap-2">
          <button
            type="button"
            onClick={() => setCategory("")}
            className={`rounded-full px-3 py-1 text-sm border ${
              !category ? "border-brand-orange bg-orange-50 text-brand-orange font-medium" : "border-gray-200
hover:bg-gray-50"
            }`}
          >
            Усі
          </button>
          {categories.map((c) => (
            <button
              key={c.id}
              type="button"
              onClick={() => setCategory(c.slug)}
              className={`rounded-full px-3 py-1 text-sm border ${
                category === c.slug
                  ? "border-brand-orange bg-orange-50 text-brand-orange font-medium"
                  : "border-gray-200 hover:bg-gray-50"
              }`}
            >
              {c.name}
            </button>
          ))}
        </div>

        <form onSubmit={applySearch} className="mt-4 flex flex-col sm:flex-row gap-2">
          <input
            value={searchInput}
            onChange={(e) => setSearchInput(e.target.value)}
            placeholder="Пошук за назвою…"
            className="flex-1 rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
          />
          <button
            type="submit"
            className="rounded bg-brand-orange px-4 py-2 text-sm font-medium text-white hover:bg-brand-
orange-hover"
          >
            Шукати
          </button>
        </form>
      </div>

      {error && <p className="text-sm text-red-600 mb-4">{error}</p>}
      {loading && <p className="text-sm text-brand-muted">Завантаження…</p>}

      {!loading && !error && (
        <>
          <p className="text-sm text-brand-muted mb-3">
            Знайдено: {catalog.total} {catalog.total === 1 ? "товар" : "товарів"}
          </p>
          <ul className="grid gap-4 sm:grid-cols-2 lg:grid-cols-3">
            {catalog.items.map((p) => (
              <li
                key={p.id}
                className="flex min-h-0 min-w-0 flex-col overflow-hidden rounded-lg border border-gray-200
bg-white shadow-sm transition-shadow hover:shadow"
              >
                <div className="relative aspect-[4/3] w-full shrink-0 overflow-hidden bg-gray-100">
                  <img
                    src={p.image_url}
                    alt=""
                    className="absolute inset-0 h-full w-full object-cover"
                    loading="lazy"
                    decoding="async"
                  />
                </div>
```

```
              <div className="p-3 flex flex-col flex-1">
                <p className="text-xs text-brand-orange font-medium mb-1">{p.category_name}</p>
                <Link
                  to={`/product/${p.slug}`}
                  className="font-medium text-brand-ink text-sm leading-snug line-clamp-2 mb-2 hover:text-
brand-orange"
                >
                  {p.name}
                </Link>
                <p className="text-lg font-semibold text-brand-ink mt-auto">{money(p.price)}</p>
                <p className={`text-xs mt-1 ${p.stock > 0 ? "text-brand-green" : "text-red-600"}`}>
                  {p.stock > 0 ? `На складі: ${p.stock} шт.` : "Немає в наявності"}
                </p>
                <div className="mt-3 flex flex-wrap gap-2">
                  {user ? (
                    inCartIds.has(p.id) ? (
                      <Link
                        to="/cart"
                        className="flex-1 min-w-[120px] text-center rounded border-2 border-brand-orange
bg-orange-50 py-2 text-xs font-medium text-brand-orange hover:bg-orange-100"
                      >
                        У кошику
                      </Link>
                    ) : (
                      <button
                        type="button"
                        disabled={p.stock < 1 || addingToCartId === p.id}
                        onClick={() => addToCart(p.id)}
                        className="flex-1 min-w-[120px] rounded bg-brand-orange py-2 text-xs font-medium
text-white hover:bg-brand-orange-hover disabled:opacity-40 disabled:cursor-not-allowed"
                      >
                        {addingToCartId === p.id ? "Додаємо…" : "У кошик"}
                      </button>
                    )
                  ) : (
                    <Link
                      to="/login"
                      className="flex-1 min-w-[120px] text-center rounded bg-gray-200 py-2 text-xs font-
medium text-brand-ink hover:bg-gray-300"
                    >
                      У кошик (вхід)
                    </Link>
                  )}
                  <button
                    type="button"
                    onClick={() =>
                      addToCfg({
                        id: p.id,
                        name: p.name,
                        price: p.price,
                        category_slug: p.category_slug,
                        specs: p.specs,
                      })
                    }
                    className="flex-1 min-w-[120px] rounded border border-brand-green py-2 text-xs font-
medium text-brand-green hover:bg-green-50"
                  >
                    У конфігуратор
                  </button>
                </div>
              </div>
            </li>
          ))}
        </ul>

        {catalog.pages > 1 && (
          <nav className="mt-8 flex flex-wrap items-center justify-center gap-2">
            <button
              type="button"
              disabled={page <= 1}
              onClick={() => goPage(page - 1)}
              className="rounded border border-gray-300 px-3 py-1 text-sm disabled:opacity-40"
            >
              Назад
            </button>
            <span className="text-sm text-brand-muted px-2">
              {page} / {catalog.pages}
            </span>
            <button
```

```
                type="button"
                disabled={page >= catalog.pages}
                onClick={() => goPage(page + 1)}
                className="rounded border border-gray-300 px-3 py-1 text-sm disabled:opacity-40"
              >
                Далі
              </button>
            </nav>
          )}
        </>
      )}
    </>
  );
}

import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";

export default function Login() {
  const { login } = useAuth();
  const navigate = useNavigate();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [busy, setBusy] = useState(false);

  async function onSubmit(e) {
    e.preventDefault();
    setError("");
    setBusy(true);
    try {
      await login({ email, password });
      navigate("/");
    } catch (err) {
      setError(err.message || "Помилка входу");
    } finally {
      setBusy(false);
    }
  }

  return (
    <div className="flex w-full flex-1 flex-col items-center justify-center">
      <div className="w-full max-w-md">
      <h1 className="text-2xl font-semibold text-brand-ink mb-1 text-center sm:text-left">Вхід</h1>
      <p className="text-sm text-brand-muted mb-6 text-center sm:text-left">
        Немає акаунта?{" "}
        <Link to="/register" className="text-brand-orange font-medium hover:underline">
          Реєстрація
        </Link>
      </p>

      <form onSubmit={onSubmit} className="rounded-lg border border-gray-200 bg-white p-5 shadow-sm space-y-
4">
        {error && <p className="text-sm text-red-600">{error}</p>}
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Email</label>
          <input
            type="email"
            autoComplete="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            required
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Пароль</label>
          <input
            type="password"
            autoComplete="current-password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            required
          />
        </div>
```

```
        <button
          type="submit"
          disabled={busy}
          className="w-full rounded bg-brand-orange py-2.5 text-sm font-medium text-white hover:bg-brand-
orange-hover disabled:opacity-60"
        >
          {busy ? "Вхід…" : "Увійти"}
        </button>
      </form>
      </div>
    </div>
  );
}

import { useEffect, useState } from "react";
import { Link, Navigate, useParams } from "react-router-dom";
import toast from "react-hot-toast";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";
import { orderStatusLabel } from "../lib/orderStatus.js";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

function formatDate(iso) {
  if (!iso) return "";
  try {
    return new Intl.DateTimeFormat("uk-UA", {
      dateStyle: "medium",
      timeStyle: "short",
    }).format(new Date(iso));
  } catch {
    return String(iso);
  }
}

export default function OrderDetail() {
  const { id } = useParams();
  const { user, loading } = useAuth();
  const [order, setOrder] = useState(null);
  const [fetching, setFetching] = useState(true);
  const [notFound, setNotFound] = useState(false);

  useEffect(() => {
    if (!user || !id) return;
    let cancelled = false;
    (async () => {
      setFetching(true);
      setNotFound(false);
      try {
        const d = await api(`/api/orders/${id}`);
        if (!cancelled) setOrder(d.order);
      } catch (e) {
        if (e.status === 404) {
          if (!cancelled) setNotFound(true);
        } else if (!cancelled) {
          toast.error(e.message || "Помилка");
        }
      } finally {
        if (!cancelled) setFetching(false);
      }
    })();
    return () => {
      cancelled = true;
    };
  }, [user, id]);

  if (!loading && !user) {
    return <Navigate to="/login" replace />;
  }

  if (loading || fetching) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  if (notFound || !order) {
    return (
```

```
      <div>
        <p className="text-brand-muted mb-4">Замовлення не знайдено.</p>
        <Link to="/orders" className="text-brand-orange font-medium hover:underline">
          До списку замовлень
        </Link>
      </div>
    );
  }

  return (
    <div>
      <nav className="text-sm text-brand-muted mb-4">
        <Link to="/orders" className="hover:text-brand-orange">
          Мої замовлення
        </Link>
        <span className="mx-2">/</span>
        <span className="text-brand-ink">#{order.id}</span>
      </nav>

      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Замовлення #{order.id}</h1>
      <p className="text-sm text-brand-orange font-medium mb-1">{orderStatusLabel(order.status)}</p>
      <p className="text-xs text-brand-muted mb-6">{formatDate(order.created_at)}</p>

      <div className="rounded-lg border border-gray-200 bg-white p-4 shadow-sm mb-6 text-sm space-y-2">
        <p>
          <span className="text-brand-muted">Адреса:</span> {order.delivery_address}
        </p>
        <p>
          <span className="text-brand-muted">Оплата:</span> {order.payment_method}
        </p>
        <p>
          <span className="text-brand-muted">Доставка:</span> {order.shipping_method}
        </p>
      </div>

      <h2 className="text-lg font-semibold text-brand-ink mb-3">Склад замовлення</h2>
      <ul className="rounded-lg border border-gray-200 divide-y divide-gray-100 bg-white text-sm shadow-sm">
        {order.items.map((row) => (
          <li key={row.product_id} className="flex flex-wrap items-center justify-between gap-2 px-4 py-3">
            <div>
              <Link to={`/product/${row.slug}`} className="font-medium text-brand-ink hover:text-brand-
orange">
                {row.name}
              </Link>
              <p className="text-xs text-brand-muted">{row.quantity} шт. × {money(row.price_at_purchase)}</p>
            </div>
            <span className="font-medium">{money(row.line_total)}</span>
          </li>
        ))}
      </ul>
      <p className="mt-4 text-base">
        Разом: <span className="font-semibold">{money(order.total)}</span>
      </p>
    </div>
  );
}

import { useEffect, useState } from "react";
import { Link, Navigate } from "react-router-dom";
import toast from "react-hot-toast";
import { api } from "../api";
import { useAuth } from "../context/AuthContext";
import { orderStatusLabel } from "../lib/orderStatus.js";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

function formatDate(iso) {
  if (!iso) return "";
  try {
    return new Intl.DateTimeFormat("uk-UA", {
      dateStyle: "medium",
      timeStyle: "short",
    }).format(new Date(iso));
  } catch {
    return String(iso);
  }
```

```
}

export default function Orders() {
  const { user, loading } = useAuth();
  const [orders, setOrders] = useState([]);
  const [fetching, setFetching] = useState(true);

  useEffect(() => {
    if (!user) return;
    let cancelled = false;
    (async () => {
      try {
        const d = await api("/api/orders");
        if (!cancelled) setOrders(d.orders || []);
      } catch (e) {
        if (!cancelled) toast.error(e.message || "Помилка");
      } finally {
        if (!cancelled) setFetching(false);
      }
    })();
    return () => {
      cancelled = true;
    };
  }, [user]);

  if (!loading && !user) {
    return <Navigate to="/login" replace />;
  }

  if (loading || fetching) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  return (
    <div>
      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Мої замовлення</h1>
      <p className="text-sm text-brand-muted mb-6">Статус замовлення можна відстежувати тут (оновлення – з
боку магазину).</p>

      {orders.length === 0 ? (
        <p className="text-brand-muted">
          Поки немає замовлень.{" "}
          <Link to="/" className="text-brand-orange font-medium hover:underline">
            Перейти в каталог
          </Link>
        </p>
      ) : (
        <ul className="space-y-3">
          {orders.map((o) => (
            <li key={o.id} className="rounded-lg border border-gray-200 bg-white p-4 shadow-sm">
              <div className="flex flex-wrap items-start justify-between gap-2">
                <div>
                  <p className="font-medium text-brand-ink">Замовлення #{o.id}</p>
                  <p className="text-xs text-brand-muted mt-1">{formatDate(o.created_at)}</p>
                  <p className="text-sm text-brand-muted mt-1">
                    {o.item_count} {o.item_count === 1 ? "позиція" : "позицій"}
                  </p>
                </div>
                <div className="text-right">
                  <p className="font-semibold text-brand-ink">{money(o.total)}</p>
                  <p className="text-sm text-brand-orange font-medium mt-1">{orderStatusLabel(o.status)}</p>
                  <Link to={`/orders/${o.id}`} className="text-xs text-brand-orange hover:underline mt-2
inline-block">
                    Деталі
                  </Link>
                </div>
              </div>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
}

import { useCallback, useEffect, useState } from "react";
import toast from "react-hot-toast";
import { Link, useParams } from "react-router-dom";
import { api } from "../api";
```

```
import { useAuth } from "../context/AuthContext";
import { useInCart } from "../hooks/useInCart.js";
import { addToConfiguratorDraft } from "../lib/configurator";

function money(n) {
  return new Intl.NumberFormat("uk-UA", { style: "currency", currency: "UAH", maximumFractionDigits:
0 }).format(n);
}

const SPEC_LABELS = {
  socket: "Сокет",
  ramType: "Тип ОЗП",
  sizeGb: "Об'єм (ГБ)",
  speedMhz: "Частота (МГц)",
  tdpW: "TDP (Вт)",
  vramGb: "VRAM (ГБ)",
  formFactor: "Форм-фактор",
  wattage: "Потужність (Вт)",
  cert: "Сертифікація",
  interface: "Інтерфейс",
  capacityGb: "Ємність (ГБ)",
};

function specLabel(key) {
  return SPEC_LABELS[key] || key;
}

function formatSpecValue(v) {
  if (v === null || v === undefined) return "–";
  if (typeof v === "object") return JSON.stringify(v);
  return String(v);
}

export default function Product() {
  const { slug } = useParams();
  const { user } = useAuth();
  const { inCartIds, addingToCartId, addToCart } = useInCart(user);
  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    let cancelled = false;
    setLoading(true);
    setError("");
    setProduct(null);
    (async () => {
      try {
        const data = await api(`/api/catalog/products/${encodeURIComponent(slug)}`);
        if (!cancelled) setProduct(data.product);
      } catch (e) {
        if (!cancelled) {
          setError(e.status === 404 ? "Товар не знайдено" : e.message || "Помилка");
        }
      } finally {
        if (!cancelled) setLoading(false);
      }
    })();
    return () => {
      cancelled = true;
    };
  }, [slug]);

  const addToCfg = useCallback(() => {
    if (!product) return;
    addToConfiguratorDraft(
      {
        id: product.id,
        name: product.name,
        price: product.price,
        category_slug: product.category_slug,
        specs: product.specs,
      },
      user
    );
    toast.success("Додано в конфігуратор");
  }, [product, user]);

  if (loading) {
```

```
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  if (error || !product) {
    return (
      <div>
        <p className="text-red-600 mb-4">{error || "Товар не знайдено"}</p>
        <Link to="/" className="text-brand-orange font-medium hover:underline">
          До каталогу
        </Link>
      </div>
    );
  }

  const specEntries =
    product.specs && typeof product.specs === "object" && !Array.isArray(product.specs)
      ? Object.entries(product.specs)
      : [];

  const catalogLink = product.category_slug ? `/?category=${encodeURIComponent(product.category_slug)}` :
"/";

  return (
    <div>
      <nav className="text-sm text-brand-muted mb-4">
        <Link to="/" className="hover:text-brand-orange">
          Каталог
        </Link>
        <span className="mx-2">/</span>
        <Link to={catalogLink} className="hover:text-brand-orange">
          {product.category_name}
        </Link>
        <span className="mx-2">/</span>
        <span className="text-brand-ink">{product.name}</span>
      </nav>

      <div className="grid gap-8 lg:grid-cols-2">
        <div className="rounded-lg border border-gray-200 bg-gray-100 overflow-hidden aspect-square max-h-
[420px]">
          <img src={product.image_url} alt="" className="h-full w-full object-contain bg-white" />
        </div>

        <div>
          <p className="text-sm text-brand-orange font-medium mb-1">{product.category_name}</p>
          <h1 className="text-2xl font-semibold text-brand-ink mb-3">{product.name}</h1>
          <p className="text-3xl font-bold text-brand-ink mb-2">{money(product.price)}</p>
          <p className={`text-sm mb-6 ${product.stock > 0 ? "text-brand-green" : "text-red-600"}`}>
            {product.stock > 0 ? `На складі: ${product.stock} шт.` : "Немає в наявності"}
          </p>

          <div className="flex flex-wrap gap-3 mb-8">
            {user ? (
              inCartIds.has(product.id) ? (
                <Link
                  to="/cart"
                  className="inline-flex items-center justify-center rounded border-2 border-brand-orange bg-
orange-50 px-5 py-2.5 text-sm font-medium text-brand-orange hover:bg-orange-100"
                >
                  У кошику
                </Link>
              ) : (
                <button
                  type="button"
                  disabled={product.stock < 1 || addingToCartId === product.id}
                  onClick={() => addToCart(product.id)}
                  className="rounded bg-brand-orange px-5 py-2.5 text-sm font-medium text-white hover:bg-
brand-orange-hover disabled:opacity-40 disabled:cursor-not-allowed"
                >
                  {addingToCartId === product.id ? "Додаємо…" : "Додати в кошик"}
                </button>
              )
            ) : (
              <Link
                to="/login"
                className="inline-flex items-center justify-center rounded bg-gray-200 px-5 py-2.5 text-sm
font-medium text-brand-ink hover:bg-gray-300"
              >
                У кошик (потрібен вхід)
              </Link>
```

```
            )}
            <button
              type="button"
              onClick={addToCfg}
              className="rounded border-2 border-brand-green px-5 py-2.5 text-sm font-medium text-brand-green
hover:bg-green-50"
            >
              У конфігуратор
            </button>
          </div>

          <section className="mb-8">
            <h2 className="text-lg font-semibold text-brand-ink mb-2">Опис</h2>
            <p className="text-sm text-brand-ink whitespace-pre-wrap leading-
relaxed">{product.description}</p>
          </section>

          {specEntries.length > 0 && (
            <section>
              <h2 className="text-lg font-semibold text-brand-ink mb-3">Характеристики</h2>
              <dl className="rounded-lg border border-gray-200 divide-y divide-gray-100 bg-white text-sm">
                {specEntries.map(([key, val]) => (
                  <div key={key} className="grid grid-cols-1 sm:grid-cols-2 gap-1 px-3 py-2 sm:gap-4">
                    <dt className="text-brand-muted">{specLabel(key)}</dt>
                    <dd className="font-medium text-brand-ink">{formatSpecValue(val)}</dd>
                  </div>
                ))}
              </dl>
            </section>
          )}
        </div>
      </div>
    </div>
  );
}

import { useEffect, useState } from "react";
import { Navigate } from "react-router-dom";
import toast from "react-hot-toast";
import { useAuth } from "../context/AuthContext";

export default function Profile() {
  const { user, loading, updateProfile } = useAuth();
  const [full_name, setFullName] = useState("");
  const [phone, setPhone] = useState("");
  const [busy, setBusy] = useState(false);

  useEffect(() => {
    if (!user) return;
    setFullName(user.full_name || "");
    setPhone(user.phone || "");
  }, [user]);

  async function onSubmit(e) {
    e.preventDefault();
    setBusy(true);
    try {
      await updateProfile({ full_name: full_name.trim(), phone: phone.trim() });
      toast.success("Збережено");
    } catch (err) {
      toast.error(err.message || "Помилка збереження");
    } finally {
      setBusy(false);
    }
  }

  if (!loading && !user) {
    return <Navigate to="/login" replace state={{ from: "/profile" }} />;
  }

  if (loading) {
    return <p className="text-sm text-brand-muted">Завантаження…</p>;
  }

  return (
    <div>
      <h1 className="text-2xl font-semibold text-brand-ink mb-2">Профіль</h1>
      <p className="text-sm text-brand-muted mb-6">Редагування імені та телефону. Зміна email – за запитом до
адміністрації.</p>
```

```
      <form onSubmit={onSubmit} className="max-w-md space-y-4 rounded-lg border border-gray-200 bg-white p-5
shadow-sm">
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Email</label>
          <input
            type="email"
            value={user.email}
            disabled
            className="w-full rounded border border-gray-200 bg-gray-50 px-3 py-2 text-sm text-brand-muted"
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Ім’я *</label>
          <input
            value={full_name}
            onChange={(e) => setFullName(e.target.value)}
            minLength={2}
            maxLength={120}
            required
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Телефон</label>
          <input
            value={phone}
            onChange={(e) => setPhone(e.target.value)}
            maxLength={40}
            placeholder="+380…"
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
          />
        </div>
        {user.role === "admin" && (
          <p className="text-xs text-brand-orange">Роль: адміністратор</p>
        )}
        <button
          type="submit"
          disabled={busy || full_name.trim().length < 2}
          className="rounded bg-brand-orange px-5 py-2.5 text-sm font-medium text-white hover:bg-brand-
orange-hover disabled:opacity-50"
        >
          {busy ? "Збереження…" : "Зберегти"}
        </button>
      </form>
    </div>
  );
}

import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";

export default function Register() {
  const { register } = useAuth();
  const navigate = useNavigate();
  const [full_name, setFullName] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [busy, setBusy] = useState(false);

  async function onSubmit(e) {
    e.preventDefault();
    setError("");
    setBusy(true);
    try {
      await register({ full_name, email, phone, password });
      navigate("/");
    } catch (err) {
      setError(err.message || "Помилка реєстрації");
    } finally {
      setBusy(false);
    }
  }
```

```
  return (
    <div className="flex w-full flex-1 flex-col items-center justify-center">
      <div className="w-full max-w-md">
      <h1 className="text-2xl font-semibold text-brand-ink mb-1 text-center sm:text-left">Реєстрація</h1>
      <p className="text-sm text-brand-muted mb-6 text-center sm:text-left">
        Вже є акаунт?{" "}
        <Link to="/login" className="text-brand-orange font-medium hover:underline">
          Вхід
        </Link>
      </p>

      <form onSubmit={onSubmit} className="rounded-lg border border-gray-200 bg-white p-5 shadow-sm space-y-
4">
        {error && <p className="text-sm text-red-600">{error}</p>}
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Ім’я</label>
          <input
            value={full_name}
            onChange={(e) => setFullName(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            minLength={2}
            required
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Email</label>
          <input
            type="email"
            autoComplete="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            required
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Телефон (необов’язково)</label>
          <input
            value={phone}
            onChange={(e) => setPhone(e.target.value)}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
          />
        </div>
        <div>
          <label className="block text-xs font-medium text-brand-muted mb-1">Пароль (мін. 6 символів)</label>
          <input
            type="password"
            autoComplete="new-password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            minLength={6}
            className="w-full rounded border border-gray-300 px-3 py-2 text-sm focus:border-brand-orange
focus:outline-none focus:ring-1 focus:ring-brand-orange"
            required
          />
        </div>
        <button
          type="submit"
          disabled={busy}
          className="w-full rounded bg-brand-orange py-2.5 text-sm font-medium text-white hover:bg-brand-
orange-hover disabled:opacity-60"
        >
          {busy ? "Створення…" : "Зареєструватися"}
        </button>
      </form>
      </div>
    </div>
  );
}
```