НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Клієнт-серверна система управління товарами з рольовою моделлю доступу

Виконав : студент __4__ курсу, групи ІО-25
(шифр групи)

_____Борисенко Таміла Олексіївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ст.викл. Каплунов А.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Нікольський С.С._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ст.викл. Тимофєєва Ю. С._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Борисенко Таміли Олексіївни

1. Тема проєкту *Клієнт-серверна система управління товарами з рольовою моделлю доступу*
керівник проєкту Каплунов Артем Володимирович, ст.викл.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 03 червня 2026 року №2055-с

2. Термін здачі студентом закінченого проєкту *10 червня 2026 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

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

Розділ 1. Аналіз предметної області.

Розділ 2. Огляд технологій для розробки системи.

Розділ 3. Деталі розробки системи.

Розділ 4. Огляд розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень): принципова схема (алгоритм функціонування системи), функціональна схема (діаграма бази даних), структурна схема (діаграма варіантів використання системи).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Нікольський С.С.		

7. Дата видачі завдання «6» січня 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*18.11.2025-16.12.2025*	
2.	*Вивчення та аналіз завдання*	*17.12.2025-21.01.2026*	
3.	*Розробка архітектури та загальної структури системи*	*22.01.2026-18.02.2026*	
4.	*Розробка структур окремих підсистем*	*19.02.2026-20.03.2026*	
5.	*Програмна реалізація системи*	*21.03.2026-04.05.2026*	
6.	*Оформлення пояснювальної записки*	*05.05.2026-20.05.2026*	
7.	*Захист програмного продукту*	*26.05.2026*	
8.	*Передзахист*	*03.06.2026*	
9.	*Захист*	*19.06.2026*	

Студент-дипломник _____ Таміла БОРИСЕНКО
(підпис)

Керівник проєкту _____ Артем КАПЛУНОВ
(підпис)

АНОТАЦІЯ

У роботі розглянуто особливості розробки веборієнтованої системи управління товарами для онлайн-книгарні з використанням клієнт-серверної архітектури. Проведено аналіз предметної області, функціональних і нефункціональних вимог, а також існуючих рішень у сфері електронної комерції. Описано архітектурні підходи до побудови системи, моделі управління доступом користувачів та принципи організації взаємодії між клієнтською і серверною частинами. У роботі обґрунтовано використання Python, Django та PostgreSQL для реалізації серверної логіки і збереження даних. Також розглянуто технології створення клієнтського інтерфейсу за допомогою HTML, CSS і JavaScript та механізми забезпечення інтерактивності вебсистеми.

Ключові слова: інтернет-магазин, вебсистема, Django, Python, PostgreSQL, клієнт-серверна архітектура, ORM, JavaScript, база даних, управління товарами.

ANNOTATION

The thesis considers the features of developing a web-based product management system for an online bookstore using client-server architecture. The subject area, functional and non-functional requirements, and existing solutions in the field of e-commerce are analyzed. Architectural approaches to building a system, user access management models, and principles for organizing interaction between the client and server parts are described. The paper justifies the use of Python, Django, and PostgreSQL to implement server logic and data storage. The technologies for creating a client interface using HTML, CSS, and JavaScript and mechanisms for ensuring web system interactivity are also considered.

Keywords: online store, web system, Django, Python, PostgreSQL, client-server architecture, ORM, JavaScript, database, product management.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Клієнт-серверна система	3		
			управління товарами з			
			рольовою моделлю доступу			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Клієнт-серверна система	68		
			управління товарами з			
			рольовою моделлю доступу			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Клієнт-серверна система	1		
			управління товарами з			
			рольовою моделлю доступу			
			Алгоритм функціонування			
			системи (Принципова схема)			
	А4	***ІАЛЦ.4672008.005 Д2***	Клієнт-серверна система	1		
			управління товарами з			
			рольовою моделлю доступу			
			Діаграма бази даних			
			(Функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Клієнт-серверна система	1		
			управління товарами з			
			рольовою моделлю доступу			
			Діаграма варіантів			
			використання системи			
	А4	***ІАЛЦ.467200.007 Д4***	Клієнт-серверна система	14		
			управління товарами з			
			рольовою моделлю доступу			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Борисенко Т.О.			*Клієнт-серверна система управління товарами з рольовою моделлю доступу* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Каплунов А.В.					
						НТУУ "КПІ" ФІОТ ІО-25	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Клієнт-серверна система управління товарами з рольовою моделлю доступу*»

Київ – 2026

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Борисенко Т.О.			**Клієнт-серверна система управління товарами з рольовою моделлю** Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Каплунов А.В.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-25	
Н. Контр.		Нікольський С.С.					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи управління товарами для онлайн-книгарні, призначеної для автоматизації процесів роботи з каталогом книжкової продукції, користувачами та замовленнями в межах електронної комерції.

Областю застосування цієї системи є онлайн-магазини книжкової продукції та інші інформаційні системи електронної комерції, що забезпечують збереження, обробку та управління товарними даними через вебінтерфейс.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом «Інформатики та обчислювальної техніки», кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веборієнтованої системи управління товарами для онлайн-книгарні, яка забезпечує ефективне ведення каталогу продукції, обробку замовлень, управління користувачами та реалізацію зручної взаємодії між клієнтською і серверною частинами системи.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет, та науково-технічна література на дану тему.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Система повинна забезпечувати управління каталогом товарів (додавання, редагування, видалення та перегляд продукції).
- Має бути реалізовано реєстрацію, авторизацію користувачів і розмежування прав доступу за ролями.
- Система повинна підтримувати роботу з кошиком і оформленням замовлень з подальшою їх обробкою.

5.2. Вимоги до програмного забезпечення

- Операційна система: Windows, Linux або macOS.
- Середовище розробки: Visual Studio Code або інше IDE з підтримкою Python і веброзробки.
- Встановлене середовище виконання Python (версія 3.10 або новіша).
- Веббраузер із підтримкою сучасних вебтехнологій (HTML5, CSS3, JavaScript).

5.3. Вимоги до апаратної частини

Будь-який сучасний персональний комп’ютер або ноутбук із доступом до мережі Інтернет та можливістю встановлення середовища розробки.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	18.11.2025-16.12.2025
Вивчення та аналіз завдання	17.12.2025-21.01.2026
Розробка архітектури та загальної структури системи	22.01.2026-18.02.2026
Розробка структур окремих частин системи	19.02.2026-20.03.2026
Програмна реалізація системи	21.03.2026-20.04.2026
Виправлення помилок	21.04.2026-04.05.2026
Оформлення пояснювальної записки	05.05.2026-20.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Клієнт-серверна система управління товарами з рольовою моделлю доступу*»

Київ – 2026

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

HTTP	HyperText Transfer Protocol (Протокол передачі даних)
HTTPS	HyperText Transfer Protocol Secure (Захищена версія HTTP)
URL	Uniform Resource Locator (Уніфікований локатор ресурсів)
SQL	Structured Query Language (Мова запитів до бази даних)
JSON	JavaScript Object Notation (Формат обміну структурованими даними)
API	Application Programming Interface (Інтерфейс програмування застосунків)
ORM	Object-Relational Mapping (Об’єктно-реляційне відображення даних)
MVC	Model–View–Controller (Архітектурний шаблон проєктування програмного забезпечення)
MTV	Model–Template–View (Архітектурна модель фреймворку Django)
CRUD	Create, Read, Update, Delete (Базові операції над даними)
RBAC	Role-Based Access Control (Рольова модель керування доступом)
CSRF	Cross-Site Request Forgery (Тип веб-атаки, пов’язаний з підробкою запитів користувача)
DOM	Document Object Model (Об'єктна модель документа)
HTML	HyperText Markup Language (Мова розмітки)
CSS	Cascading Style Sheets (Мова стилю сторінок)
JS	JavaScript (Мова програмування)
AJAX	Asynchronous JavaScript and XML (Технологія асинхронного обміну даними з сервером)
UX/UI	User Experience / User Interface (Досвід користувача / інтерфейс користувача)

ВСТУП

Сьогодні інформаційні технології поступово стають невід'ємною частиною більшості процесів, пов'язаних із повсякденним життям людини. Те, що раніше вимагало значних витрат часу та зусиль, зараз часто вирішується за допомогою кількох дій у вебсередовищі. Особливо це помітно у сфері електронної комерції, де взаємодія між користувачем і товарами відбувається через цифрові платформи, що працюють у режимі реального часу.

У таких умовах зростає роль систем, які дозволяють ефективно організовувати роботу з товарними даними, забезпечуючи їх зберігання, оновлення та зручне представлення для користувача. Подібні рішення лежать в основі сучасних інтернет-магазинів, де важливо не лише показати перелік продукції, а й забезпечити логічну структуру, швидкий доступ до інформації та стабільну взаємодію між усіма компонентами системи.

Разом із цим, розвиток вебтехнологій і збільшення кількості користувачів онлайн-сервісів висувають нові вимоги до таких систем. Вони мають бути не тільки функціональними, але й гнучкими, безпечними та здатними коректно працювати під навантаженням. Окремої уваги потребує також питання розмежування доступу до функціоналу системи, оскільки різні категорії користувачів виконують різні ролі в межах однієї платформи.

Саме тому актуальним є дослідження та розробка системи, яка поєднує в собі зручність взаємодії, чітку організацію даних та сучасні підходи до побудови архітектури вебзастосунків. У межах даної роботи розглядаються основні принципи побудови подібних систем та підходи, що дозволяють реалізувати їх у вигляді цілісного програмного рішення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області систем управління товарами

Сучасні веборієнтовані системи управління товарами використовуються для організації, зберігання та обробки інформації про продукцію в межах електронних торговельних платформ. Такі системи забезпечують взаємодію між користувацьким інтерфейсом, серверною частиною застосунку та базою даних, що дозволяє автоматизувати процеси роботи з товарами, обробки запитів і оформлення замовлень. Одним із найбільш поширених прикладів реалізації таких систем є інтернет-магазини. Основою їх функціонування є клієнт-серверна архітектура, де клієнтська частина відповідає за взаємодію з користувачем, а серверна – за обробку даних, реалізацію бізнес-логіки та взаємодію з базою даних [1].

Ключовим компонентом системи управління товарами є каталог продукції, який забезпечує структуроване відображення доступних товарів. Зазвичай він включає категорії, фільтри, механізми сортування та систему пошуку, що дозволяє користувачу швидко знаходити необхідні позиції. У процесі роботи система отримує користувацькі запити через вебінтерфейс, обробляє їх на сервері та повертає результат у вигляді вебсторінок або динамічних даних.

Окрему роль у роботі системи відіграє механізм кошика, який дозволяє користувачу тимчасово зберігати обрані товари до моменту оформлення замовлення. Дані про кошик можуть зберігатися як у межах користувацької сесії, так і централізовано в базі даних залежно від реалізації системи. Після підтвердження замовлення інформація передається на сервер, де формується запис про покупку та здійснюється подальша обробка даних.

Важливою складовою веборієнтованої системи є підсистема управління користувачами, яка забезпечує реєстрацію, автентифікацію, авторизацію та керування персональними даними [2]. Вона дозволяє зберігати історію

замовлень, адреси доставки та інші персоналізовані налаштування користувача. Реалізація таких механізмів забезпечує не лише зручність використання системи, а й контроль доступу до функціональних можливостей відповідно до ролей користувачів.

1.2 Вимоги до системи управління товарами та користувачами

1.2.1 Функціональні вимоги

Функціональні вимоги визначають перелік основних можливостей, які повинна забезпечувати система для коректної роботи з товарами та користувачами. Вони описують, які операції доступні користувачам і адміністраторам, а також яким чином система має опрацьовувати їх виконання.

Однією з базових функцій є управління каталогом товарів. До основних функцій належить додавання нових товарів, редагування існуючих позицій, видалення застарілих або недоступних продуктів, а також перегляд повного списку товарів. До кожного товару має зберігатися набір атрибутів, таких як назва, опис, ціна, зображення, категорія та кількість на складі. Це дозволяє забезпечити повноцінне відображення інформації для кінцевого користувача.

Важливою функціональною вимогою є реалізація системи категоризації товарів. Вона повинна дозволяти групувати продукцію за певними ознаками, що значно спрощує навігацію по каталогу. Крім того, система має підтримувати фільтрацію та сортування товарів за різними параметрами, зокрема за ціною, популярністю або датою додавання.

Окрему групу вимог становлять функції, пов’язані з управлінням користувачами. Система повинна забезпечувати реєстрацію нових користувачів, їхню авторизацію, а також можливість редагування особистих даних. Також важливо передбачити різні рівні доступу, наприклад, звичайний користувач та адміністратор, які мають різні права в системі.

Не менш важливою є функціональність роботи з кошиком і замовленнями. Користувач повинен мати можливість додавати товари до кошика, змінювати їх кількість, видаляти позиції та оформлювати замовлення. Після підтвердження замовлення система повинна зберігати інформацію про покупку та змінювати відповідні дані у базі даних.

До функціональних вимог також відноситься система обробки замовлень на стороні адміністратора. Вона повинна дозволяти переглядати список замовлень, змінювати їх статуси (наприклад, “нове”, “в обробці”, “відправлено”, “завершено”) та контролювати виконання замовлень.

1.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики якості системи, які впливають на стабільність її роботи, безпеку, продуктивність та зручність використання. На відміну від функціональних вимог, вони описують не конкретні можливості системи, а умови, за яких ці можливості повинні працювати ефективно та надійно.

Однією з основних нефункціональних вимог є продуктивність інформаційної системи. Система повинна забезпечувати швидке завантаження сторінок каталогу, коректну роботу пошуку товарів та оперативну обробку дій користувача навіть при одночасному виконанні великої кількості запитів. Оптимізація роботи сервера та бази даних дозволяє уникнути затримок під час перегляду товарів або підтвердження покупки.

Не менш важливою вимогою є безпека системи. Оскільки онлайн-магазин працює з персональними даними користувачів та інформацією про замовлення, необхідно забезпечити захист облікових записів і конфіденційної інформації. Для цього використовуються механізми автентифікації користувачів, перевірки прав доступу, захисту форм від CSRF-атак та шифрування критично важливих

даних [3]. Особлива увага приділяється адміністративній частині системи, доступ до якої повинні мати лише авторизовані адміністратори.

Важливою характеристикою програмної системи є також надійність роботи. Система повинна стабільно функціонувати навіть при високому навантаженні або виникненні помилок у процесі обробки даних. Наприклад, при оформленні замовлення не допускається втрата інформації про товари у кошику або некоректне збереження даних користувача. Для цього застосовуються механізми резервного копіювання бази даних, обробки винятків та контролю цілісності інформації.

Окрему роль відіграє масштабованість системи. Інформаційна система повинна мати можливість подальшого розширення функціоналу без суттєвого перепроєктування архітектури. Це стосується як збільшення кількості товарів і користувачів, так і можливого додавання нових модулів, наприклад системи відгуків, онлайн-оплати або рекомендаційних сервісів.

1.3 Архітектурні підходи до побудови клієнт-серверних систем

1.3.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура є базовою моделлю побудови більшості сучасних інформаційних систем. Вона передбачає поділ системи на дві основні частини: клієнтську, яка відповідає за взаємодію з користувачем, та серверну, яка виконує обробку даних і реалізацію бізнес-логіки. Такий підхід дозволяє чітко розмежувати відповідальність між компонентами системи та забезпечити її структурованість.

У межах цієї архітектури клієнт виступає ініціатором запитів. Користувач через веббраузер взаємодіє з інтерфейсом, формує запити до сервера та отримує відповіді у вигляді вебсторінок або структурованих даних. Сервер, у свою

чергу, приймає ці запити, обробляє їх, взаємодіє з базою даних і повертає результат.

Загальну схему взаємодії клієнта, сервера та бази даних наведено на рисунку 1.1.

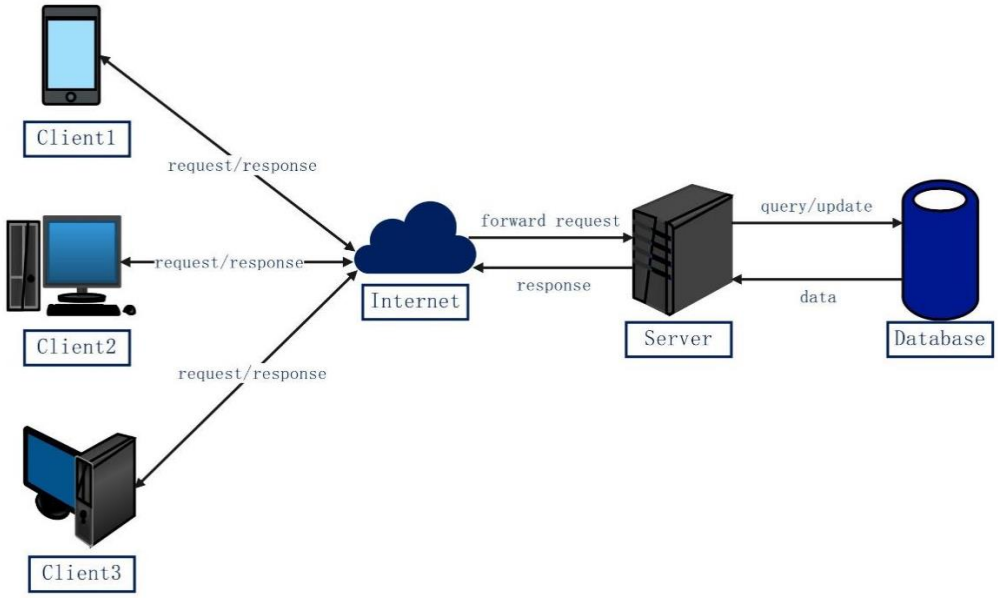


Рисунок 1.1 – Загальна схема клієнт-серверної архітектури вебзастосунку

Важливою перевагою клієнт-серверної моделі є централізоване зберігання даних на стороні сервера. Це означає, що усі відомості містяться в єдиній базі даних, що забезпечує її цілісність, узгодженість та спрощує управління. Клієнт при цьому не має прямого доступу до бази даних, що підвищує рівень безпеки системи.

Ще однією характерною особливістю є можливість масштабування. Серверна частина може бути розширена для обробки більшої кількості запитів шляхом збільшення ресурсів або розподілу навантаження між кількома серверами. Це особливо важливо для інтернет-магазинів, де кількість користувачів може суттєво змінюватися залежно від часу або навантаження.

Клієнт-серверна архітектура дозволяє розділяти розвиток інтерфейсу та серверної логіки. Це означає, що фронтенд і бекенд можуть розроблятися

незалежно один від одного, використовуючи різні технології, але взаємодіючи через стандартизовані протоколи, такі як HTTP або HTTPS.

1.3.2 Моделі управління доступом у клієнт-серверних системах

У клієнт-серверних інформаційних системах питання управління користувачами та контролю доступу є одним із ключових аспектів безпеки та організації роботи системи. Від правильно обраної моделі доступу залежить, які дії можуть виконувати різні категорії користувачів, а також наскільки захищеними будуть дані від несанкціонованого використання.

Моделі управління доступу визначають правила взаємодії користувачів із ресурсами системи. Найбільш поширеною є рольова модель доступу (Role-Based Access Control, RBAC), яка передбачає розподіл користувачів за ролями, кожна з яких має свій набір дозволів. Наприклад, у межах інтернет-магазину можна виділити ролі звичайного користувача, адміністратора та менеджера, і кожна з них матиме різні права доступу до функціоналу системи.

У такій моделі звичайний користувач може переглядати товари, додавати їх до кошика та оформлювати замовлення, тоді як адміністратор має розширені можливості, включаючи управління каталогом товарів, користувачами та замовленнями. Це дозволяє чітко розмежувати функціональні можливості та зменшити ризик випадкового або навмисного втручання в критичні частини системи.

Графічне представлення розподілу прав доступу між різними групами користувачів наведено на рисунку 1.2.

Рисунок 1.2 – Схема рольової моделі доступу користувачів у вебсистемі

Окрім рольової моделі, у інформаційних системах також використовується модель на основі дозволів (Permission-Based Access Control), де доступ до кожної конкретної дії визначається окремими дозволами. Такий підхід є більш гнучким, оскільки дозволяє тонко налаштовувати права доступу для кожного користувача або групи користувачів.

Важливою складовою управління користувачами є процес автентифікації та авторизації. Автентифікація відповідає за перевірку особи користувача, зазвичай через логін і пароль, тоді як авторизація визначає, які дії цей користувач може виконувати після входу в систему. У клієнт-серверній архітектурі ці процеси часто реалізуються через сесії або токени доступу.

Також слід враховувати механізми захисту користувацьких даних. У системі повинні використовуватися методи шифрування паролів, захисту сесій та обмеження спроб входу, щоб запобігти несанкціонованому доступу. Це особливо важливо для інтернет-магазинів, де обробляються персональні дані та інформація про замовлення.

1.4 Аналіз існуючих рішень у сфері онлайн-книгарень

Для практичної реалізації та дослідження принципів побудови розподілених інформаційних систем у межах даної роботи обрано галузь онлайн-торгівлі книжковою продукцією. Книжкові веборієнтовані системи управління товарами є окремим видом інформаційних систем електронної комерції, орієнтованих на продаж та організацію книжкової продукції. Для таких систем характерними є деталізована структура каталогу, розширені механізми пошуку та фільтрації, а також використання додаткових елементів взаємодії з користувачем, зокрема рейтингів, відгуків і рекомендаційних сервісів.

Для визначення особливостей реалізації сучасних книжкових інтернет-магазинів доцільно провести аналіз існуючих платформ, які використовуються в українському сегменті електронної комерції.

1.4.1 Онлайн-книгарня Yakaboo

Одним із найвідоміших прикладів українських онлайн-книгарень є Yakaboo [4], яка представляє собою масштабну платформу для продажу книжкової продукції в цифровому та друкованому форматах. Сервіс орієнтований на широкий спектр користувачів і пропонує значний каталог літератури різних жанрів, включаючи художні твори, навчальні видання та професійну літературу.

Особливістю платформи є добре структурована система навігації, що дозволяє користувачам швидко знаходити необхідні книги за категоріями, авторами або ключовими словами. Окрім базового функціоналу пошуку, система підтримує персоналізовані рекомендації, що формуються на основі історії переглядів та покупок користувача.

Також важливо відзначити наявність продуманої системи картки товару, де кожна книга містить детальний опис, інформацію про автора, видавництво, формат та ціну. Додатково реалізовано функціонал відгуків і рейтингів, що дозволяє користувачам обмінюватися думками щодо прочитаних видань та формувати більш обґрунтоване рішення щодо покупки.

З технічної точки зору платформа демонструє стабільну роботу навіть при високому навантаженні, що досягається завдяки оптимізованій серверній архітектурі та використанню кешування даних. Окрему увагу приділено мобільній адаптації, що дозволяє комфортно користуватися сервісом на різних пристроях без втрати функціональності.

1.4.2 Онлайн-книгарня «Книгарня Є»

Серед українських онлайн-ресурсів, що спеціалізуються на продажу книг, важливе місце займає Книгарня Є [5]. Це платформа, яка поєднує функції інтернет-магазину та культурного простору, орієнтованого на популяризацію читання. Сервіс пропонує широкий вибір української та зарубіжної літератури, а також додаткові товари, пов’язані з книжковою тематикою.

Однією з характерних особливостей даної книгарні є акцент на українському контенті та підтримці національних видавництв. Це проявляється у структурі каталогу, де значна увага приділена українській класичній і сучасній літературі, а також перекладним виданням. Такий підхід формує не лише комерційну, а й культурну цінність платформи.

Інтерфейс системи побудований таким чином, щоб користувач міг легко орієнтуватися в асортименті та швидко знаходити потрібні книги. Реалізовано зручну систему фільтрів за жанрами, авторами, мовою та ціновим діапазоном. Окремо варто відзначити логіку відображення картки товару, де міститься основна інформація про видання, включаючи опис, обкладинку, наявність та ціну. Це дозволяє користувачу швидко приймати рішення щодо покупки.

Окрему увагу платформа приділяє додатковим функціям взаємодії з користувачами, таким як акційні пропозиції, тематичні добірки та рекомендаційні блоки. Це дозволяє підвищити залученість користувачів та стимулювати додаткові покупки. Крім того, сайт підтримує інформаційні розділи, пов’язані з літературними подіями, що підсилює його роль як культурного ресурсу.

1.4.3 Онлайн-книгарня Bookling

Серед сучасних онлайн-платформ для продажу книжкової продукції окрему увагу привертає Bookling [6], яка орієнтується на простий та швидкий процес взаємодії користувача з системою. Платформа поєднує базові

можливості інтернет-магазину з лаконічним інтерфейсом, що дозволяє користувачам швидко знаходити необхідні книги та оформлювати замовлення без складних додаткових дій.

Однією з особливостей сервісу є спрощена структура інтерфейсу та мінімалістичний підхід до організації навігації. Каталог побудований за основними категоріями та жанрами, що дозволяє користувачам легко орієнтуватися у доступному асортименті. На відміну від більш масштабних платформ, система не перевантажена великою кількістю додаткових елементів або складних функцій, завдяки чому сторінки залишаються зрозумілими та зручними у використанні.

Функціонально Bookling забезпечує основний набір можливостей для роботи з книжковою продукцією, а пошуковий механізм реалізований у базовому вигляді та дозволяє знаходити книги за ключовими словами або назвами. Такий підхід забезпечує швидкий доступ до потрібної інформації та спрощує процес взаємодії з платформою.

Важливою особливістю платформи є орієнтація саме на базові сценарії використання. Система не містить складних механізмів аналітики або розширеної персоналізації, однак забезпечує достатній рівень функціональності для стандартного процесу онлайн-покупки книг. Це дозволяє зосередити увагу користувача безпосередньо на пошуку та замовленні товарів.

1.4.4 Порівняльний аналіз існуючих рішень

Аналіз сучасних онлайн-книгарень показує, що більшість з них мають спільні базові функціональні можливості, однак відрізняються підходами до реалізації інтерфейсу, рівнем персоналізації та додатковими сервісами. Для формування вимог до розроблюваної системи важливо порівняти ключові характеристики розглянутих платформ, зокрема Yakaboo, Книгарня Є та Bookling.

Основними критеріями порівняння обрано: зручність користувацького інтерфейсу, наявність системи пошуку та фільтрації, рівень персоналізації, функціонал кошика та оформлення замовлення, а також загальна адаптивність під різні пристрої. Саме ці параметри найбільшою мірою впливають на якість взаємодії користувача з інтернет-магазином та ефективність його використання.

Нижче наведено узагальнену таблицю порівняння основних характеристик розглянутих онлайн-книгарень.

Таблиця 1.1 – Порівняльний аналіз онлайн-книгарень

Критерій	**Yakaboo**	**Книгарня Є**	**Bookling**
Зручність інтерфейсу	Висока	Середня	Висока
Система пошуку	Розширена (фільтри, рекомендації)	Базова + фільтри	Оптимізована + пошук
Персоналізація	Розвинена	Обмежена	Середня
Кошик та оформлення замовлення	Повноцінний функціонал	Стандартний	Спрощений, зручний
Мобільна адаптація	Так	Так	Так

Як видно з порівняння, найбільш функціонально розвиненою є платформа Yakaboo, яка має потужну систему рекомендацій та розширений пошук. Водночас Книгарня Є робить акцент не лише на продажах, а й на культурній складовій, тоді як Bookling вирізняється простотою інтерфейсу та зручністю взаємодії. Проведений аналіз дозволяє зробити висновок, що кожна з розглянутих систем має свої сильні сторони, однак водночас містить і певні обмеження, зокрема у сфері персоналізації або гнучкості функціоналу.

ВИСНОВОК ДО РОЗДІЛУ

Проведений аналіз предметної області засвідчив, що ефективне функціонування веборієнтованої системи управління товарами потребує комплексного поєднання засобів роботи з каталогом продукції, механізмів обробки замовлень та інструментів управління користувачами. Встановлено, що важливими чинниками якості таких систем є зручність пошуку інформації, швидкість обробки запитів, захист даних користувачів і можливість подальшого розширення функціоналу.

Дослідження архітектурних підходів показало доцільність використання клієнт-серверної моделі, оскільки вона забезпечує централізоване зберігання даних, спрощує адміністрування системи та створює умови для її масштабування. Застосування рольового розмежування доступу дозволяє підвищити рівень безпеки та забезпечити контроль над виконанням операцій різними категоріями користувачів.

Порівняння існуючих онлайн-книгарень дало змогу визначити функціональні рішення, які позитивно впливають на зручність використання системи, зокрема структурований каталог, механізми фільтрації та пошуку, адаптивний інтерфейс і підтримку процесу оформлення замовлень. Водночас аналіз показав необхідність поєднання широких можливостей управління товарами із простотою взаємодії для кінцевого користувача.

Отримані результати стали основою для формування вимог до розроблюваної інформаційної системи та обґрунтували вибір архітектурних і функціональних рішень, які будуть використані в подальших етапах проєктування та реалізації вебзастосунку.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Загальна концепція та архітектура системи

Розроблювана система інтернет-магазину книжкової продукції базується на сучасному підході до побудови вебзастосунків, де ключову роль відіграє поділ на логічні рівні: клієнтську частину, серверну логіку та рівень збереження даних. Така структура дозволяє забезпечити зрозумілу організацію коду, спрощує масштабування проєкту та робить систему більш гнучкою до подальших змін.

У якості основи технологічного стеку використано Python як мову серверної розробки, вебфреймворк Django для реалізації бізнес-логіки, PostgreSQL як систему управління базами даних, а також стандартні фронтенд-технології HTML, CSS і JavaScript для побудови інтерфейсу користувача. Кожна з цих технологій виконує свою окрему роль, але разом вони утворюють цілісну систему, орієнтовану на швидку розробку та стабільну роботу вебзастосунку.

Вибір саме такого стеку обумовлений поєднанням кількох факторів. По-перше, Python і Django дозволяють швидко реалізовувати функціональні вебсистеми завдяки великій кількості вбудованих інструментів. По-друге, PostgreSQL забезпечує надійне зберігання структурованих даних і добре масштабується при збільшенні обсягів інформації. По-третє, використання класичних вебтехнологій на клієнтській стороні дозволяє створити зручний та інтуїтивний інтерфейс без надмірної складності.

Архітектурно система побудована за клієнт-серверною моделлю. Користувач взаємодіє із вебінтерфейсом через браузер, надсилаючи запити до сервера. Сервер обробляє ці запити, виконує необхідні операції з даними та

повертає сформовану відповідь у вигляді HTML-сторінки або структурованого JSON-відгуку.

Особливістю даної архітектури є чітке розділення відповідальності між компонентами. Відображення інформації та забезпечення взаємодії з користувачем виконує клієнтська частина, а серверна частина реалізує всю бізнес-логіку, включаючи обробку замовлень, роботу з кошиком та управління каталогом товарів. Рівень бази даних, у свою чергу, відповідає за надійне збереження всієї інформації та її структуровану організацію.

2.2 Технології та засоби реалізації системи

2.2.1 Технічні характеристики та переваги Python

Python є мовою високого рівня, яка відзначається простим і читабельним синтаксисом, що робить її зручною як для розробників-початківців, так і для реалізації масштабних комерційних проєктів [7]. Однією з ключових особливостей є те, що код на Python максимально наближений до природної мови, що значно спрощує його супровід і модифікацію в майбутньому.

З технічної точки зору Python підтримує динамічну типізацію та автоматичне керування пам’яттю, що дозволяє зменшити кількість низькорівневих помилок і зосередитися на логіці програми. Крім того, мова має потужну стандартну бібліотеку, яка покриває широкий спектр задач – від роботи з файлами до мережевих протоколів і обробки даних.

Важливою перевагою Python у веброзробці є велика кількість сторонніх бібліотек і фреймворків. Завдяки цьому розробник може не створювати функціонал з нуля, а використовувати готові рішення для автентифікації користувачів, роботи з базами даних, обробки форм або створення API. Саме це суттєво прискорює процес розробки вебзастосунків і знижує кількість потенційних помилок.

Окремо слід відзначити популярність Python у серверній розробці. Мова активно використовується у великих проєктах завдяки своїй стабільності, підтримці спільноти та можливості масштабування.

У практичному застосуванні для даного проєкту Python виступає основою для реалізації серверної логіки, включаючи обробку запитів користувачів, роботу з моделями даних та взаємодію з ORM-рівнем Django. Це дозволяє створити єдину логічну структуру, де всі операції виконуються послідовно.

2.2.2 Реалізація серверної логіки та обробка даних

Серверна логіка у розроблюваній системі будується навколо обробки HTTP-запитів, які надходять від клієнтської частини, та подальшого формування відповідей на основі бізнес-правил застосунку. У контексті Python і Django кожен запит проходить через маршрутизацію URL, після чого передається у відповідний контролер (view), де відбувається основна обробка даних і прийняття рішень щодо того, яку інформацію необхідно повернути користувачу.

На рівні реалізації це виглядає як послідовність дій: отримання запиту, перевірка параметрів, взаємодія з моделями бази даних, формування результату та повернення відповіді. Наприклад, при перегляді списку товарів система звертається до бази даних через ORM Django , отримує необхідні записи, після чого передає їх у шаблон для відображення [8]. У цьому процесі Python виступає як центральний інструмент, що координує всі етапи обробки.

Важливим аспектом є робота з даними користувача, зокрема з формами, які надходять через POST-запити. Перед збереженням інформації система виконує її валідацію, перевіряючи коректність введених значень, що дозволяє уникнути помилок та неузгодженості даних у базі. Наприклад, при оформленні замовлення перевіряється наявність товарів у кошику, правильність контактних даних та інші обов’язкові поля.

Окрему роль у серверній логіці відіграє механізм взаємодії з базою даних. Замість прямого використання SQL-запитів Python у Django працює через ORM-рівень, який дозволяє оперувати об’єктами замість таблиць. Це спрощує код і робить його більш читабельним. Наприклад, отримання списку товарів може виконуватися одним виразом, який автоматично перетворюється у SQL-запит на стороні фреймворку.

Ще одним важливим елементом є забезпечення безпеки серверної частини. Python у поєднанні з Django надає вбудовані механізми захисту, такі як захист від CSRF-атак, контроль сесій користувачів та перевірка прав доступу. Це особливо важливо для операцій, пов’язаних із замовленнями, кошиком та адміністративною частиною системи, де необхідно обмежувати доступ до певних функцій лише авторизованим користувачам.

2.3 Вебфреймворк Django як основа системи управління продуктами

2.3.1 Архітектурна модель MTV та принципи побудови фреймворку

Архітектура Django базується на моделі MTV (Model–Template–View), яка є адаптованою варіацією класичної MVC-архітектури [9]. У цій концепції кожен компонент виконує чітко визначену роль, що дозволяє розділити відповідальність між різними частинами системи та зробити код більш структурованим і підтримуваним.

Модель (Model) відповідає за роботу з даними та бізнес-логіку, зокрема за опис структури бази даних та операції над нею. У контексті інтернет-магазину саме моделі визначають сутності, такі як товари, користувачі або замовлення, а також правила їх зберігання та взаємодії.

Шаблони (Template) відповідають за рівень представлення даних користувачу. Вони визначають, як саме інформація буде відображатися у браузері, формуючи HTML-сторінки на основі даних, отриманих від серверної

частини. Це дозволяє відокремити логіку обробки даних від їхнього візуального представлення, що позитивно впливає на гнучкість системи.

View виступає проміжним шаром між моделями та шаблонами. Саме у цьому компоненті відбувається обробка HTTP-запитів, виклик необхідних моделей та передача даних у шаблони для подальшого відображення. Таким чином View фактично координує роботу всієї серверної логіки.

Однією з важливих концепцій Django є принцип “batteries included”, який означає наявність великої кількості вбудованих інструментів для вирішення типових задач. Це включає систему автентифікації користувачів, ORM, адмін-панель, систему маршрутизації та багато інших компонентів, які дозволяють розробнику зосередитися на бізнес-логіці, а не на реалізації базової інфраструктури.

Окрему роль у роботі фреймворку відіграє механізм маршрутизації запитів (URL routing), який забезпечує перенаправлення HTTP-запитів до відповідних View-компонентів [10]. Після отримання запиту система визначає потрібний маршрут, виконує обробку даних через View та Model, а потім формує HTML-відповідь за допомогою Template. Загальну схему взаємодії компонентів MTV у Django наведено на рисунку 2.1.

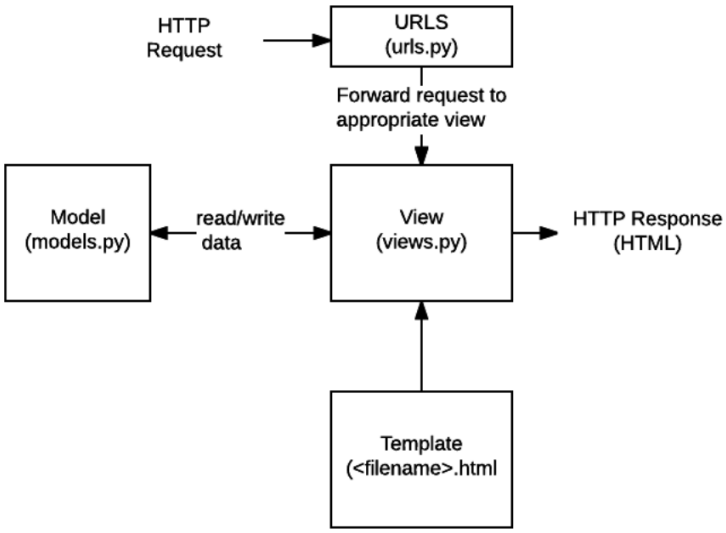


Рисунок 2.1 – Схема взаємодії компонентів архітектури MTV у Django під час обробки HTTP-запиту

2.3.2 Робота з даними через Django ORM та інструменти адміністрування

Після формування архітектурної основи фреймворку важливо розглянути, яким чином у системі реалізується робота з даними. Саме цей рівень відповідає за взаємодію з базою даних, створення бізнес-логіки та забезпечення зручного керування інформацією без необхідності ручного написання SQL-запитів. У межах Django ці задачі вирішуються через ORM та вбудовану адміністративну панель.

Однією з ключових особливостей Django є ORM, який дозволяє працювати з базою даних через об’єкти мови Python, а не через прямі SQL-запити [8]. Це значно спрощує розробку та зменшує ризик помилок при взаємодії з даними.

Моделі в Django описують структуру таблиць у базі даних у вигляді класів. Кожна модель відповідає окремій сутності системи, наприклад, товарам, користувачам або замовленням. Поля моделі визначають типи даних, які будуть зберігатися, а також зв’язки між таблицями, наприклад один-до-одного, один-

до-багатьох або багато-до-багатьох. Саме завдяки цьому підходу структура бази даних стає більш зрозумілою та логічно пов’язаною з кодом застосунку.

Для виконання операцій над даними використовується механізм QuerySet, який дозволяє отримувати, фільтрувати та змінювати записи, використовуючи лише методи Python. Загальну логіку цього процесу, що ілюструє шлях від виклику методів у коді до виконання запитів у базі даних, наведено на рисунку 2.2.

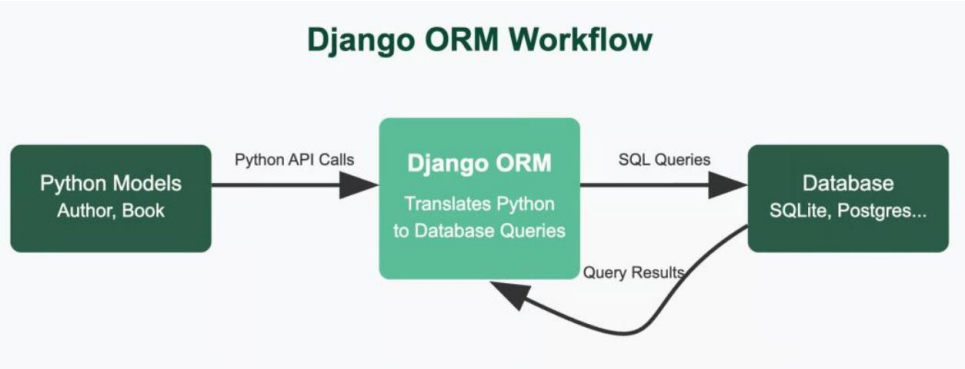


Рисунок 2.2 – Схема взаємодії Django ORM із базою даних

Як видно зі схеми, ORM виступає проміжним шаром, який автоматично транслює Python-код у валідні SQL-запити до PostgreSQL, що дозволяє розробнику зосередитися на бізнес-логіці, а не на синтаксисі бази даних.

Ще одним важливим інструментом є автоматично згенерована адміністративна панель Django, яка надає готовий інтерфейс для керування даними системи [11]. Вона дозволяє без додаткової розробки виконувати CRUD-операції (створення, читання, оновлення та видалення записів), що суттєво прискорює розробку проєкту.

Після реєстрації моделей у адміністративній панелі розробник отримує можливість керувати товарами, користувачами та іншими сутностями через зручний веб-інтерфейс. Це особливо корисно для інтернет-магазинів, де необхідно часто оновлювати інформацію про продукцію, змінювати ціни або додавати нові позиції.

Також адмін-панель підтримує додаткові можливості налаштування, такі як фільтрація, пошук, сортування та групування даних. Завдяки цьому навіть великі обсяги інформації можуть ефективно оброблятися без необхідності створення окремого інтерфейсу для адміністратора.

2.4 Система управління базами даних PostgreSQL

2.4.1 Характеристика об'єктно-реляційної СУБД PostgreSQL

PostgreSQL є однією з найпотужніших об’єктно-реляційних систем управління базами даних, яка поєднує класичний реляційний підхід із розширеними можливостями об’єктної моделі [12]. Це дозволяє не лише зберігати структуровані дані у вигляді таблиць, але й використовувати складні типи даних, функції та механізми розширення.

Однією з ключових переваг PostgreSQL є його надійність та відповідність принципам ACID (атомарність, узгодженість, ізольованість, довговічність), що гарантує коректність транзакцій навіть у випадку збоїв системи. Завдяки цьому система особливо добре підходить для інтернет-магазинів, де важливо уникати втрати або некоректного збереження замовлень та даних користувачів.

База даних підтримує широкий спектр типів даних: числові, текстові, логічні, дати та час, а також більш складні структури, такі як JSON і масиви. Це дозволяє гнучко проєктувати схему даних відповідно до потреб конкретного застосунку. Наприклад, інформація про товари може містити як стандартні поля (назва, ціна), так і додаткові характеристики у форматі структурованих даних.

Ще однією важливою особливістю PostgreSQL є можливість розширення функціоналу через власні типи даних, індекси та користувацькі функції. Крім того, вона ефективно працює з великими обсягами інформації, зберігаючи стабільну швидкість виконання запитів навіть при високому навантаженні.

2.4.2 Проєктування структури бази даних та інтеграція з backend-частиною

Проєктування структури бази даних є одним із ключових етапів розробки вебзастосунку, оскільки саме воно визначає, як буде організовано збереження, обробка та взаємозв’язок інформації. У межах даного проєкту схема даних формується таким чином, щоб ефективно підтримувати роботу інтернет-магазину та забезпечувати узгоджену взаємодію між користувачами, товарами та замовленнями.

Під час проєктування структури бази даних визначаються основні сутності системи, зокрема товари, категорії, користувачі та замовлення, а також їх взаємозв’язки. Кожна сутність реалізується у вигляді окремої таблиці, що дозволяє уникати дублювання даних та забезпечує їх логічну організацію. Важливу роль відіграє нормалізація, яка допомагає зменшити надмірність інформації та підвищити цілісність даних.

Інтеграція PostgreSQL із backend-частиною на базі Django здійснюється через ORM-шар, який виступає проміжною ланкою між кодом застосунку та базою даних. Завдяки цьому розробник працює не з SQL-запитами напряму, а з об’єктами Python, що значно спрощує процес розробки та супроводу системи. Наприклад, при додаванні нового товару створюється відповідний об’єкт моделі, який автоматично синхронізується з базою даних.

Особливістю такої інтеграції є автоматичне відображення змін у структурі моделей на рівні бази даних через механізм міграцій. Це дозволяє поступово розвивати систему, не порушуючи цілісності вже існуючих даних. Крім того, Django забезпечує зручні інструменти для виконання складних запитів, фільтрації та агрегації даних без необхідності написання складного SQL-коду.

2.5 Проєктування клієнтської частини інтерфейсу

Клієнтська частина вебзастосунку відповідає за те, як користувач взаємодіє з системою, а також за візуальне представлення даних. У даному проєкті вона реалізована з використанням базових вебтехнологій, які забезпечують структурованість сторінок, їх стилізацію та базову інтерактивність.

2.5.1 Технології верстки та стилізації користувацького інтерфейсу

Основою будь-якого вебінтерфейсу є HTML, який відповідає за логічну структуру сторінок. У межах проєкту HTML використовується для формування блоків контенту, таких як списки товарів, сторінки опису продуктів, форма авторизації та кошик користувача. Саме ця технологія визначає, як елементи розташовані на сторінці та як вони взаємодіють між собою на рівні структури документа [13].

Для візуального оформлення використовується CSS, який дозволяє перетворити базову структуру HTML на зручний та естетично привабливий інтерфейс. За допомогою стилів задаються кольори, шрифти, відступи, розміщення елементів та адаптивність сторінок під різні пристрої [14]. Особлива увага приділяється принципам UX/UI, оскільки зручність користування безпосередньо впливає на ефективність взаємодії користувача із системою.

У процесі розробки також враховується адаптивність інтерфейсу, що дозволяє коректно відображати сторінки на екранах різного розміру, включно з мобільними пристроями. Це досягається через використання гнучких сіток, медіа-запитів та відносних одиниць вимірювання у CSS.

Завдяки поєднанню HTML та CSS формується цілісний користувацький інтерфейс, який забезпечує зрозумілу навігацію та комфортну взаємодію з функціоналом системи. Такий підхід дозволяє розділити логічну структуру

сторінок і їх візуальне оформлення, що спрощує подальшу підтримку та розвиток проєкту.

2.5.2 Забезпечення інтерактивності за допомогою JavaScript

Інтерактивна частина вебзастосунку відповідає за динамічну поведінку інтерфейсу та взаємодію користувача з системою без необхідності постійного перезавантаження сторінок. Саме JavaScript дозволяє зробити інтернет-магазин більш “живим”, скорочуючи час відгуку системи та підвищуючи зручність користування [15].

У межах проєкту JavaScript використовується для реалізації ключових сценаріїв взаємодії, зокрема роботи кошика товарів, перевірки введених даних у формах та асинхронного оновлення інформації через AJAX-запити. Наприклад, додавання товару до кошика може відбуватися без перезавантаження сторінки, що створює більш плавний користувацький досвід і зменшує навантаження на сервер.

Особливе значення має використання AJAX-технологій, які дозволяють обмінюватися даними між клієнтською та серверною частинами у фоновому режимі [16]. Це дає можливість оновлювати окремі елементи сторінки, наприклад кількість товарів у кошику або загальну суму замовлення, без повного перезавантаження інтерфейсу.

Детальний механізм такої взаємодії на прикладі модуля кошика представлено на рисунку 2.3.

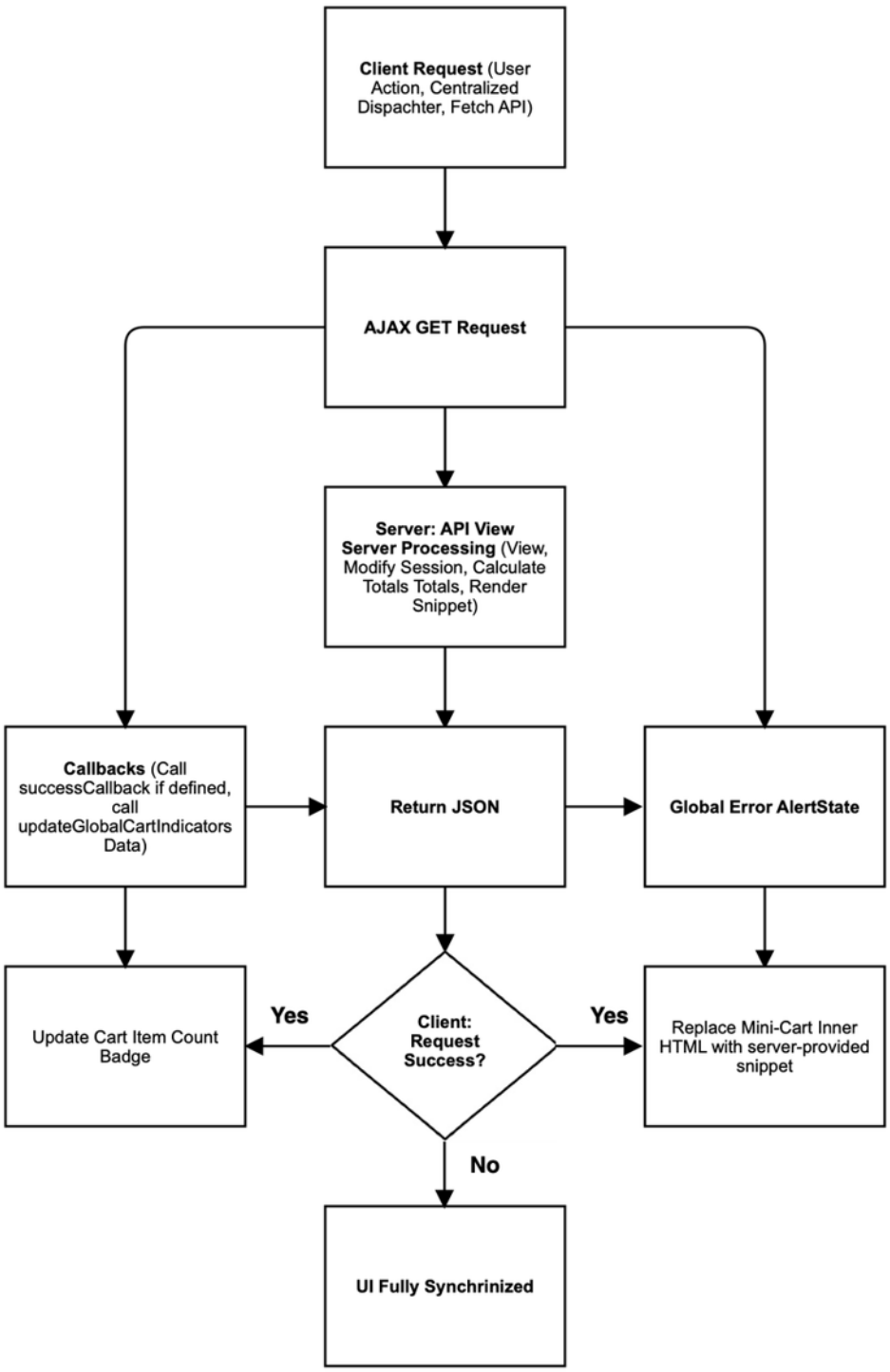


Рисунок 2.3 – Алгоритм обробки AJAX-запиту при оновленні стану кошика

На представленій схемі відображено повний цикл асинхронної операції. Процес розпочинається з виклику Fetch API або виконання AJAX-запиту після дії користувача. Серверна частина (Django API View) обробляє запит, виконує

необхідні операції в сесії або базі даних і повертає відповідь у форматі JSON. Після успішної перевірки статусу запиту на стороні клієнта JavaScript виконує маніпуляції з DOM-деревом: оновлює індикатор кількості товарів (badge) та замінює фрагменти HTML-коду для відображення актуального стану кошика.

Також JavaScript використовується для перевірки коректності заповнення форм. Це дозволяє виявляти помилки ще до відправки даних на сервер, наприклад, порожні поля або некоректні значення у формі реєстрації чи оформлення замовлення. Таким чином зменшується кількість зайвих запитів до backend-частини та підвищується загальна стабільність системи.

У результаті використання JavaScript забезпечує динамічність та інтерактивність вебзастосунку, дозволяючи користувачеві взаємодіяти із системою швидко та без зайвих перезавантажень сторінок. Це підвищує загальну ефективність інтерфейсу та робить роботу з інтернет-магазином більш комфортною.

ВИСНОВОК ДО РОЗДІЛУ

У результаті проведеного аналізу встановлено, що для розробки веборієнтованої системи управління книжковою продукцією доцільним є використання архітектури, яка забезпечує чітке розмежування функцій між рівнями представлення даних, бізнес-логіки та їх збереження. Такий підхід сприяє підвищенню керованості програмного коду, спрощує модернізацію системи та створює передумови для її подальшого масштабування.

Обраний технологічний стек поєднує інструменти, які взаємно доповнюють один одного та дозволяють реалізувати повний цикл функціонування вебзастосунку. Використання Django забезпечує швидку реалізацію серверної логіки та засобів адміністрування, ORM спрощує взаємодію з даними, а PostgreSQL гарантує надійне зберігання інформації та підтримання її цілісності під час виконання операцій.

Застосування HTML, CSS і JavaScript дозволяє створити інтерфейс, який поєднує структуроване представлення інформації, адаптивність та інтерактивність. Це підвищує зручність взаємодії користувача із системою та забезпечує оперативне виконання основних сценаріїв роботи без зайвого навантаження на серверну частину.

Таким чином, обрані технології утворюють узгоджену та ефективну основу для побудови системи управління книжковою продукцією. Вони забезпечують стабільність роботи, зручність розробки та можливість подальшого розширення функціоналу без суттєвих змін архітектури.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1 Архітектура та структура

Структура проєкту організована у вигляді окремих Django-застосунків, кожен із яких реалізує певну функціональність системи. Основними модулями вебзастосунку є:

- модуль користувачів (users);
- модуль товарів (goods);
- модуль кошика (carts);
- модуль замовлень (orders);
- модуль головних сторінок (main).

Такий підхід дозволяє зробити код більш структурованим, спрощує підтримку системи та дає можливість розширювати функціонал без значного впливу на інші компоненти застосунку.

У модулі users реалізовано функціональність авторизації, реєстрації, редагування профілю користувача та роботи з особистим кабінетом. Модуль goods відповідає за відображення каталогу книг, категорій, пошук товарів та сторінки окремого товару. Модуль carts реалізує функціонал кошика покупця, а модуль orders забезпечує створення та обробку замовлень. Модуль main використовується для реалізації головних інформаційних сторінок.

Для організації маршрутів у системі використано окремі файли urls.py для кожного застосунку. Це дозволяє логічно розділити URL-адреси відповідно до функціональних модулів системи.

Для запуску та керування Django-проєктом використовується файл manage.py, який забезпечує виконання адміністративних команд, запуск локального сервера та керування міграціями бази даних.

Взаємодія між серверною та клієнтською частинами застосунку здійснюється через механізм шаблонів Django. Базовий шаблон base.html містить основну структуру сторінок вебзастосунку: навігаційну панель, блок контенту, підключення стилів та JavaScript-файлів. Інші HTML-шаблони наслідують даний шаблон за допомогою механізму template inheritance, що дозволяє уникнути дублювання коду та спрощує підтримку інтерфейсу.

Для зберігання статичних ресурсів у проєкті використовується окрема папка static, у якій розміщено CSS-файли, JavaScript-файли, іконки та інші елементи інтерфейсу. Основні стилі реалізовано у файлі my_css.css, де визначено кольорову палітру, стилізацію навігаційної панелі, кнопок, карток товарів та інших елементів інтерфейсу.

Для реалізації інтерактивної взаємодії без перезавантаження сторінки використано AJAX-запити за допомогою бібліотеки jQuery. Такий підхід застосовується, зокрема, при додаванні товарів у кошик, зміні кількості товарів та видаленні товарів із кошика.

3.2 Реалізація серверної частини

3.2.1 Реалізація модуля користувача

У межах загальної системи модуль користувачів відповідає за управління обліковими записами, включаючи реєстрацію, авторизацію, збереження персональних даних та інтеграцію користувача з іншими частинами системи, зокрема з кошиком та замовленнями. Основою реалізації є розширена модель користувача, побудована на базі стандартного класу AbstractUser, що дозволяє використовувати вбудовану систему автентифікації Django та одночасно додавати власні поля без зміни внутрішньої логіки фреймворку.

У моделі користувача реалізовано додаткові поля для збереження аватара та номера телефону. Під час створення користувача Django ORM автоматично формує SQL-запити на основі Python-об’єктів, що дозволяє працювати з базою

даних без написання SQL-коду вручну. На рисунку 3.1 наведено реалізацію моделі користувача.

```python
from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    image = models.ImageField(upload_to='users_images', blank=True, null=True, verbose_name='Аватар')
    phone_number = models.CharField(max_length=10, blank=True, null=True)
```

Рисунок 3.1 – Модель користувача

Механізм авторизації користувача реалізовано у вигляді окремої view-функції, яка обробляє POST-запит із форми входу. Після отримання даних система перевіряє їх за допомогою функції authenticate, яка звертається до бази даних користувачів і виконує перевірку логіна та пароля шляхом порівняння хешованого значення пароля з тим, що збережений у базі даних. У разі успішної автентифікації створюється сесія користувача через login, яка фіксує його стан у межах поточного сеансу.

Особливістю реалізації є додаткова логіка інтеграції процесу автентифікації із кошиком. Після успішного входу система зчитує session_key анонімного сеансу для ідентифікації товарів, які користувач додав до моменту авторизації. Якщо в базі даних виявляються тимчасові записи, прив'язані до цього ключа, вони автоматично мігрують до облікового запису користувача (або видаляються у разі виникнення конфліктів із уже наявним кошиком). Це забезпечує безперервність клієнтського досвіду під час переходу від анонімного перегляду до авторизованого сеансу.

Також реалізовано механізм редиректу користувача після входу в систему. Якщо у POST-запиті передано параметр next, користувач перенаправляється на попередню сторінку, з якої було здійснено перехід до авторизації. У випадку відсутності цього параметра виконується перенаправлення на головну сторінку системи.

На рисунку 3.2 наведено реалізацію функції авторизації користувача з інтеграцією сесій та кошика.

```python
def login(request) -> HttpResponseRedirect | HttpResponse:
    if request.method == 'POST':
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            username: Any = request.POST['username']
            password: Any = request.POST['password']
            user: AbstractBaseUser | None = auth.authenticate(username=username, password=password)

            session_key: Any = request.session.session_key

            if user:
                auth.login(request, user)
                messages.success(request, f"{username}, Ви увійшли до облікового запису")

                if session_key:
                    forgot_carts: BaseManager[Cart] = Cart.objects.filter(user=user)
                    if forgot_carts.exists():
                        forgot_carts.delete()

                    Cart.objects.filter(session_key=session_key).update(user=user)

                redirect_page: Any = request.POST.get('next', None)
                if redirect_page and redirect_page != reverse('user:logout'):
                    return HttpResponseRedirect(request.POST.get('next'))

                return HttpResponseRedirect(reverse('main:index'))
    else:
        form = UserLoginForm()

    context: dict[str, Any] = {
        'title': 'Bookstore - Авторизація',
        'form': form
    }
    return render(request, 'users/login.html', context)
```

Рисунок 3.2 – Реалізація функції авторизації користувача з обробкою сесій та кошика

Реєстрація користувача реалізована на основі форми UserRegistrationForm, яка наслідує стандартний клас Django UserCreationForm. Під час обробки запиту сервер виконує валідацію даних на рівні форми, включаючи перевірку унікальності імені користувача та коректності пароля. Після успішної перевірки Django ORM створює новий запис у таблиці користувачів, автоматично застосовуючи хешування пароля перед збереженням. Одразу після створення облікового запису система виконує автоматичну авторизацію користувача, що дозволяє перейти до використання функціоналу без повторного введення даних входу.

Окремо реалізовано функціональність профілю користувача, яка забезпечує оновлення персональних даних. При відкритті сторінки профілю форма ініціалізується поточними даними користувача, отриманими з бази даних через ORM. Після відправки змін сервер виконує повторну валідацію та оновлює відповідний запис у базі даних. Таким чином забезпечується синхронізація змін між інтерфейсом користувача та серверною частиною системи.

Доступ до персональних сторінок обмежується за допомогою декоратора @login_required, який перевіряє наявність активної сесії перед виконанням view-функції та у разі її відсутності виконує перенаправлення на сторінку авторизації.

3.2.2 Реалізація кошика та замовлень

Реалізація кошика в клієнт-серверній системі управління товарами побудована таким чином, щоб забезпечити однакову поведінку функціоналу для авторизованих і неавторизованих користувачів. Для цього в системі використовується комбінований підхід збереження стану кошика: для авторизованих користувачів дані прив’язуються до облікового запису через зовнішній ключ user, а для неавторизованих – через session_key, який зберігає ідентифікатор поточної сесії браузера.

Основна модель кошика реалізована як Django-модель Cart, яка містить зв’язки з користувачем і товаром, а також поле кількості товару та часову мітку створення запису. Така структура дозволяє однозначно ідентифікувати кожен запис кошика та контролювати його стан у межах системи. На рівні бази даних модель представляє таблицю cart, яка містить записи як для авторизованих, так і для анонімних користувачів.

На рисунку 3.3 наведено реалізацію моделі кошика та її основних зв’язків у системі.

```python
class CartQueryset(models.QuerySet):

    def total_price(self) -> int:
        return sum(cart.products_price() for cart in self)

    def total_quantity(self) -> int:
        if self:
            return sum(cart.quantity for cart in self)
        return 0

class Cart(models.Model):

    user = models.ForeignKey(to=User, on_delete=models.CASCADE, blank=True, null=True, verbose_name='Користувач')
    product = models.ForeignKey(to=Products, on_delete=models.CASCADE, verbose_name='Товар')
    quantity = models.PositiveSmallIntegerField(default=0, verbose_name='Кількість')
    session_key = models.CharField(max_length=32, null=True, blank=True)
    created_timestamp = models.DateTimeField(auto_now_add=True, verbose_name='Дата додання')
```

Рисунок 3.3 – Реалізація моделі кошика та зв’язків між користувачем і товарами

Окремо в моделі реалізовано кастомний QuerySet, який дозволяє виконувати агрегаційні операції над кошиком безпосередньо на рівні ORM. Зокрема, метод total_price() виконує підрахунок загальної вартості всіх товарів у кошику шляхом множення ціни товару на його кількість для кожного запису, а метод total_quantity() повертає загальну кількість одиниць товарів. Такий підхід дозволяє уникнути зайвих обчислень у view-функціях і переносить логіку на рівень моделі.

Функціональність додавання товару до кошика реалізована у view-функції cart_add, яка обробляє POST-запит із ідентифікатором товару. Після отримання product_id система виконує запит до бази даних через ORM і отримує відповідний об’єкт товару. Далі логіка розгалужується залежно від того, чи авторизований користувач.

У випадку авторизованого користувача виконується пошук запису кошика за комбінацією user і product. Якщо запис уже існує, система збільшує значення поля quantity на одиницю та зберігає зміни. Якщо запис відсутній – створюється новий об’єкт кошика з початковою кількістю 1. Аналогічна логіка реалізована для неавторизованого користувача, однак замість user використовується session_key, що дозволяє зберігати тимчасовий стан кошика.

На рисунку 3.4 наведено реалізацію функції додавання товару до кошика з урахуванням сесії та авторизації користувача.

```python
def cart_add(request) -> JsonResponse:

    product_id: Any = request.POST.get("product_id")

    product: Products = Products.objects.get(id=product_id)

    if request.user.is_authenticated:
        carts: BaseManager[Cart] = Cart.objects.filter(user=request.user, product=product)

        if carts.exists():
            cart: Cart | None = carts.first()
            if cart:
                cart.quantity += 1
                cart.save()
        else:
            Cart.objects.create(user=request.user, product=product, quantity=1)

    else:
        carts: BaseManager[Cart] = Cart.objects.filter(
            session_key=request.session.session_key, product=product)

        if carts.exists():
            cart: Cart | None = carts.first()
            if cart:
                cart.quantity += 1
                cart.save()
        else:
            Cart.objects.create(
                session_key=request.session.session_key, product=product, quantity=1)

    user_cart: BaseManager[Cart] = get_user_carts(request)
    cart_items_html: str = render_to_string(
        "carts/includes/included_cart.html", {"carts": user_cart}, request=request)

    response_data: dict[str, str] = {
        "message": "Товар доданий до кошика",
        "cart_items_html": cart_items_html,
    }

    return JsonResponse(response_data)
```

Рисунок 3.4 – Реалізація функції додавання товару до кошика

Після оновлення стану кошика система додатково формує актуальний список товарів користувача за допомогою допоміжної функції get_user_carts(). Отримані дані передаються до шаблону included_cart.html, який рендериться на сервері та повертається у вигляді HTML-рядка. Далі цей HTML включається у JSON-відповідь і передається на клієнтську сторону.

Такий підхід дозволяє реалізувати асинхронне оновлення кошика без повного перезавантаження сторінки, оскільки клієнт отримує вже готовий HTML-блок і оновлює лише відповідну частину інтерфейсу. Це підвищує

зручність використання системи та зменшує навантаження на сервер за рахунок часткового оновлення даних.

Наступним етапом логіки є формування замовлення, яке базується на поточному стані кошика. Під час оформлення замовлення всі товари з кошика переносяться у пов'язану сутність OrderItem, після чого кошик очищається. Таким чином створюється фіксований запис замовлення, який не змінюється навіть у випадку подальшого оновлення кошика або товарів у системі. Це забезпечує цілісність даних і дозволяє зберігати історію покупок користувача.

3.2.3 Реалізація каталогу товарів

Основою каталогу виступає модель Products, яка визначає структуру зберігання товарів у базі даних. Вона містить атрибути, що описують товар: назву, slug для формування URL, опис, зображення, ціну, знижку, кількість та категорію. Окрім зберігання даних, у моделі реалізовано логіку обробки бізнес-правил. Зокрема, метод sell_price() виконує розрахунок фінальної ціни товару з урахуванням знижки, що дозволяє уникнути дублювання обчислень у різних частинах системи та централізувати логіку формування вартості безпосередньо в моделі. На рисунку 3.5 наведено реалізацію моделі товару та методів обробки даних.

```
21  class Products(models.Model):
22      name = models.CharField(max_length=150, unique=True, verbose_name='Назва')
23      slug = models.SlugField(max_length=200, unique=True, blank=True, null=True, verbose_name='URL')
24      description = models.TextField(blank=True, null=True, verbose_name='Опис')
25      image = models.ImageField(upload_to='goods_images', blank=True, null=True, verbose_name='Зображення')
26      price = models.DecimalField(default=0.00, max_digits=7, decimal_places=2, verbose_name='Ціна')
27      discount = models.DecimalField(default=0.00, max_digits=4, decimal_places=2, verbose_name='Знижка в %')
28      quantity = models.PositiveIntegerField(default=0, verbose_name='Кількість')
29      category = models.ForeignKey(to=Categories, on_delete=models.CASCADE, verbose_name='Категорія')
30
31
32      def sell_price(self) -> Any:
33          if self.discount:
34              return round(self.price - self.price*self.discount/100, 2)
35
36          return self.price
```

Рисунок 3.5 – Реалізація моделі Products та методів обчислення вартості

Формування самого каталогу реалізовано у функції catalog, яка відповідає за обробку HTTP-запитів та побудову вибірки товарів відповідно до параметрів, переданих користувачем. Логіка функції дозволяє реалізувати одразу декілька сценаріїв роботи системи: перегляд усіх товарів, перегляд товарів певної категорії, застосування пошукового запиту, а також додаткову фільтрацію та сортування результатів.

У межах функції спочатку аналізуються параметри GET-запиту, після чого формується відповідний QuerySet. Якщо передано параметр пошуку, використовується окрема функція q_search, яка реалізує повнотекстовий пошук із використанням PostgreSQL механізмів SearchVector, SearchQuery та SearchRank. Такий підхід дозволяє не лише знаходити збіги за ключовими словами, але й ранжувати результати за релевантністю, що підвищує якість пошуку в межах каталогу.

На рисунку 3.6 наведено реалізацію обробки логіки каталогу з фільтрацією та пошуком.

```python
def catalog(request, category_slug=None) -> HttpResponse:

    page: Any = request.GET.get('page', 1)
    on_sale: Any = request.GET.get('on_sale', None)
    order_by: Any = request.GET.get('order_by', None)
    query: Any = request.GET.get('q', None)


    if category_slug == 'all':
        goods: BaseManager[Products] = Products.objects.all()
    elif query:
        goods: BaseManager[Products] = q_search(query)
    else:
        goods: BaseManager[Products] = Products.objects.filter(category__slug=category_slug)
        if not goods.exists():
            raise Http404()

    if on_sale:
        goods: BaseManager[Products] = goods.filter(discount__gt=0)

    if order_by and order_by != "default":
        goods: BaseManager[Products] = goods.order_by(order_by)

    paginator = Paginator(goods, 8)
    current_page: Page = paginator.page(int(page))
```

Рисунок 3.6 – Реалізація логіки формування каталогу товарів

Важливу роль у реалізації каталогу відіграють користувацькі template tags, які забезпечують гнучкість відображення даних у шаблонах. Функція tag_categories() дозволяє отримувати список усіх категорій товарів для формування навігаційного меню, а функція change_params() забезпечує можливість динамічного оновлення GET-параметрів без втрати вже застосованих фільтрів. Це дозволяє реалізувати зручну навігацію між сторінками каталогу із збереженням стану фільтрації.

Окремо в межах каталогу реалізовано механізм пошуку товарів через функцію q_search, яка використовує можливості повнотекстового пошуку PostgreSQL. На першому етапі виконується побудова вектору пошуку за полями назви та опису товару, після чого формується запит і обчислюється ранг відповідності. Результати сортуються за релевантністю, що дозволяє відображати найбільш відповідні товари у верхній частині списку.

3.2.4 Налаштування адміністративної панелі

Адміністративна панель у клієнт-серверній системі управління товарами реалізована на основі стандартного Django Admin, який було розширено та налаштовано відповідно до структури предметної області. Налаштування адміністративної панелі виконується у файлі admin.py для кожної сутності окремо. Для моделі категорій Categories реалізовано базову конфігурацію відображення, яка включає виведення назви категорії у списку об’єктів, а також автоматичне формування URL-ідентифікатора (slug) на основі назви. Це дозволяє значно пришвидшити процес створення нових категорій та зменшити кількість ручних помилок при заповненні даних.

Для моделі товарів Products адміністративна панель має більш розширену конфігурацію, оскільки саме ця сутність є ключовою у системі. Тут налаштовано відображення основних полів у списку об’єктів, таких як назва товару, кількість, ціна та знижка. Додатково реалізовано можливість

редагування поля знижки без переходу в детальну форму об’єкта, що спрощує масове оновлення даних. Також налаштовано пошук товарів за назвою та описом, а фільтрацію – за категоріями, наявністю знижки та залишками товару. Це дозволяє адміністратору швидко знаходити необхідні записи навіть при великій кількості товарів у базі даних.

На рисунку 3.7 наведено конфігурацію адміністративного класу для моделі товарів.

```python
@admin.register(Products)
class ProductsAdmin(admin.ModelAdmin):
    prepopulated_fields: dict[str, tuple[Literal['na... = {"slug": ("name",)}
    list_display: list[str] = ["name", "quantity", "price", "discount"]
    list_editable: list[str] = ["discount"]
    search_fields: list[str] = ["name", "description"]
    list_filter: list[str] = ["discount", "quantity", "category"]
    fields: list[tuple[Literal['price']... = [
        "name",
        "category",
        "slug",
        "description",
        "image",
        ("price", "discount"),
        "quantity",
    ]
```

Рисунок 3.7 – Налаштування адміністративної панелі для моделі Products

Окремо в адміністративній панелі реалізовано керування замовленнями та кошиком користувача. Для моделі Order налаштовано відображення основної інформації про замовлення, включаючи користувача, статус, спосіб оплати, стан оплати та дату створення. Додатково реалізовано фільтрацію за статусом замовлення, типом доставки та станом оплати, що дозволяє швидко знаходити необхідні записи та контролювати процес обробки замовлень.

Особливістю реалізації є використання inline-моделей, які дозволяють відображати товари замовлення (OrderItem) безпосередньо у формі замовлення. Це забезпечує зручне адміністрування пов’язаних даних у межах одного інтерфейсу та спрощує роботу адміністратора із замовленнями користувачів.

На рисунку 3.8 наведено налаштування адміністративної панелі для моделі замовлень.

```python
class OrderItemTabulareAdmin(admin.TabularInline):
    model = OrderItem
    fields: tuple[Literal['product'], L... = "product", "name", "price", "quantity"
    search_fields: tuple[Literal['product'], L... = (
        "product",
        "name",
    )
    extra = 0

@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display: tuple[Literal['id'], Litera... = (
        "id",
        "user",
        "requires_delivery",
        "status",
        "payment_on_get",
        "is_paid",
        "created_timestamp",
    )

    search_fields: tuple[Literal['id']] = (
        "id",
    )
    readonly_fields: tuple[Literal['created_time... = ("created_timestamp",)
    list_filter: tuple[Literal['requires_del... = (
        "requires_delivery",
        "status",
        "payment_on_get",
        "is_paid",
    )
    inlines: tuple[type[OrderItemTabular... = (OrderItemTabulareAdmin,)
```

Рисунок 3.8 – Налаштування адміністративної панелі для моделі Order

Додатково реалізовано інтеграцію адміністративної панелі користувачів із кошиком та замовленнями через механізм inlines. Це дозволяє переглядати пов’язані кошики та замовлення безпосередньо на сторінці конкретного користувача, що спрощує аналіз активності користувачів та адміністрування системи.

Окремо налаштовано адміністративну частину для моделі відгуків Review, яка дозволяє контролювати користувацький контент у системі. У списку відгуків відображаються користувач, товар, рейтинг та дата створення, що забезпечує швидкий аналіз зворотного зв’язку. Додатково реалізовано можливість фільтрації за рейтингом та датою, а також пошук за текстом відгуку, іменем користувача та назвою товару. Така конфігурація дозволяє ефективно опрацьовувати відгуки та контролювати якість контенту в системі.

3.3 Розробка клієнтської частини

3.3.1 Розробка користувацького інтерфейсу (HTML + CSS)

Розробка користувацького інтерфейсу виконувалася з використанням HTML як основної технології для формування структури сторінок та CSS для їх стилізації. У межах проєкту застосовано підхід шаблонізації Django, що дозволяє розділяти логіку представлення даних і їх візуальне відображення, а також забезпечує повторне використання компонентів інтерфейсу.

Структура інтерфейсу реалізована не як набір окремих ізольованих HTML-сторінок, а як система взаємопов'язаних Django-шаблонів. Основу всього інтерфейсу становить базовий шаблон base.html, який визначає загальну структуру вебзастосунку, включаючи підключення стилів, навігаційне меню, області контенту. Водночас у проєкті використовується велика кількість дочірніх шаблонів для різних сторінок і функціональних розділів, зокрема каталогу товарів, сторінки продукту, кошика, оформлення замовлення, авторизації, реєстрації, профілю користувача та інформаційних сторінок.

Завдяки використанню механізму наслідування шаблонів Django ({% block content %}, {% block modal_cart %} та інших) забезпечується єдина структура інтерфейсу на всіх сторінках, при цьому кожен шаблон відповідає за власну функціональну частину. Окремі повторно використовувані елементи інтерфейсу винесені в окремі шаблони, що підключаються за допомогою механізму include (наприклад, компоненти кошика та системні повідомлення), що дозволяє уникати дублювання коду та спрощує підтримку і масштабування проєкту.

Навігаційна частина інтерфейсу реалізована у межах цього ж шаблону та включає каталог товарів, пошук, інформаційні сторінки та блок авторизації користувача. Важливою особливістю є використання умовного рендерингу Django, який дозволяє змінювати доступні елементи інтерфейсу залежно від

статусу користувача, зокрема відображати або приховувати адміністративні функції.

Основним елементом користувацького інтерфейсу є сторінка каталогу товарів (catalog.html), яка реалізує відображення продукції у вигляді адаптивної сітки карток. Для побудови макету використовується Bootstrap Grid-система, що забезпечує коректне відображення контенту на різних типах пристроїв [17].

На рисунку 3.9 наведено фрагмент реалізації картки товару в каталозі.

```
<div class="row g-4">
    {% for product in goods %}
    <div class="col-xl-3 col-lg-3 col-md-6">
        <div class="card h-100 border-0 custom-shadow">
            <a href="{% url "catalog:product" product.slug %}">
                {% if product.image %}
                    <img src="{{ product.image.url }}"
                         class="card-img-top"
                         alt="{{ product.name }}">
                {% else %}
                    <img src="{% static "deps/images/Not found image.png" %}"
                         class="card-img-top"
                         alt="Зображення відсутнє">
                {% endif %}
            </a>

            <div class="card-body d-flex flex-column">
                <div class="mb-2">
                    <p class="text-muted mb-2 small">
                        ID: {{ product.display_id }}
                    </p>

                    <a href="{% url "catalog:product" product.slug %}">
                        <h5 class="card-title mb-2">
                            {% if product.headline %}
                                {{ product.headline|safe }}
                            {% else %}
                                {{ product.name }}
                            {% endif %}
                        </h5>
                    </a>
                </div>

                <p class="card-text text-muted mb-3">
                    {% if product.bodyline %}
                        {{ product.bodyline|safe|truncatechars:90 }}
                    {% else %}
                        {{ product.description|truncatechars:90 }}
                    {% endif %}
                </p>
```

Рисунок 3.9 – Фрагмент HTML-шаблону картки товару в каталозі

Таким чином, інтерфейс каталогу побудований за компонентним принципом, де кожна картка товару є автономним елементом, що дозволяє легко масштабувати кількість товарів без зміни логіки відображення. Окремо слід відзначити, що сторінка підтримує динамічну фільтрацію та сортування, однак ці механізми не впливають на структуру HTML, а лише змінюють набір даних, що передається у шаблон.

Окрему роль у формуванні інтерфейсу відіграє CSS-стилізація, яка реалізована у файлі my_css.css. Вона доповнює базову систему Bootstrap і формує єдиний візуальний стиль застосунку. Основний підхід полягає у використанні CSS-змінних, що дозволяє централізовано керувати кольоровою схемою, забезпечує узгодженість стилів у всьому застосунку та спрощує подальшу модифікацію дизайну.

У проєкті реалізовано власну систему стилістичних змінних (design tokens), які визначають базові кольори інтерфейсу, акцентні елементи та параметри оформлення компонентів. На їх основі формуються всі основні стилі інтерфейсу – від типографіки до кнопок, карток товарів та навігаційних елементів. Особлива увага приділена узгодженості компонентів: картки мають округлені краї, тінь і ефект підняття при наведенні, а кнопки – плавну зміну кольору при взаємодії користувача.

На рисунку 3.10 наведено фрагмент реалізації CSS-змінних та базових стилів інтерфейсу.

```css
:root {
  --bg: #F6F1E8;
  --text: #1E1E1E;
  --muted: #6B6B6B;
  --card: #FFFFFF;
  --border: #E6DED2;

  --accent: #3F5E5A;
  --accent-hover: #4E6F6A;
  --soft-hover: #EFE7DA;

  --shadow: rgba(0, 0, 0, 0.08);
}

body {
  background-color: var(--bg);
  color: var(--text);
  font-family: "Inter", "Segoe UI", sans-serif;
}

.card {
  background-color: var(--card);
  border: 1px solid var(--border) !important;
  border-radius: 16px;
  box-shadow: 0 6px 18px var(--shadow);
  transition: 0.2s ease;
}
```

Рисунок 3.10 – Реалізація CSS-змінних та базових стилів інтерфейсу

3.3.2 Реалізація динамічної взаємодії інтерфейсу

Динамічна взаємодія клієнтської частини вебзастосунку реалізована за допомогою JavaScript та бібліотеки jQuery із застосуванням технології AJAX, що дозволяє організувати асинхронний обмін даними між клієнтом і сервером без необхідності повного перезавантаження сторінки [18].

Основна ідея реалізації полягає у тому, що всі ключові дії користувача (додавання товарів, зміна кількості, відкриття кошика та оформлення замовлення) обробляються через події JavaScript. Після цього формується AJAX-запит до серверної частини, яка повертає актуальні дані у вигляді JSON або HTML-фрагментів для часткового оновлення сторінки.

Одним із центральних сценаріїв є додавання товару до кошика. Після натискання відповідної кнопки виконується POST-запит, який передає

ідентифікатор товару та CSRF-токен для забезпечення безпеки запиту. На рисунку 3.11 наведено реалізацію даного механізму.

```
$(document).on("click", ".add-to-cart", function (e) {
    e.preventDefault();

    var goodsInCartCount = $("#goods-in-cart-count");
    var cartCount = parseInt(goodsInCartCount.text() || 0);

    var product_id = $(this).data("product-id");
    var add_to_cart_url = $(this).attr("href");

    $.ajax({
        type: "POST",
        url: add_to_cart_url,
        data: {
            product_id: product_id,
            csrfmiddlewaretoken: $("[name=csrfmiddlewaretoken]").val(),
        },
        success: function (data) {
            successMessage.html(data.message);
            successMessage.fadeIn(400);
            setTimeout(function () {
                successMessage.fadeOut(400);
            }, 7000);

            cartCount++;
            goodsInCartCount.text(cartCount);

            var cartItemsContainer = $("#cart-items-container");
            cartItemsContainer.html(data.cart_items_html);
        },
```

Рисунок 3.11 – AJAX-реалізація додавання товару до кошика

Після отримання відповіді від сервера система виконує одразу кілька дій: оновлює лічильник товарів у кошику, замінює HTML-вміст блоку кошика та відображає користувачу повідомлення про успішне виконання операції. Таким чином забезпечується миттєва реакція інтерфейсу без перезавантаження сторінки.

Окремим елементом динамічної взаємодії є система користувацьких повідомлень та модального відображення кошика. Повідомлення використовуються для інформування користувача про результат виконаних дій, а модальне вікно дозволяє переглядати кошик без переходу на окрему сторінку, зберігаючи контекст поточного перегляду каталогу.

Наступним важливим елементом динамічної взаємодії є робота з формою оформлення замовлення. У системі реалізовано умовне відображення поля

адреси доставки залежно від обраного способу отримання товару. Це дозволяє адаптувати інтерфейс форми під конкретний сценарій використання та зменшити кількість зайвих полів.

Окрім цього, у межах форми оформлення замовлення реалізовано клієнтську валідацію номера телефону (рис. 3.12). Під час введення даних виконується автоматичне форматування у заданий шаблон, а при відправці форми здійснюється перевірка відповідності регулярному виразу. У випадку помилки відправка блокується, що дозволяє запобігти потраплянню некоректних даних на сервер.

```
199     document.getElementById('id_phone_number').addEventListener('input', function (e) {
200         var x = e.target.value.replace(/\D/g, '').match(/(\d{0,3})(\d{0,3})(\d{0,4})/);
201         e.target.value = !x[2] ? x[1] : '(' + x[1] + ') ' + x[2] + (x[3] ? '-' + x[3] : '');
202     });
203 
204     $('#create_order_form').on('submit', function (event) {
205         var phoneNumber = $('#id_phone_number').val();
206         var regex = /^\(\d{3}\) \d{3}-\d{4}$/;
207 
208         if (!regex.test(phoneNumber)) {
209             $('#phone_number_error').show();
210             event.preventDefault();
211         } else {
212             $('#phone_number_error').hide();
213 
214             var cleanedPhoneNumber = phoneNumber.replace(/[()\-\s]/g, '');
215             $('#id_phone_number').val(cleanedPhoneNumber);
216         }
217     });
```

Рисунок 3.12 – Реалізація клієнтської валідації та форматування номера телефону

ВИСНОВОК ДО РОЗДІЛУ

У результаті проєктування та розробки системи сформовано цілісну програмну архітектуру, яка забезпечує узгоджену взаємодію всіх функціональних компонентів інтернет-магазину. Обраний підхід до побудови структури проєкту дозволяє ізолювати окремі підсистеми, що спрощує супровід програмного забезпечення, підвищує його зрозумілість та створює передумови для подальшого розширення функціональних можливостей.

Реалізовані механізми роботи з користувачами, товарами, кошиком і замовленнями забезпечують цілісність бізнес-процесів та коректну обробку даних на всіх етапах взаємодії із системою. Використання засобів Django дозволило підвищити надійність обробки інформації, мінімізувати ризик помилок під час роботи з базою даних і забезпечити належний рівень безпеки доступу до функціоналу застосунку.

Важливим результатом розробки є створення інтерфейсу, орієнтованого на зручність користувача. Поєднання шаблонізації, адаптивного дизайну та асинхронного обміну даними забезпечує швидке виконання основних операцій і покращує комфорт роботи з вебзастосунком незалежно від типу пристрою.

У підсумку розроблене програмне рішення відповідає встановленим вимогам щодо функціональності, зручності використання та масштабованості. Запропонована архітектура забезпечує гнучкість системи та дозволяє надалі впроваджувати нові можливості без значного перероблення вже створених компонентів.

РОЗДІЛ 4. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Інтерфейс та навігація

Інтерфейс розробленого інтернет-магазину побудований на основі шаблонної архітектури Django із використанням єдиного базового шаблону, який застосовується для всіх сторінок системи. Основні елементи інтерфейсу винесені у верхню навігаційну панель, яка залишається спільною для всіх розділів сайту та містить ключові точки переходу між функціональними модулями. Загальна структура сторінок сформована таким чином, щоб користувач мав доступ до основних розділів незалежно від поточної сторінки.

Головна сторінка є стартовою точкою взаємодії з системою та містить основні елементи навігації, включаючи доступ до каталогу товарів, інформаційних сторінок, профілю користувача та кошика. Загальний вигляд головної сторінки з верхньою панеллю наведено на рисунку 4.1.

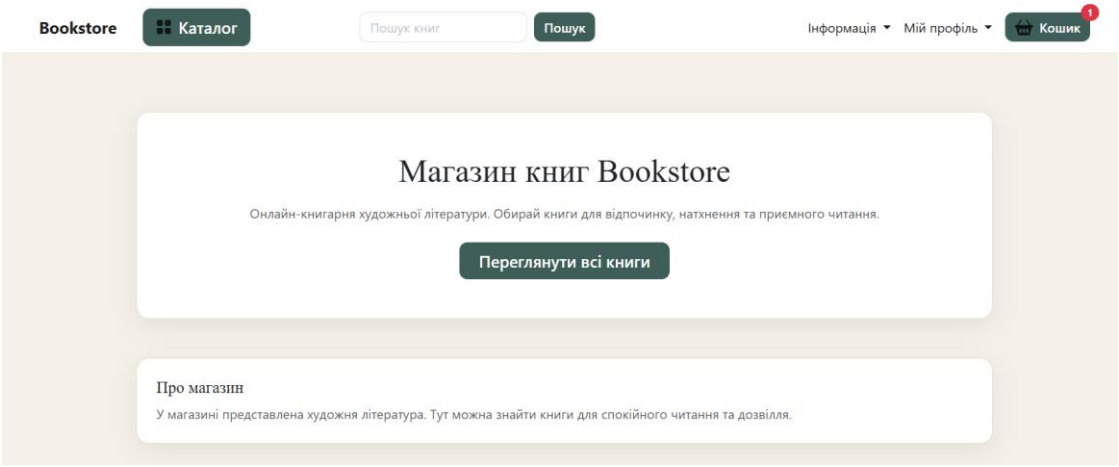


Рисунок 4.1 – Головна сторінка інтернет-магазину з верхньою навігаційною панеллю

Одним із ключових елементів навігації є розділ «Каталог», який реалізовано у вигляді випадаючого меню (рис. 4.2). У ньому представлено перелік категорій товарів, що відповідають жанрам книг. При виборі конкретної

категорії здійснюється перехід до сторінки каталогу з відповідним набором товарів.

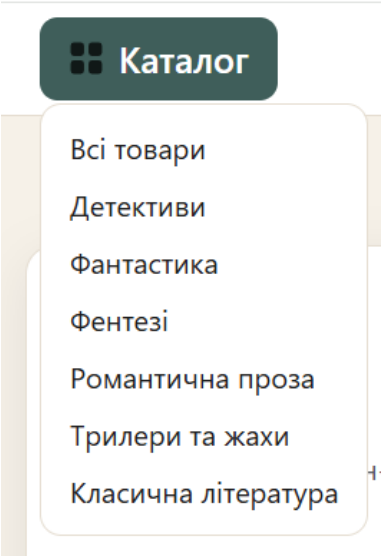


Рисунок 4.2 – Випадаюче меню “Каталог” з переліком категорій товарів

Окрім каталогу, у навігаційній панелі присутній інформаційний розділ, який містить сторінки з додатковими відомостями про інтернет-магазин. До нього входять сторінки «Доставка та оплата», «Контакти» та «Про нас». Кожна зі сторінок реалізована як окремий статичний маршрут та містить відповідну текстову інформацію, доступну через випадаюче меню.

Також у верхній панелі розміщено елементи, що відповідають за роботу з користувацьким обліковим записом та кошиком. Залежно від стану авторизації користувача відображається або меню профілю, або кнопки входу та реєстрації.

4.2 Каталог товарів, пошук та перегляд книг

Каталог товарів реалізовано як основний розділ інтернет-магазину, у якому відображається перелік доступних книг. Дані формуються динамічно на основі записів у базі даних та виводяться у вигляді карток товарів. Кожна картка містить зображення книги, її назву, id, частину опису товару, оцінку товару,

ціну та розмір знижки, після чого користувач може перейти на сторінку детального перегляду товару (рис. 4.3).

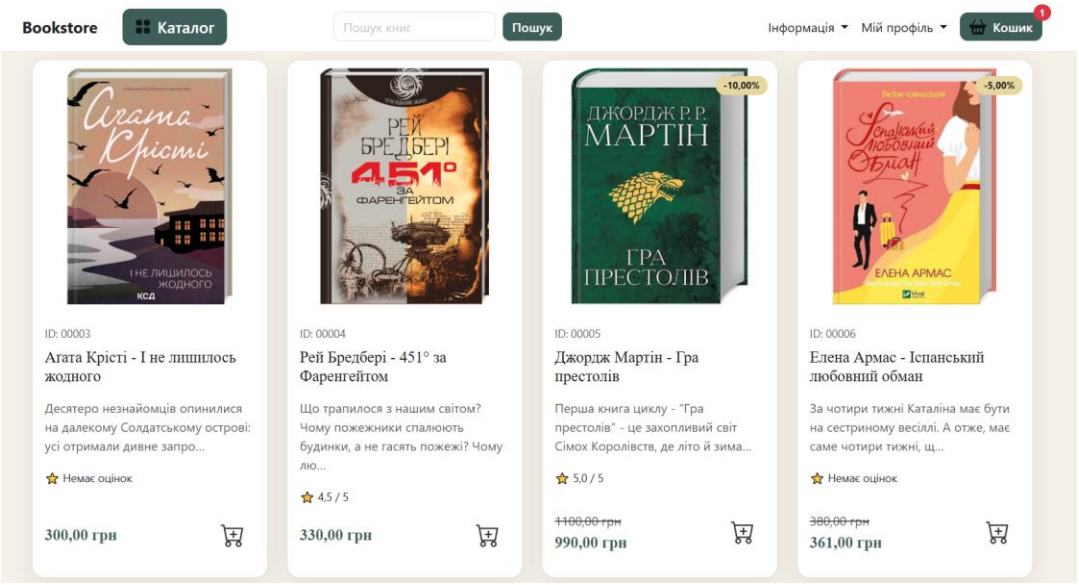


Рисунок 4.3 – Сторінка каталогу товарів інтернет-магазину

Сторінка окремої книги містить розширену інформацію про товар (рис. 4.4). Також реалізовано можливість додавання товару до кошика. Логіка відображення побудована на зв’язках між моделями товарів та замовлень, що відповідає структурі, описаній у розділі серверної частини застосунку.

Рисунок 4.4 – Сторінка товару (картка товару з основною інформацією)

Окремим елементом сторінки товару є система відгуків і оцінювання. Для кожного товару зберігаються користувацькі відгуки, що містять текстовий коментар та оцінку. Дані прив’язані до конкретного користувача та товару через відповідні зв’язки в базі даних. Також реалізовано можливість видалення власних відгуків, що виконується через перевірку авторства запису (рис. 4.5).

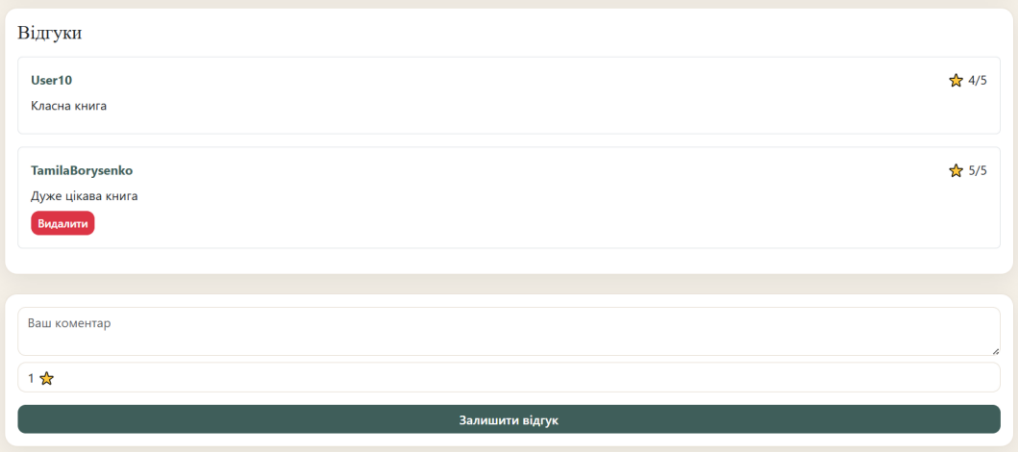


Рисунок 4.5 – Блок відгуків та оцінок користувачів на сторінці товару

Фільтрація товарів реалізована на рівні запитів до бази даних і включає сортування за ціною у двох напрямках (від меншої до більшої та навпаки), а також відбір акційних товарів. У залежності від вибраного параметра формується відповідний queryset, який передається у шаблон для відображення каталогу.

Пошук товарів реалізовано через перевірку даних у декількох полях моделі товару, зокрема назви та опису. Результати формуються на рівні ORM-запитів і повертаються у вигляді оновленого списку товарів. При введенні пошукового запиту знайдені збіги відображаються у стандартному вигляді каталогу з додатковим підсвічуванням відповідних фрагментів тексту (рис. 4.6). У випадку, коли за заданими параметрами фільтрації або пошуку відсутні результати, відображається окрема сторінка з відповідним повідомленням про відсутність записів у вибірці.

Рисунок 4.6 – Результати пошуку товарів із підсвічуванням збігів

4.3 Користувацький обліковий запис

Користувацький обліковий запис охоплює процеси реєстрації, авторизації та роботи з особистими даними користувача. Він є основою для доступу до персоналізованих можливостей інтернет-магазину та взаємодії з функціоналом системи від імені конкретного користувача.

Створення облікового запису виконується через окрему форму реєстрації, у якій користувач вводить основні персональні дані та дані для входу. Під час введення інформації застосовується перевірка коректності заповнення полів. Зокрема, пароль проходить базову перевірку на відповідність вимогам безпеки, і у разі невідповідності можуть відображатися повідомлення про те, що пароль є занадто коротким, занадто поширеним або складається лише з цифр. Також перевіряється коректність введення електронної пошти, зокрема наявність символу «@» та загальний формат адреси. У випадку помилок у заповненні обов’язкових полів користувачу відображаються відповідні повідомлення з вимогою коректного введення даних. Сторінка реєстрації наведена на рисунку 4.7.

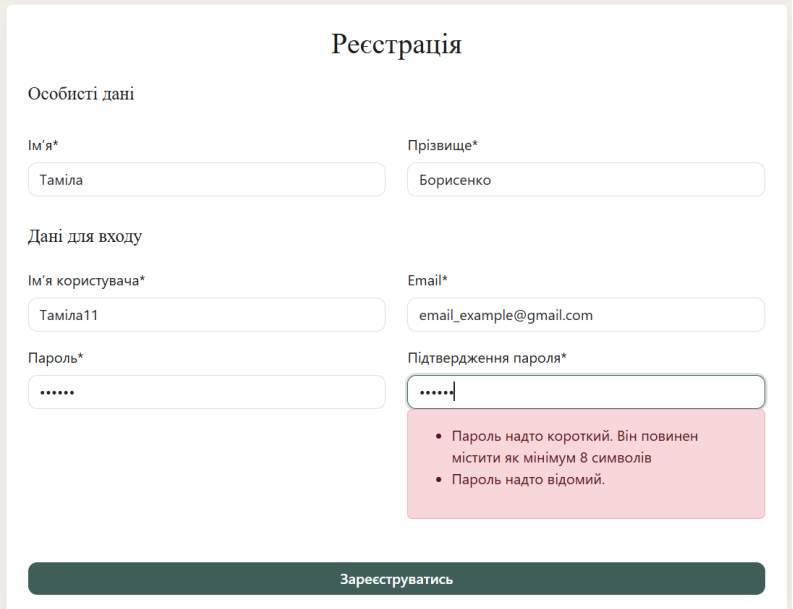


Рисунок 4.7 – Сторінка реєстрації користувача з перевіркою введених даних

Авторизація користувача виконується через форму входу, де вводяться ім’я користувача та пароль. Під час входу враховується чутливість до регістру, тому неправильне написання навіть одного символу призводить до відхилення авторизації. У разі помилки відображається повідомлення про необхідність введення правильних даних. Після успішного входу користувач отримує доступ до особистого кабінету. Інтерфейс сторінки входу представлено на рисунку 4.8.

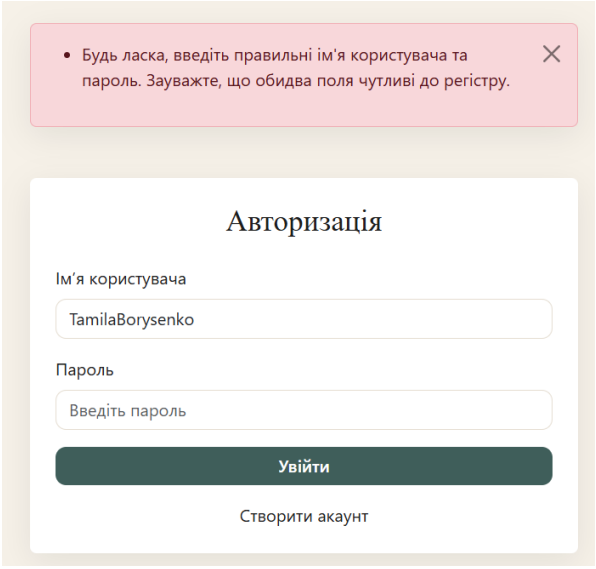


Рисунок 4.8 – Сторінка авторизації користувача

У системі реалізовано механізм інформаційних повідомлень, які супроводжують основні дії користувача. До них належать повідомлення про успішну реєстрацію, успішний вхід у систему та вихід з облікового запису. Також відображаються попередження у випадку некоректного введення даних або помилок під час авторизації. Повідомлення виводяться у верхній частині інтерфейсу та мають тимчасовий характер.

Після входу користувач отримує доступ до сторінки профілю, де відображаються його основні дані (рис. 4.9). У профілі передбачена можливість перегляду та редагування особистої інформації, зокрема імені, прізвища, електронної пошти та імені користувача. Також реалізовано можливість завантаження та зміни зображення профілю.

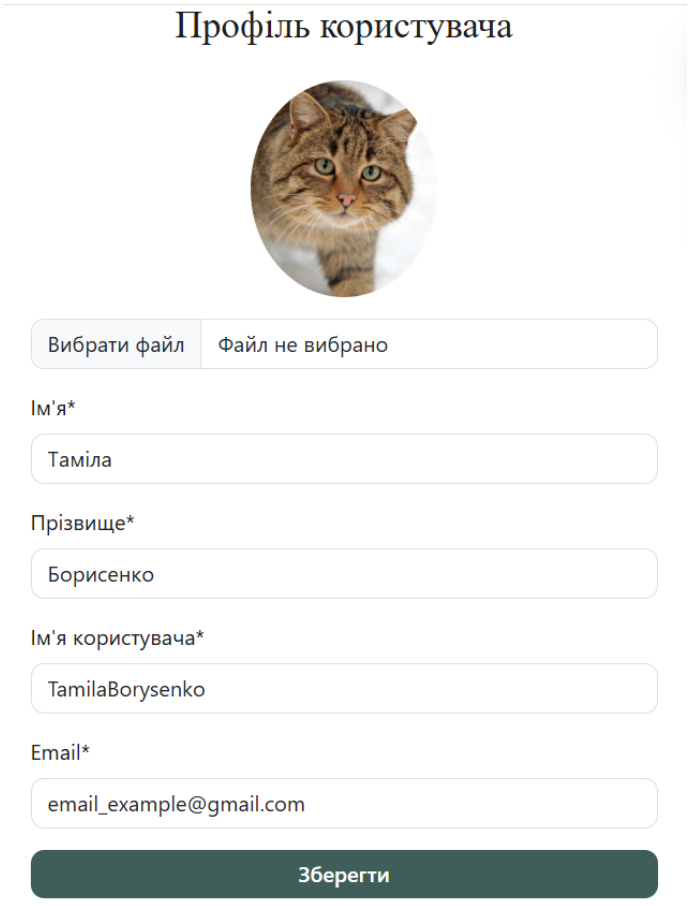


Рисунок 4.9 – Сторінка профілю користувача

4.4 Кошик, оформлення замовлення та історія покупок

Функціонал кошика є центральним елементом взаємодії користувача з товарами. Саме через нього формується замовлення та відбувається подальше оформлення покупки. Додавання товарів у кошик виконується без перезавантаження сторінки, після чого інформація про кількість товарів автоматично оновлюється в інтерфейсі.

Візуально кошик реалізовано у вигляді окремого модального вікна, яке відкривається поверх основного інтерфейсу сайту. У ньому відображається перелік доданих товарів із зазначенням кількості, ціни та загальної вартості. Загальний вигляд кошика наведено на рисунку 4.10.

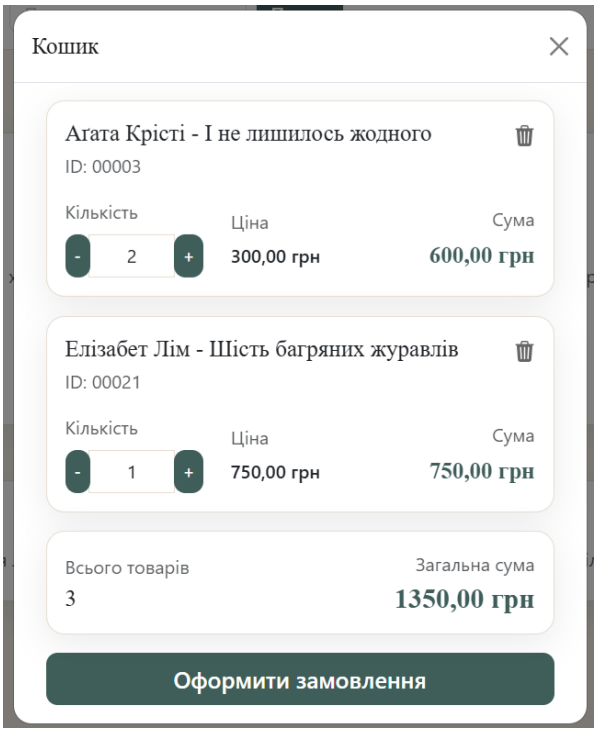


Рисунок 4.10 – Модальне вікно кошика з переліком товарів

У межах кошика реалізовано можливість зміни кількості товарів без переходу на інші сторінки. Користувач може збільшувати або зменшувати кількість позицій, при цьому відбувається автоматичне оновлення підсумкової

вартості. Також передбачено видалення товарів із кошика, після чого список одразу оновлюється відповідно до актуального стану.

Після формування кошика користувач переходить до оформлення замовлення. На цьому етапі відображається форма, у якій вводяться або підтверджуються особисті дані, а також обираються параметри доставки та оплати. Зокрема, користувач вказує контактні дані, адресу доставки (у разі необхідності) та спосіб оплати (рис. 4.11).

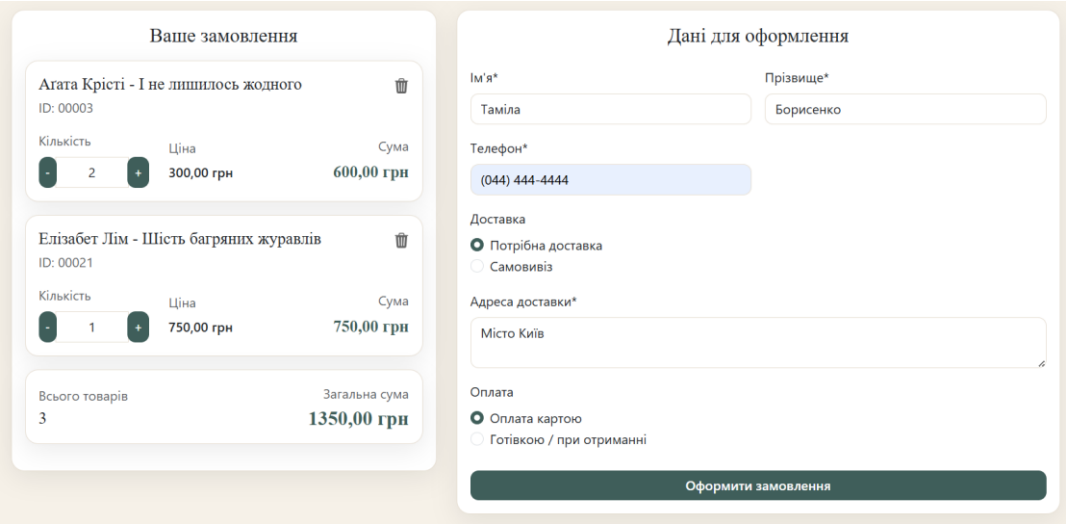


Рисунок 4.11 – Сторінка оформлення замовлення

Після підтвердження даних формується нове замовлення, яке зберігається в системі разом із переліком товарів, їх кількістю та актуальною ціною на момент покупки. Після успішного оформлення кошик очищується, а користувач перенаправляється до відповідної сторінки профілю.

У профілі користувача реалізовано історію замовлень, де відображається перелік усіх створених замовлень. Для кожного замовлення доступна детальна інформація про склад замовлення, кількість товарів, їх вартість та загальну суму. Кожне замовлення має свій статус, який відображає поточний етап обробки (наприклад, “в обробці”, “оплачено” тощо). Інтерфейс історії замовлень наведено на рисунку 4.12.

Рисунок 4.12 – Історія замовлень користувача в особистому кабінеті

4.5 Адміністративна панель

Адміністративна частина проєкту реалізована на базі стандартної Django Admin. Доступ до неї мають лише користувачі з правами адміністратора після проходження авторизації через форму входу. Адмінпанель інтегрована у систему керування даними інтернет-магазину та дозволяє працювати з основними сутностями проєкту: товарами, категоріями, замовленнями, кошиком та відгуками користувачів. Структура адмінпанелі побудована навколо моделей, що використовуються у прикладній частині застосунку, і відображає їх у вигляді таблиць із можливістю пошуку, фільтрації та редагування безпосередньо через інтерфейс (рис. 4.13).

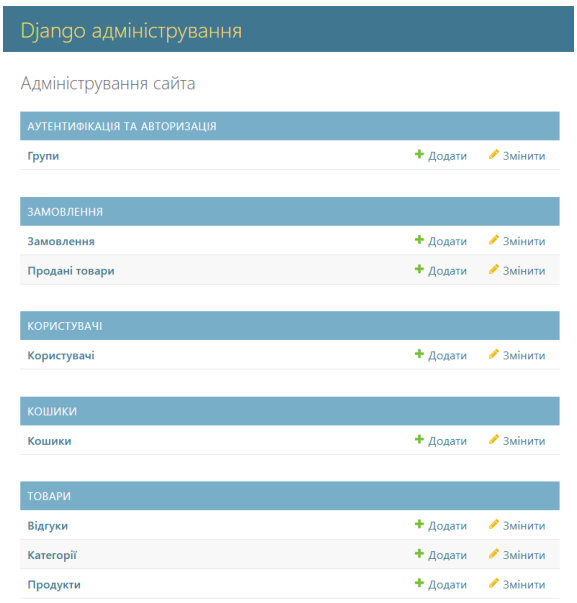


Рисунок 4.13 – Головний інтерфейс адміністративної панелі Django

У розділі керування товарами представлено список продуктів із можливістю перегляду основних параметрів, таких як назва, ціна, кількість, наявність знижки та прив’язка до категорії (рис. 4.14). Для зручності адміністрування реалізовано пошук за назвою та описом товару, а також фільтрацію за ключовими параметрами, зокрема знижкою, кількістю та категорією. Додатково налаштовано швидке редагування окремих полів без необхідності відкривати повну форму редагування, що спрощує оновлення даних у каталозі.

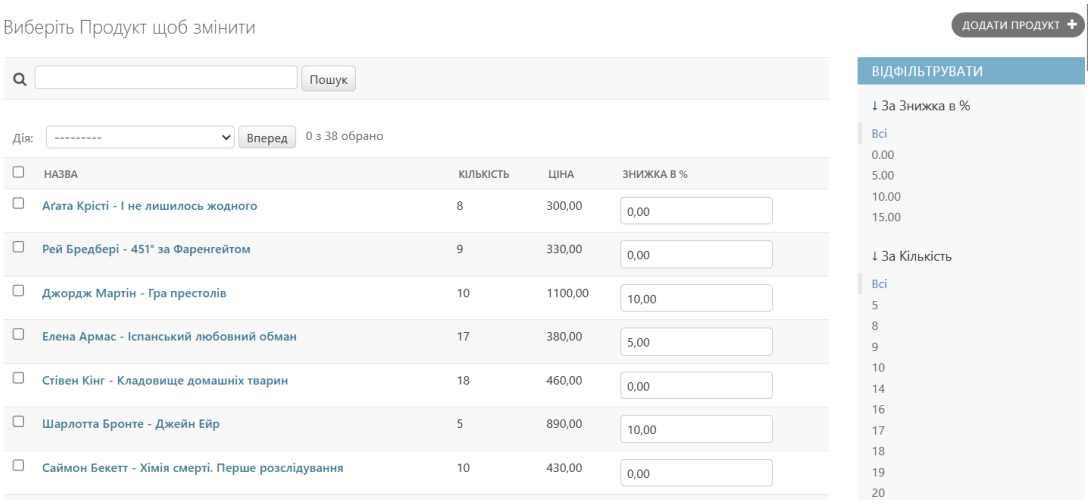


Рисунок 4.14 – Список товарів

Окремим блоком реалізовано керування категоріями товарів, де кожна категорія має унікальний ідентифікатор-slug, що автоматично формується на основі назви. Це забезпечує коректну структуру URL-адрес та впорядкування товарів у межах каталогу.

Також у системі адміністрування передбачено розділ керування відгуками користувачів, де зберігається інформація про оцінки, текстові коментарі та прив’язку до конкретного товару й користувача. Для цього розділу реалізовано можливість пошуку та фільтрації за рейтингом і датою створення, що дозволяє швидко аналізувати зворотний зв’язок від покупців.

Окремо реалізовано керування замовленнями та їхніми позиціями. У цьому блоці адміністратор може переглядати основні параметри замовлення: користувача, статус, спосіб оплати, факт оплати та дату створення. Замовлення пов’язані з позиціями товарів через вкладену структуру, що дозволяє переглядати склад замовлення без переходу між сторінками (рис. 4.15).

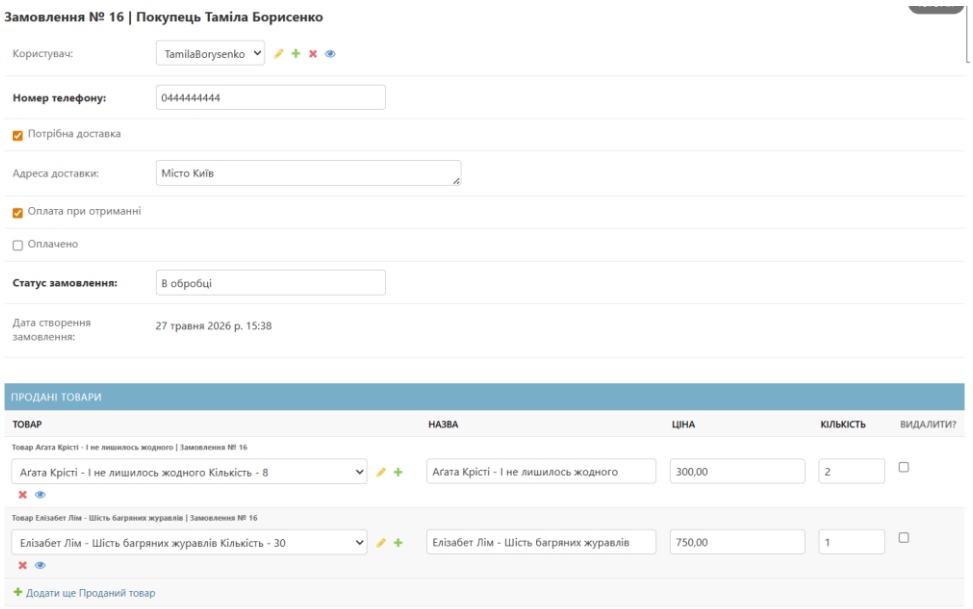


Рисунок 4.15 – Перегляд замовлення з вкладеними позиціями товарів

Також реалізовано інтеграцію даних користувачів із пов’язаними записами кошика та замовлень. Це дозволяє в межах профілю користувача в адміністративній панелі переглядати його активність, включаючи додані товари до кошика та оформлені замовлення. Такий підхід забезпечує цілісне відображення взаємодії користувача з системою та спрощує адміністрування даних.

ВИСНОВОК ДО РОЗДІЛУ

Проведений огляд підтвердив, що розроблена система забезпечує цілісний та логічно організований процес взаємодії користувача з інтернет-магазином на всіх етапах – від пошуку необхідного товару до завершення оформлення замовлення. Узгодженість навігаційних елементів, зрозуміла структура сторінок та єдині принципи побудови інтерфейсу сприяють швидкому освоєнню функціональних можливостей системи та зменшують складність роботи з нею.

Реалізовані механізми пошуку, фільтрації та категоризації товарів підвищують ефективність роботи з каталогом і дозволяють скоротити час, необхідний для знаходження потрібної продукції. Наявність детальних сторінок товарів, системи оцінювання та відгуків створює додаткові умови для прийняття обґрунтованих рішень під час вибору товарів.

Функціональність особистого кабінету, кошика та історії замовлень забезпечує персоналізацію взаємодії із системою та надає користувачам можливість контролювати власні покупки й актуальний стан замовлень. Використання асинхронної взаємодії між клієнтською та серверною частинами позитивно впливає на швидкодію інтерфейсу та підвищує комфорт використання вебзастосунку.

Адміністративна підсистема створює необхідні умови для ефективного супроводу інтернет-магазину, оскільки забезпечує централізоване керування товарами, замовленнями, категоріями та користувацьким контентом. Це дозволяє оперативно підтримувати актуальність інформації та контролювати основні бізнес-процеси.

Отже, результати тестування та аналізу інтерфейсу підтверджують, що система є повноцінно реалізованою та придатною до практичного використання. Вона забезпечує зручну взаємодію користувачів із сервісом і створює основу для подальшого розвитку функціоналу.

ВИСНОВКИ

У ході виконання роботи було спроєктовано та реалізовано клієнт-серверну систему управління товарами з рольовою моделлю доступу у форматі вебзастосунку інтернет-магазину, яка поєднує серверну та клієнтську частини в єдине функціональне середовище. Результатом стала цілісна інформаційна система, що забезпечує взаємодію користувача з каталогом товарів, кошиком, замовленнями та персональним кабінетом у межах одного логічного середовища.

Особливістю розробленого рішення є орієнтація на узгоджену роботу всіх компонентів системи, де кожен функціональний блок не існує ізольовано, а є частиною загального процесу взаємодії користувача з платформою. Це дозволило досягти природної інтеграції основних сценаріїв використання – від перегляду товарів до оформлення замовлення та подальшого супроводу даних у системі.

Важливим результатом є реалізація стабільної логіки взаємодії між клієнтською та серверною частинами, що забезпечує коректну обробку запитів, збереження стану користувача та узгодженість даних у різних частинах системи. Завдяки цьому користувацький досвід є безперервним незалежно від того, чи взаємодіє користувач із системою як гість, чи як авторизований користувач.

Окремо слід відзначити, що реалізовані підходи до побудови інтерфейсу та роботи з даними дозволили сформувати масштабовану структуру застосунку, яка не обмежується поточним функціоналом і може бути використана як основа для розширення системи в майбутньому. При цьому архітектурні рішення забезпечують достатній рівень гнучкості для подальшої адаптації під нові вимоги без необхідності кардинальної перебудови існуючої логіки.

Отже, розроблена система демонструє комплексний підхід до створення вебзастосунків, у якому поєднуються структурованість серверної логіки,

зручність клієнтського інтерфейсу та узгоджена робота всіх компонентів. Отриманий результат підтверджує ефективність використаних технологій і обраних архітектурних рішень для реалізації сучасного інтернет-магазину.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mozilla Developer Network – HTTP Overview [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата звернення: 03.02.2026).
2. Django Authentication System [Електронний ресурс] – Офіційна документація. – Режим доступу: https://docs.djangoproject.com/en/stable/topics/auth/ (дата звернення: 09.03.2026).
3. OWASP Foundation – Cross Site Request Forgery (CSRF) [Електронний ресурс] – Режим доступу: https://owasp.org/www-community/attacks/csrf (дата звернення: 11.04.2026).
4. Yakaboo [Електронний ресурс] – Офіційний сайт. – Режим доступу: https://www.yakaboo.ua/ (дата звернення: 12.02.2026).
5. Книгарня Є [Електронний ресурс] – Офіційний сайт. – Режим доступу: https://book-ye.com.ua/ (дата звернення: 12.02.2026).
6. Bookling [Електронний ресурс] – Офіційний сайт. – Режим доступу: https://bookling.ua/ (дата звернення: 12.02.2026).
7. Python Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://docs.python.org/3/ (дата звернення: 06.03.2026).
8. Django ORM Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://docs.djangoproject.com/en/stable/topics/db/models/ (дата звернення: 17.03.2026).
9. Django Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://docs.djangoproject.com/ (дата звернення: 20.03.2026).
10. Melé A. Django 4 By Example: Build powerful and reliable Python web applications from scratch. 4th ed. Birmingham : Packt Publishing, 2022. 736 p.

11. Django Admin Site Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://docs.djangoproject.com/en/stable/ref/contrib/admin/ (дата звернення: 11.04.2026).
12. PostgreSQL Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 12.04.2026).
13. MDN Web Docs – HTML [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 20.03.2026).
14. MDN Web Docs – CSS [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 20.03.2026).
15. MDN Web Docs – JavaScript [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 02.04.2026).
16. MDN Web Docs – AJAX [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (дата звернення: 04.04.2026).
17. Bootstrap Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://getbootstrap.com/docs/ (дата звернення: 04.04.2026).
18. jQuery Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://api.jquery.com/ (дата звернення: 28.04.2026).
19. PostgreSQL Full Text Search Documentation [Електронний ресурс] – Офіційна документація. – Режим доступу: https://www.postgresql.org/docs/current/textsearch.html (дата звернення: 03.05.2026).

ДОДАТОК А

Клієнт-серверна система управління товарами з рольовою моделлю доступу

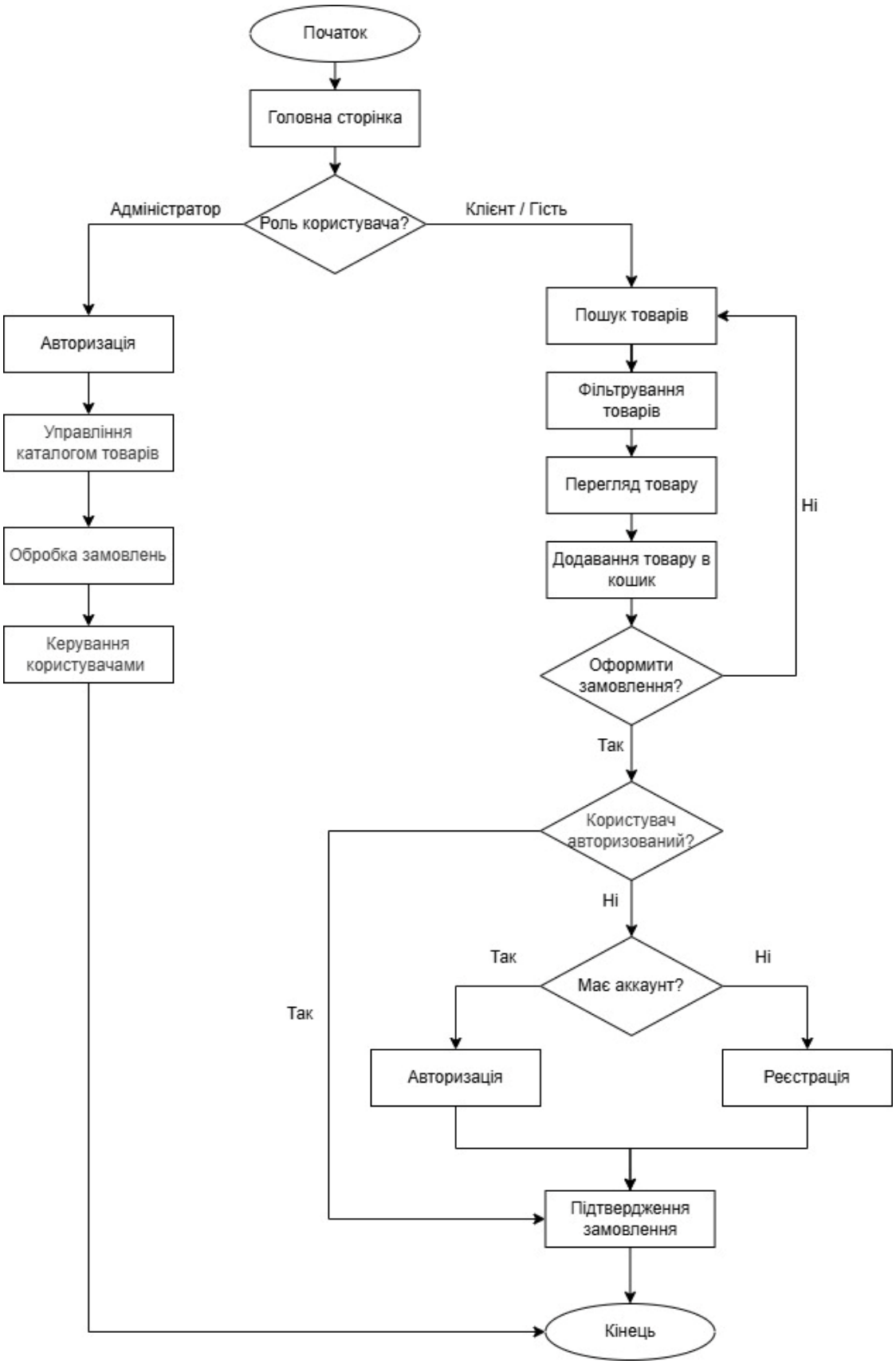
Алгоритм функціонування системи

(Принципова схема)

ІАЛЦ.467200.004 ДА

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Борисенко Т.О.			**Клієнт-серверна система управління товарами з рольовою моделлю доступу** Алгоритм функціонування системи (Принципова схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Каплунов А.В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-25
Н. Контр.		Нікольський С. С.				
Затвердив						

ДОДАТОК Б

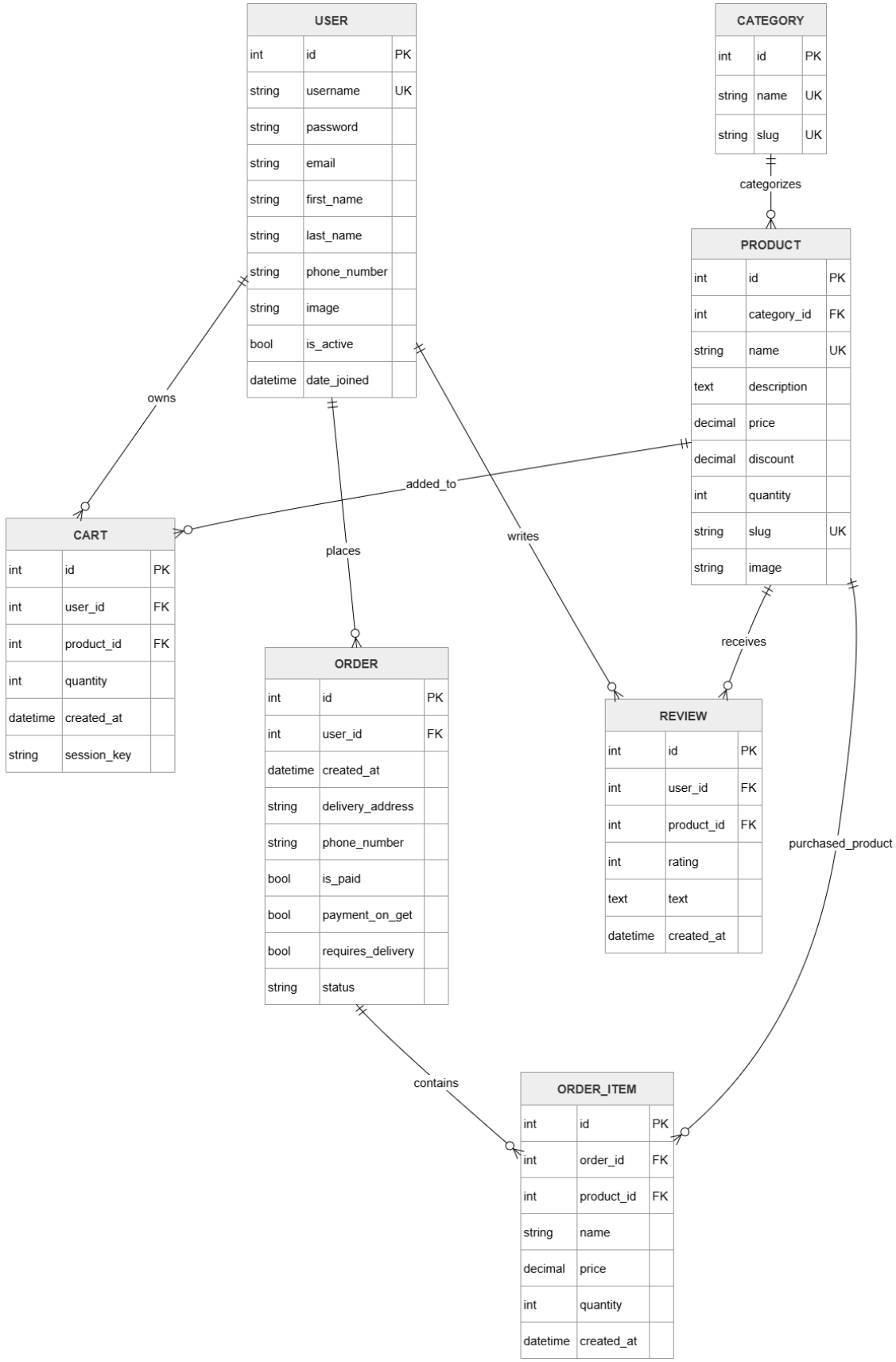
Клієнт-серверна система управління товарами з рольовою моделлю доступу

Діаграма бази даних (Функціональна схема)

ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Борисенко Т.О.			**Клієнт-серверна система управління товарами з рольовою моделлю доступу** Діаграма бази даних (Функціональна схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Каплунов А.В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-25
Н. Контр.		Нікольський С. С.				
Затвердив						

ДОДАТОК В

Клієнт-серверна система управління товарами з рольовою моделлю доступу

Діаграма варіантів використання системи

(Структурна схема)

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ 2026 р

					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Борисенко Т.О.			**Клієнт-серверна система управління товарами з рольовою моделлю доступу** Діаграма варіантів використання системи (Структурна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Каплунов А.В.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-25	
Н. Контр.		Нікольський С. С.					
Затвердив							

ДОДАТОК Г

Клієнт-серверна система управління товарами з рольовою моделлю доступу

Текст програмного коду

ІАЛЦ.467200.007 ДГ

Аркушів 14

Київ 2026 р

carts\views.py:

```
from django.http import JsonResponse
from django.template.loader import render_to_string
from django.shortcuts import get_object_or_404
from django.urls import reverse

from carts.models import BasketItem
from carts.utils import get_user_carts
from goods.models import BookItems

def cart_add(request):
    target_id = request.POST.get("product_id")
    chosen_product = get_object_or_404(BookItems, id=target_id)

    if request.user.is_authenticated:
        try:
            existing_record   =   BasketItem.objects.get(user=request.user,
product=chosen_product)
            existing_record.quantity += 1
            existing_record.save()
        except BasketItem.DoesNotExist:
            BasketItem.objects.create(user=request.user,
product=chosen_product, quantity=1)
    else:
        current_session = request.session.session_key
        try:
            existing_record                                                =
BasketItem.objects.get(guest_session_token=current_session,
product=chosen_product)
            existing_record.quantity += 1
            existing_record.save()
        except BasketItem.DoesNotExist:
            BasketItem.objects.create(guest_session_token=current_session,
product=chosen_product, quantity=1)

    updated_basket = get_user_carts(request)
    rendered_basket_blocks = render_to_string(
        "carts/includes/included_cart.html",
        {"carts": updated_basket},
        request=request
    )

    return JsonResponse({
        "message": "Продукт успішно додано",
        "cart_items_html": rendered_basket_blocks,
    })

def cart_change(request):
    record_id = request.POST.get("basket_item_id")
    target_quantity = request.POST.get("quantity")

active_item = BasketItem.objects.get(id=record_id)
    active_item.quantity = target_quantity
    active_item.save()

      current_volume = active_item.quantity
```

```
    updated_basket = get_user_carts(request)
    template_data = {"carts": updated_basket}

    previous_page = request.META.get('HTTP_REFERER', '')
    if reverse('orders:create_order') in previous_page:
        template_data["order"] = True

    rendered_basket_blocks = render_to_string(
        "carts/includes/included_cart.html",
        template_data,
        request=request
    )

    return JsonResponse({
        "message": "Обсяг оновлено",
        "cart_items_html": rendered_basket_blocks,
        "quantity": current_volume,
    })


def cart_remove(request):
    record_id = request.POST.get("basket_item_id")

    active_item = BasketItem.objects.get(id=record_id)
    removed_quantity = active_item.quantity
    active_item.delete()

    updated_basket = get_user_carts(request)
    template_data = {"carts": updated_basket}

    previous_page = request.META.get('HTTP_REFERER', '')
    if reverse('orders:create_order') in previous_page:
        template_data["order"] = True

    rendered_basket_blocks = render_to_string(
        "carts/includes/included_cart.html",
        template_data,
        request=request
    )

    return JsonResponse({
        "message": "Позицію видалено",
        "cart_items_html": rendered_basket_blocks,
        "quantity_deleted": removed_quantity,
    })
```

carts\models.py:

```
from django.db import models
from goods.models import BookItems
from users.models import User

class UserBasketQuerySet(models.QuerySet):

    def total_quantity(self):
        if not self:
            return 0
        return sum(item.quantity for item in self)

    def total_price(self):
        return sum(item.calculate_holding_cost() for item in self)
```

```
    class BasketItem(models.Model):
        product = models.ForeignKey(on_delete=models.CASCADE, to=BookItems,
verbose_name='Товар')
        user = models.ForeignKey(on_delete=models.CASCADE, to=User, null=True,
blank=True, verbose_name='Користувач')
        quantity = models.PositiveSmallIntegerField(verbose_name='Кількість',
default=0)
        guest_session_token = models.CharField(blank=True, null=True,
max_length=32)
        added_at = models.DateTimeField(verbose_name='Дата додавання',
auto_now_add=True)

        objects = UserBasketQuerySet().as_manager()

        class Meta:
            db_table = 'user_basket'
            ordering = ("id",)
            verbose_name = "Елемент кошика"
            verbose_name_plural = "Елементи кошиків"

        def calculate_holding_cost(self):
            return round(self.product.sell_price() * self.quantity, 2)

        def __str__(self):
            if not self.user:
                return f'Гість | {self.product.name} | К-сть: {self.quantity}'
            return f'Кошик ({self.user.username}) | {self.product.name} | К-
сть: {self.quantity}'
```

goods\models.py:

```
    from django.db import models
    from django.urls import reverse
    from django.db.models import Avg
    from django.core.validators import MinValueValidator, MaxValueValidator
    from users.models import User

    class ItemGroup(models.Model):
        name = models.CharField(max_length=160, unique=True,
verbose_name='Назва категорії')
        slug = models.SlugField(max_length=210, blank=True, unique=True,
null=True, verbose_name='Слаг URL')

        class Meta:
            db_table = 'item_groups'
            ordering = ("id",)
            verbose_name = 'Категорія'
            verbose_name_plural = 'Категорії товарів'

        def __str__(self):
            return self.name


    class BookItems(models.Model):
        name = models.CharField(unique=True, max_length=160,
verbose_name='Назва книги')
        slug = models.SlugField(max_length=220, null=True, unique=True,
blank=True, verbose_name='Слаг книги')
```

```
        category = models.ForeignKey(on_delete=models.CASCADE, to=ItemGroup,
verbose_name='Категорія')
        description = models.TextField(null=True, blank=True,
verbose_name='Опис видання')
        image = models.ImageField(upload_to='goods_images', blank=True,
null=True, verbose_name='Обкладинка')
        price = models.DecimalField(default=0.00, decimal_places=2,
max_digits=8, verbose_name='Базова вартість')
        sale_percent = models.DecimalField(default=0.00, decimal_places=2,
max_digits=5, verbose_name='Знижка (%)')
        quantity = models.PositiveIntegerField(default=0, verbose_name='Залишок
на складі')

        class Meta:
            db_table = 'book_products'
            ordering = ("id",)
            verbose_name = 'Книжкова позиція'
            verbose_name_plural = 'Книжкові позиції'

        def get_average_rating(self):
            return
self.customer_feedbacks.aggregate(average_score=Avg('rating'))['average_score']

        def sell_price(self):
            if not self.sale_percent:
                return self.price
            discount_amount = (self.price * self.sale_percent) / 100
            return round(self.price - discount_amount, 2)

        def display_id(self):
            return f"{self.id:05d}"

        def get_absolute_url(self):
            return reverse("goods:product", kwargs={"product_slug": self.slug})

        def __str__(self):
            return f'{self.name} [Залишок: {self.quantity}]'


    class Feedback(models.Model):
        product = models.ForeignKey(
            BookItems,
            on_delete=models.CASCADE,
            related_name='customer_feedbacks',
            verbose_name='Товар'
        )
        user = models.ForeignKey(
            User,
            on_delete=models.CASCADE,
            verbose_name='Автор відгуку'
        )
        text = models.TextField(verbose_name='Текст рецензії')
        rating = models.PositiveSmallIntegerField(
            validators=[MinValueValidator(1), MaxValueValidator(5)],
            verbose_name='Оцінка'
        )
        published_date = models.DateTimeField(auto_now_add=True)

        class Meta:
            db_table = 'product_feedbacks'
            verbose_name = 'Рецензія'
            verbose_name_plural = 'Рецензії та відгуки'

        def __str__(self):
```

```
            return f"Відгук {self.user.username} на {self.product.name}"
```

goods\views.py:

```
from django.core.paginator import Paginator
from django.shortcuts import render, get_object_or_404, redirect
from django.http import Http404
from django.contrib.auth.decorators import login_required

from goods.models import BookItems, Feedback
from goods.utils import q_search
from .forms import ReviewForm

def catalog(request, category_slug=None):
    requested_page = request.GET.get('page', 1)
    filter_sale = request.GET.get('on_sale', None)
    sorting_rule = request.GET.get('order_by', None)
    search_query = request.GET.get('q', None)

    if search_query:
        catalog_items = q_search(search_query)
    elif category_slug == 'all':
        catalog_items = BookItems.objects.all()
    else:
        catalog_items                                                    =
BookItems.objects.filter(category__slug=category_slug)
        if catalog_items.count() == 0:
            raise Http404()

    if filter_sale:
        catalog_items = catalog_items.filter(sale_percent__gt=0)

    if sorting_rule and sorting_rule != "default":
        catalog_items = catalog_items.order_by(sorting_rule)

    paginator_instance = Paginator(catalog_items, 8)
    paginated_page = paginator_instance.page(int(requested_page))

    response_context = {
        "title": "Bookstore - Каталог товарів",
        "goods": paginated_page,
        "slug_url": category_slug
    }
    return render(request, "goods/catalog.html", response_context)


def product(request, product_slug):
    selected_book = BookItems.objects.get(slug=product_slug)

    return render(request, "goods/product.html", {
        'product': selected_book
    })


@login_required
def add_review(request, product_id):
    target_book = get_object_or_404(BookItems, id=product_id)

    if request.method == "POST":
        review_form = ReviewForm(request.POST)
        if review_form.is_valid():
            new_comment = review_form.save(commit=False)
            new_comment.product = target_book
```

```
                new_comment.user = request.user
                new_comment.save()

        return redirect('goods:product', product_slug=target_book.slug)


    @login_required
    def delete_review(request, id):
        user_feedback = get_object_or_404(Feedback, id=id)

        if user_feedback.user == request.user:
            user_feedback.delete()

        return redirect(request.META.get('HTTP_REFERER', '/'))
```

main\views.py:

```
    from django.shortcuts import render

    def contacts(page_request):
        page_data = {
            'title': 'Bookstore - Контакти',
            'content': 'Зворотний зв’язок',
            'text_on_page': 'Контактна інформація для зв’язку з адміністрацією
магазину.'
        }
        return render(page_request, 'main/about.html', context=page_data)


    def delivery_information(page_request):
        page_data = {
            'title': 'Bookstore - Сервіс доставки',
            'content': 'Умови отримання та оплати',
            'text_on_page': 'Детальні правила доставки замовлень та варіанти
проведення розрахунків.'
        }
        return render(page_request, 'main/about.html', context=page_data)


    def index(page_request):
        page_data = {
            'title': 'Bookstore - Онлайн-платформа',
            'content': 'Книжковий всесвіт Bookstore',
        }
        return render(page_request, 'main/index.html', context=page_data)


    def about(page_request):
        description_text = (
            "Раді вітати вас у Bookstore – вашому надійному провіднику у
неосяжному світі літератури. "
            "Наш проект створений спеціально для поціновувачів друкованого
слова, які шукають захопливі "
            "сюжети та глибокі сенси, здатні перевернути світогляд. Головна мета
нашої команди – забезпечити "
            "кожному клієнту приємні миті натхнення, нових знань та затишку за
читанням. Ми переконані, що "
            "гарне видання – це не просто паперові сторінки. Це справжні двері
до нових відкриттів, унікальна "
            "можливість подорожувати крізь епохи та вірний супутник у
повсякденному житті. Завітайте до каталогу "
```

```
            "Bookstore, де кожна книга відкриває нові обрії, а кожна прочитана
глава наближає до чогось "
            "незвичайного. Ставайте частиною нашої великої спільноти читачів, і
ми з радістю допоможемо "
            "вам обрати вашу наступну незабутню літературну мандрівку!"
        )

        page_data = {
            'title': 'Bookstore - Історія компанії',
            'content': 'Наші цінності',
            'text_on_page': description_text
        }
        return render(page_request, 'main/about.html', context=page_data)
```

orders\models.py:

```
from django.db import models
from goods.models import BookItems
from users.models import User

class TransactionDetailsQuerySet(models.QuerySet):

    def total_quantity(self):
        if not self:
            return 0
        return sum(single_item.quantity for single_item in self)

    def total_price(self):
        return sum(single_item.products_price() for single_item in self)


class Purchase(models.Model):
    user    =    models.ForeignKey(on_delete=models.SET_DEFAULT,    to=User,
null=True, blank=True, verbose_name="Клієнт", default=None)
    registered_at         =         models.DateTimeField(auto_now_add=True,
verbose_name="Час реєстрації замовлення")
    phone_number = models.CharField(max_length=20, verbose_name="Контактний
телефон")
    requires_delivery          =          models.BooleanField(default=False,
verbose_name="Оформлення доставки")
    payment_on_get           =           models.BooleanField(default=False,
verbose_name="Розрахунок при отриманні")
    delivery_address     =     models.TextField(blank=True,     null=True,
verbose_name="Точна адреса")
    is_paid   =   models.BooleanField(default=False,   verbose_name="Статус
оплати")
    status = models.CharField(max_length=60, default='Прийнято на обробку',
verbose_name="Стан виконання")

    class Meta:
        db_table = "client_purchases"
        ordering = ("id",)
        verbose_name = "Замовлення"
        verbose_name_plural = "Замовлення користувачів"

    def __str__(self):
        return  f"Покупка  №  {self.pk}  |  Клієнт:  {self.user.first_name}
{self.user.last_name}"


class TransactionDetails(models.Model):
```

```
        order   =   models.ForeignKey(to=Purchase,   on_delete=models.CASCADE,
verbose_name="Картка замовлення")
        name = models.CharField(max_length=170, verbose_name="Найменування на
момент купівлі")
        product = models.ForeignKey(to=BookItems, on_delete=models.SET_DEFAULT,
null=True, verbose_name="Товарна позиція", default=None)
        price     =     models.DecimalField(max_digits=8,     decimal_places=2,
verbose_name="Фіксована ціна")
        quantity           =           models.PositiveIntegerField(default=0,
verbose_name="Кількість одиниць")
        registered_at        =        models.DateTimeField(auto_now_add=True,
verbose_name="Дата фіксації")

        objects = TransactionDetailsQuerySet.as_manager()

        class Meta:
            db_table = "purchase_items_details"
            ordering = ("id",)
            verbose_name = "Придбана книга"
            verbose_name_plural = "Придбані книги"

        def products_price(self):
            return round(self.price * self.quantity, 2)

        def __str__(self):
            return f"Елемент: {self.name} | До покупки № {self.order.pk}"
```

orders\views.py:

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.db import transaction
from django.forms import ValidationError

from carts.models import BasketItem
from orders.forms import CheckoutForm
from orders.models import Purchase, TransactionDetails

@login_required
def create_order(request):
    if request.method == 'POST':
        checkout_form = CheckoutForm(data=request.POST)

        if checkout_form.is_valid():
            try:
                with transaction.atomic():
                    current_user = request.user
                    active_basket_items                                    =
BasketItem.objects.filter(user=current_user)

                    if active_basket_items.count() > 0:
                        new_purchase = Purchase.objects.create(
                            user=current_user,

phone_number=checkout_form.cleaned_data['phone_number'],

delivery_address=checkout_form.cleaned_data['delivery_address'],

requires_delivery=checkout_form.cleaned_data['requires_delivery'],

payment_on_get=checkout_form.cleaned_data['payment_on_get'],
                            )
```

```
                            for element in active_basket_items:
                                target_book = element.product
                                book_title = element.product.name
                                fixed_price = element.product.sell_price()
                                requested_qty = element.quantity

                                if target_book.quantity < requested_qty:
                                    raise ValidationError(
                                        f'На складі не вистачає позиції:
{book_title}. '
                                        f'Доступно для замовлення:
{target_book.quantity}'
                                    )

                                TransactionDetails.objects.create(
                                    order=new_purchase,
                                    product=target_book,
                                    name=book_title,
                                    price=fixed_price,
                                    quantity=requested_qty,
                                )

                                target_book.quantity -= requested_qty
                                target_book.save()

                            active_basket_items.delete()
                            messages.success(request, 'Ваше замовлення успішно
сформовано та прийнято!')
                            return redirect('users:profile')

                    except ValidationError as error:
                        messages.success(request, str(error))
                        return redirect('orders:create_order')
            else:
                user_info = {
                    'first_name': request.user.first_name,
                    'last_name': request.user.last_name,
                }
                checkout_form = CheckoutForm(initial=user_info)

            render_context = {
                'title': 'Bookstore - Оформлення покупки',
                'form': checkout_form,
                'order': True,
            }
            return render(request, 'orders/create_order.html',
context=render_context)
```

users\views.py:

```
from django.contrib.auth.decorators import login_required
from django.db.models import Prefetch
from django.contrib import auth, messages
from django.http import HttpResponseRedirect
from django.urls import reverse
from django.shortcuts import redirect, render

from carts.models import BasketItem
from orders.models import Purchase, TransactionDetails
```

```
    from users.forms import UserProfileEditForm, AccountSignInForm,
AccountSignUpForm


    def registration(request):
        if request.method == 'POST':
            signup_form = AccountSignUpForm(data=request.POST)
            if signup_form.is_valid():
                signup_form.save()

                cached_session_key = request.session.session_key
                new_user = signup_form.instance
                auth.login(request, new_user)

                if cached_session_key:

BasketItem.objects.filter(guest_session_token=cached_session_key).update(user=ne
w_user)

                messages.success(request, f"{new_user.username}, реєстрація
пройшла успішно. Ласкаво просимо!")
                return HttpResponseRedirect(reverse('main:index'))
        else:
            signup_form = AccountSignUpForm()

        render_context = {
            'title': 'Bookstore - Створення акаунту',
            'form': signup_form
        }
        return render(request, 'users/registration.html', render_context)


    def login(request):
        if request.method == 'POST':
            signin_form = AccountSignInForm(data=request.POST)
            if signin_form.is_valid():
                auth_username = signin_form.cleaned_data.get('username')
                auth_password = signin_form.cleaned_data.get('password')

                authenticated_user = auth.authenticate(username=auth_username,
password=auth_password)
                cached_session_key = request.session.session_key

                if authenticated_user:
                    auth.login(request, authenticated_user)
                    messages.success(request, f"{auth_username}, ви успішно
авторизувалися!")

                    if cached_session_key:

BasketItem.objects.filter(user=authenticated_user).delete()

BasketItem.objects.filter(guest_session_token=cached_session_key).update(user=au
thenticated_user)

                    target_next_page = request.POST.get('next', None)
                    if target_next_page and target_next_page !=
reverse('users:logout'):
                        return HttpResponseRedirect(target_next_page)

                    return HttpResponseRedirect(reverse('main:index'))
        else:
            signin_form = AccountSignInForm()
```

```
        render_context = {
            'title': 'Bookstore - Вхід до кабінету',
            'form': signin_form
        }
        return render(request, 'users/login.html', render_context)


    @login_required
    def profile(request):
        if request.method == 'POST':
            profile_form = UserProfileEditForm(data=request.POST,
instance=request.user, files=request.FILES)
            if profile_form.is_valid():
                profile_form.save()
                messages.success(request, "Дані вашого профілю успішно
оновлено")
                return HttpResponseRedirect(reverse('users:profile'))
        else:
            profile_form = UserProfileEditForm(instance=request.user)

        user_purchases =
Purchase.objects.filter(user=request.user).prefetch_related(
            Prefetch(
                "transactiondetails_set",
                queryset=TransactionDetails.objects.select_related("product"),
            )
        ).order_by("-id")

        render_context = {
            'title': 'Bookstore - Профіль користувача',
            'form': profile_form,
            'orders': user_purchases,
        }
        return render(request, 'users/profile.html', render_context)


    def users_cart(request):
        return render(request, 'users/users_cart.html')


    @login_required
    def logout(request):
        username_placeholder = request.user.username
        auth.logout(request)
        messages.success(request, f"{username_placeholder}, ви вийшли з
системи.")
        return redirect(reverse('main:index'))
```

jquery-ajax.js:

```
    $(document).ready(function () {
        var popupAlert = $("#jq-notification");

        function triggerAlertFade(element, messageText) {
            element.html(messageText).fadeIn(500);
            setTimeout(function () {
                element.fadeOut(500);
            }, 8000);
        }

        $(document).on("click", ".increment", function () {
            var currentQtyInput = $(this).closest('.input-
group').find('.number');
```

```
            var targetUrl = $(this).data("cart-change-url");
            var recordID = $(this).data("cart-id");
            var parsedValue = parseInt(currentQtyInput.val()) || 0;

            currentQtyInput.val(parsedValue + 1);
            syncBasketQuantity(recordID, parsedValue + 1, 1, targetUrl);
        });

        $(document).on("click", ".decrement", function () {
            var currentQtyInput = $(this).closest('.input-
group').find('.number');
            var targetUrl = $(this).data("cart-change-url");
            var recordID = $(this).data("cart-id");
            var parsedValue = parseInt(currentQtyInput.val()) || 0;

            if (parsedValue > 1) {
                currentQtyInput.val(parsedValue - 1);
                syncBasketQuantity(recordID, parsedValue - 1, -1, targetUrl);
            }
        });

        function syncBasketQuantity(basketItemID, totalQty, direction,
endpointUrl) {
            $.ajax({
                type: "POST",
                url: endpointUrl,
                data: {
                    basket_item_id: basketItemID,
                    quantity: totalQty,
                    csrfmiddlewaretoken:
$("[name=csrfmiddlewaretoken]").val(),
                },
                success: function (response) {
                    triggerAlertFade(popupAlert, response.message);

                    var basketBadge = $("#goods-in-cart-count");
                    var dynamicCount = parseInt(basketBadge.text() || 0);
                    dynamicCount += direction;
                    basketBadge.text(dynamicCount);

                    $("#cart-items-container").html(response.cart_items_html);
                },
                error: function (error) {
                    console.error("Помилка під час синхронізації кількості у
кошику");
                },
            });
        }

        $(document).on("click", ".add-to-cart", function (event) {
            event.preventDefault();

            var basketBadge = $("#goods-in-cart-count");
            var totalCount = parseInt(basketBadge.text() || 0);
            var targetItemId = $(this).data("product-id");
            var postRequestUrl = $(this).attr("href");

            $.ajax({
                type: "POST",
                url: postRequestUrl,
                data: {
                    product_id: targetItemId,
                    csrfmiddlewaretoken:
$("[name=csrfmiddlewaretoken]").val(),
```

```
                },
                success: function (response) {
                    triggerAlertFade(popupAlert, response.message);

                    totalCount++;
                    basketBadge.text(totalCount);

                    $("#cart-items-container").html(response.cart_items_html);
                },
                error: function (xhr) {
                    console.error("Неможливо додати товар до кошика");
                },
            });
        });

        $(document).on("click", ".remove-from-cart", function (event) {
            event.preventDefault();

            var basketBadge = $("#goods-in-cart-count");
            var currentTotal = parseInt(basketBadge.text() || 0);
            var targetRecordId = $(this).data("cart-id");
            var executionUrl = $(this).attr("href");

            $.ajax({
                type: "POST",
                url: executionUrl,
                data: {
                    basket_item_id: targetRecordId,
                    csrfmiddlewaretoken:
$("[name=csrfmiddlewaretoken]").val(),
                },
                success: function (response) {
                    triggerAlertFade(popupAlert, response.message);

                    currentTotal -= response.quantity_deleted;
                    basketBadge.text(currentTotal);

                    $("#cart-items-container").html(response.cart_items_html);
                },
                error: function (err) {
                    console.error("Невдала спроба видалення об'єкта з кошика");
                },
            });
        });

        var coreNotification = $('#notification');
        if (coreNotification.length > 0) {
            setTimeout(function () {
                coreNotification.alert('close');
            }, 8000);
        }

        $('#modalButton').on('click', function () {
            var baseModal = $('#exampleModal');
            baseModal.appendTo('body').modal('show');
        });

        $('#exampleModal .btn-close').on('click', function () {
            $('#exampleModal').modal('hide');
        });

        $("input[name='requires_delivery']").on('change', function () {
            if ($(this).val() === "1") {
                $("#deliveryAddressField").show();
```

```
            } else {
                $("#deliveryAddressField").hide();
            }
        });

        var phoneInputField = document.getElementById('id_phone_number');
        if (phoneInputField) {
            phoneInputField.addEventListener('input', function (event) {
                var digitsOnly = event.target.value.replace(/\D/g, '');
                var                        matchingGroups                        =
digitsOnly.match(/^(\d{0,3})(\d{0,3})(\d{0,4})$/);

                if (matchingGroups) {
                    event.target.value = !matchingGroups[2]
                        ? matchingGroups[1]
                        : '(' + matchingGroups[1] + ') ' + matchingGroups[2] +
(matchingGroups[3] ? '-' + matchingGroups[3] : '');
                }
            });
        }

        $('#checkout-submission-form').on('submit', function (event) {
            var rawPhoneValue = $('#id_phone_number').val();
            var validationRegex = /^\(\d{3}\) \d{3}-\d{4}$/;

            if (!validationRegex.test(rawPhoneValue)) {
                $('#phone_number_error').show();
                event.preventDefault();
            } else {
                $('#phone_number_error').hide();
                var  numbersOnlyString  =  rawPhoneValue.replace(/[\(\)\-\s]/g,
'');
                $('#id_phone_number').val(numbersOnlyString);
            }
        });
    });
```