МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка веб-застосунку з продажу комп'ютерів**

Виконав: студент 4 курсу, групи 6.1212-ПЗС
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
В.С. Іваненко
(ініціали та прізвище)
Керівник доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Діскус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2026 року

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

Іваненко Вікторії Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка веб-застосунку з продажу комп’ютерів
керівник роботи Безверхий Анатолій Ігорович, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ЗНУ від 23.12.2025 р. № 1830-с
2. Строк подання студентом кваліфікаційної роботи 20.05.2026
3. Вихідні дані бакалаврської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - огляд та аналіз існуючих рішень та аналогів;
   - дослідження проблеми створення Інтернет магазину з продажу комп’ютерів
   - створення програмного продукту та його опис;
   - перелік вимог для роботи програми;
   - розробка висновків та пропозицій.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) 18 слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1.	Аналіз предметної області.	20.03-24.03	виконано
2.	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником.	25.03	виконано
3.	Аналіз програм-аналогів.	26.03-29.04	виконано
4.	Вибір технологій для реалізації.	01.04-02.04	виконано
5.	Аналіз існуючих рішень і технологій.	02.04-25.04	виконано
6.	Проектування веб-магазину з продажу комп'ютерів	02.04-05.04	виконано
7.	Побудова архітектури застосунку.	06.04-08.04	виконано
8.	Розробка бази даних застосунку	08.04-16.05	виконано
9.	Реалізація системи продажу і розрахунків клієнтів	10.04-11.04	виконано
10.	Розробка користувацького інтерфейсу.	12.04-13.04	виконано
11.	Розробка модулів контролю продажів	08.04-16.05	виконано
12.	Розширення функціоналу застосунку	14.05-16.05	виконано
13.	Рефакторинг коду.	16.05-22.05	виконано
14.	Системне тестування імпорту/візуалізації/експорту	23.05-28.05	виконано
15.	Оформлення пояснювальної записки.	29.05-06.05	виконано
16.	Оформлення презентації і отримання рецензії.	07.05-13.06	в процесі

Студентка _____ _____В.С. Іваненко

Керівник роботи _____ _____А. І. Безверхий

Нормоконтроль пройдено

Нормоконтролер _____ _____ _І.А. Скрипник

АНОТАЦІЯ

Сторінок — 73,

Рисунків — 16,

Джерел — 16.

Іваненко В.С. Розробка веб-застосунку з продажу комп’ютерів кваліфікаційна робота бакалавра спеціальності 121 Інженерія програмного забезпечення / наук. керівник А.І. Безверхий Запоріжжя : ЗНУ, 2026. 73 с.

У сучасних умовах цифровізації суспільства електронна комерція займає провідне місце у сфері торгівлі. Інтернет-магазини дозволяють автоматизувати процес продажу товарів, забезпечити доступність сервісів для користувачів та оптимізувати бізнес-процеси. Електронна комерція потребує надійних, масштабованих і безпечних веб-застосунків. Інтернет-магазини комп’ютерної техніки повинні забезпечувати зручний каталог товарів, управління замовленнями, контроль складських залишків та інтеграцію платіжних систем.

Метою роботи є розробка веб-застосунку інтернет-магазину комп’ютерів із використанням фреймворку Django та інтеграцією платіжної системи. Основні завдання роботи — спроєктувати структуру веб-застосунку, реалізувати функціонал магазину, інтегрувати платіжну систему

Для цього проведено аналіз найпоширеніших інтернет-магазинв з продажу комп’ютерів і розроблено власний веб-застосунок, проведена апробація на реальних даних. Реалізовано веб-застосунок зі зручним клієнтським інтерфейсом та інтерфейсом адміністратора. Перевірена працездатність застосунку на реальних даних.

Ключові слова: *Web application, E-commerce, Product reserve, Django, LiqPay.*

ABSTRACT

Pages — 73,
Figures — 16,
References — 16.

Ivanenko V.S. Development of a web application for selling computers: bachelor's qualification thesis, specialty 121 Software Engineering / scientific supervisor A.I. Bezverkhyi. Zaporizhzhia: ZNU, 2026. 73 p.

In the modern conditions of society's digitalization, e-commerce takes a leading place in the trade sector. Online stores allow automating the process of selling goods, ensuring the availability of services for users, and optimizing business processes. E-commerce requires reliable, scalable, and secure web applications. Computer hardware online stores must provide a convenient product catalog, order management, inventory control, and payment system integration.

The aim of the work is to develop a computer online store web application using the Django framework and integrating a payment system. The main tasks of the work are to design the structure of the web application, implement the store's functionality, and integrate a payment system.

For this purpose, an analysis of the most popular computer online stores was conducted, a custom web application was developed, and testing on real data was performed. A web application with a user-friendly client interface and an administrator interface was implemented. The application's performance was verified on real data.

Keywords: Web application, E-commerce, Product reserve, Django, LiqPay.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ СУЧАСНИХ АНАЛОГІВ І ТЕХНОЛОГІЙ 13
1.1 Огляд сучасних інтернет-магазинів з продажу компютерної технки 13
1.2 Ключові гравці на ринку України [ ] 13
1.2.1 Універсальні маркетплейси (Rozetka, Moyo, Comfy) 13
1.2.2 Спеціалізовані магазини комп'ютерної техніки 15
1.3 Світові лідери (США та Європа) [ ] 17
1.4 Порівняльний аналіз функціоналу магазинів 20
1.5 Сучасні тренди електронної комерції в IT-сегменті 21
1.6 Огляд засобів та технологій розробок інтернет-магазинів комп'ютерної техніки 21
1.6.1 Готові CMS та SaaS-платформи 22
1.6.2 Фреймворки для Custom-розробки 22
1.7 Висновки з розділу 1 24
2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ З ПРОДАЖУ КОМП'ЮТЕРНОЇ ТЕХНІКИ 26
2.1 Розробка архітектури програмної системи 26
2.1.1 Загальна архітектурна модель MVT 26
2.1.2 Основні компоненти системи та їх взаємодія 26
2.1.4 Архітектура механізму резервування товарів 29
2.1.5 Архітектура платіжної інтеграції (LiqPay Flow) 29
2.1.6 Безпека архітектури 30
2.2 Проєктування та розробка користувацького інтерфейсу веб-застосунку 30
2.3 Висновки з розділу 2 33

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ З ПРОДАЖУ КОМП'ЮТЕРНОЇ ТЕХНІКИ 34
3.1 Створення проєкту веб-сайту в фреймворку Django 34
3.2 Створення моделей для ORM системи в Django 36
3.3 Створення логіки безпечного резервування товарів 45
3.4 Створення інтерфейсів користувачів (frontend) 53
3.4.1 Інтерфейс адміністратора 53
3.4.2 Інтерфейси користувачів 58
3.5 Підключення платіжної системи LiqPay 68
3.6 Висновки з розділу 3 70
ВИСНОВКИ 71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72

ВСТУП

Актуальність теми

У сучасних умовах цифровізації суспільства електронна комерція займає провідне місце у сфері торгівлі. Інтернет-магазини дозволяють автоматизувати процес продажу товарів, забезпечити доступність сервісів для користувачів та оптимізувати бізнес-процеси [12]. Сучасна електронна комерція потребує надійних, масштабованих і безпечних веб-застосунків. Інтернет-магазини комп'ютерної техніки повинні забезпечувати зручний каталог товарів, управління замовленнями, контроль складських залишків та інтеграцію платіжних систем.

Актуальність теми зумовлена стрімким розвитком електронної комерції та зростанням попиту на онлайн-продаж комп'ютерної техніки. Сучасні користувачі очікують швидкого, безпечного та зручного процесу придбання товарів через Інтернет. Особливо актуальним є створення веб-застосунків для продажу комп'ютерної техніки, що характеризується широким асортиментом і необхідністю точного контролю складських залишків. Особливої уваги потребує забезпечення коректної обробки замовлень у реальному часі, оскільки одночасні дії користувачів можуть призводити до логічних помилок, зокрема продажу товарів, яких фактично немає на складі.

Таким чином, розробка веб-застосунку з механізмами контролю складських залишків, резервування товарів та інтеграції платіжних систем є актуальним і практично значущим завданням

Мета дослідження

Метою роботи є розробка веб-застосунку інтернет-магазину комп'ютерів із використанням фреймворку Django та інтеграцією платіжної системи.

Завдання дослідження

Для досягнення мети поставлено такі завдання:

- проаналізувати предметну область;
- обрати технології розробки;
- спроєктувати структуру системи;
- реалізувати функціонал магазину;
- інтегрувати платіжну систему;
- провести апробацію застосунку.

Об’єкт дослідження

Об’єктом дослідження є розробка Веб-застосунку з продажу комп’ютерів

Предмет дослідження

Предметом дослідження є методи та інструменти для реалізації веб-застосунку та інтеграції платіжної системи

Методи дослідження

Аналіз літературних джерел та огляд існуючих веб-рішень у сфері веб-розробок сайтів з продажу комп’ютерної техніки. Вивчення методів та інструментів розробки веб-сайтів із використанням Django-фреймворку, а також аналіз сучасних тенденцій у розробці Інтернет - магазинів.

Практичне значення одержаних результатів

Практичне значення розробленого веб-застосунку у створенні готового до впровадження програмного продукту — сучасного інтернет-магазину комп’ютерної техніки, який відповідає актуальним вимогам електронної комерції та потребам українського бізнесу.

Результати роботи мають як бізнес-орієнтовану, так і технічну цінність:

1. Автоматизація продажів: Розроблена система дозволяє автоматизувати повний цикл продажу комп'ютерної техніки — від вибору товару в каталозі до його оплати онлайн, що знижує завантаження у менеджерів.

2. Безпечні фінансові операції: Інтеграція національного платіжного шлюзу LiqPay забезпечує безпечний та захищений механізм прийому безготівкових платежів, що забезпечує довіру клієнтам та конверсію магазину.
3. Оптимізація складського обліку: Реалізований механізм резервування товарів розв'язує гостру проблему інтернет-торгівлі — продаж одного і того ж товару кількома покупцями одночасно. Це мінімізує репутаційні ризики та фінансові втрати компанії.
4. Масштабованість: Запропоноване рішення може бути адаптоване не тільки для великих ритейлерів, але й стати базовою платформою для малого та середнього бізнесу (МСБ) в Україні, яка прагне вийти на ринок електронної комерції з мінімальними витратами на ліцензії (оскільки використовує технології з відкритим кодом).
5. Запропоновано та реалізовано надійну архітектурну веб-застосунку на базі фреймворка Python Django, що дозволяє легко розширити функціонал у майбутньому (наприклад, додавання CRM-системи, аналітики або фонових завдань через Celery/Redis).

Глосарій

***Веб-застосунок (Web application*)** — клієнт-серверна програма, в якій клієнтом виступає веб-браузер, а логіка та дані зберігаються на веб-сервері.

Вебхук (Webhook) — механізм розширення або зміни поведінки веб-сторінок чи застосування за допомогою використовуваних зворотних викликів (HTTP callbacks). У контексті проєкту використовується використання для отримання миттєвих сповіщень від LiqPay про зміну статусу платежу.

Електронна комерція (E-commerce) — сфера цифрової економіки, що охоплює всі фінансові та торгові транзакції, що розміщуються за

допомогою комп'ютерних мереж та бізнес-процесів, пов'язаних із проведенням таких транзакцій.

Платіжний шлюз (Payment gateway) — апаратно-програмний комплекс, що автоматизує прийом платежів в Інтернеті та забезпечує безпечну передачу даних між інтернет-магазином, еквайєром та банком клієнта (напр., LiqPay).

Резервування товару (Product reserve) — процес тимчасового вилучення товару з вільного продажу (фіксації залишків на складі) на період, необхідний для завершення процесу оплати.

Стан перегонів (Race condition) — помилка проектування багатопотокових систем, при якій робота системи залежить від того, в якому порядку виконуються паралельні потоки. У e-commerce це відбувається до продажу товару, якого вже немає на складі.

Фреймворк (Framework) — програмна платформа, що визначає структуру програмної системи; програмне забезпечення, що полегшує розробку та об'єднання різних компонентів великого програмного проекту.

CMS (Content Management System) — система управління контентом, інформаційна система або комп'ютерна програма, яка використовується для забезпечення та організації спільного процесу створення, редагування та управління вмістом веб-сайту.

Django — високорівневий веб-фреймворк мовою Python, який сприяє швидкій розробці та чистому, прагматичному дизайну, слідуючи патерну MTV (Model-Template-View).

LiqPay — українська платіжна система, створена ПриватБанком, яка дозволяє проводити моментальні платежі з використанням банківських карток Visa/Mastercard.

Маркетплейс (marketplace) **—** це велика онлайн-платформа, яка об'єднує безліч незалежних продавців та покупців в одному місці, де різні магазини продають свої товари, а сама платформа виступає посередником і гарантом безпеки.

ORM (Object-Relational Mapping) — технологія програмування, яка пов'язує бази даних з концептами об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних". Дозволяє розробникам працювати з BD за допомогою класу Python замість SQL-запитів.

Ритейл (роздрібна торгівля) — це продаж товарів або послуг безпосередньо кінцевим споживачам для їхнього особистого чи домашнього використання.

1 АНАЛІЗ СУЧАСНИХ АНАЛОГІВ І ТЕХНОЛОГІЙ

1.1 Огляд сучасних інтернет-магазинів з продажу комп’ютерної техніки

Ринок електронної комерції в сегменті комп'ютерної техніки (Consumer Electronics & PC Hardware) є одним із найдинамічніших та найбільш конкурентних у світі. Сегмент характеризується швидким оновленням асортименту, високими вимогами споживачів до технічних характеристик, а також необхідністю надання якісних консультацій. Огляд охоплює ключових гравців на цьому ринку України, а також світових лідерів, аналізуючи їхні бізнес-моделі, функціональний та клієнтський досвід а також визначення основних технологічних трендів

1.2 Ключові гравці на ринку України [10, 11]

Український ринок e-commerce в цьому сегменті є високорозвиненим і пропонує споживачеві широкий вибір платформ: від універсальних гігантів до спеціалізованих бутиків.

1.2.1 Універсальні маркетплейси (Rozetka, Moyo, Comfy)

Rozetka.ua

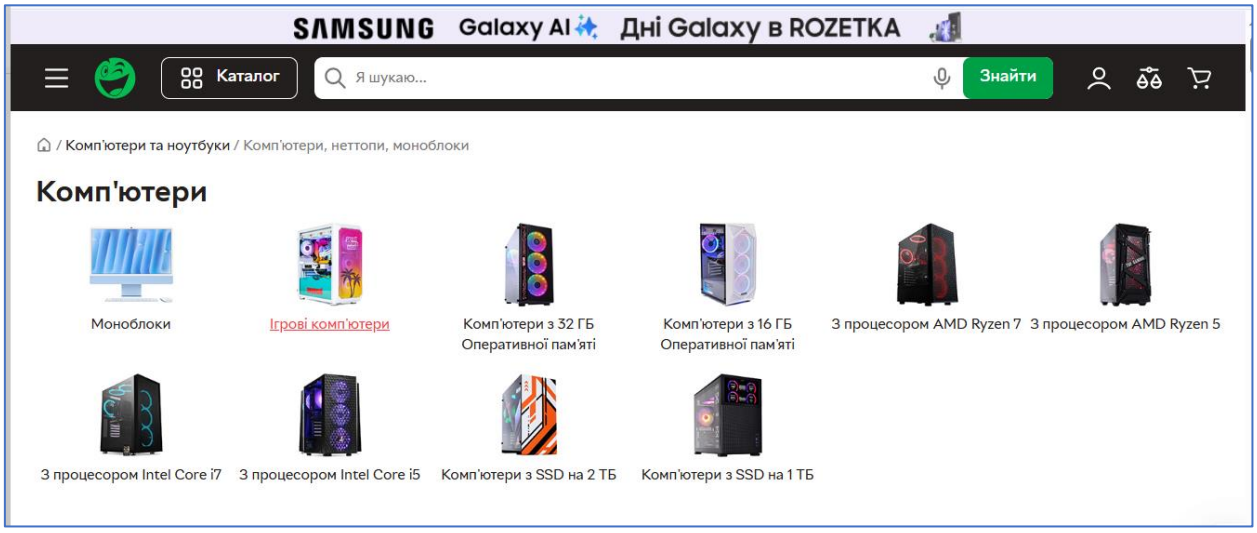


Рисунок *1– Веб-сайт комп’ютерної секції магазину Rozetka*

Rozetka: Беззаперечний лідер української електронної комерції. Починали з електроніки, але перетворилися на "Амазон України". Пропонують найбільший вибір готових рішень, ноутбуків та периферії. Основна перевага — довіра бренду, розгалужена мережа точок видачі та система відгуків.

- Модель: Маркетплейс + власний рітейл.
- Переваги: Найбільший асортимент, розвинена логістика (власні точки видачі), величезна кількість відгуків користувачів, якісна система фільтрації.
- Недоліки: Через перехід на модель маркетплейсу якість обслуговування може варіюватися залежно від продавця.

Telemart.ua

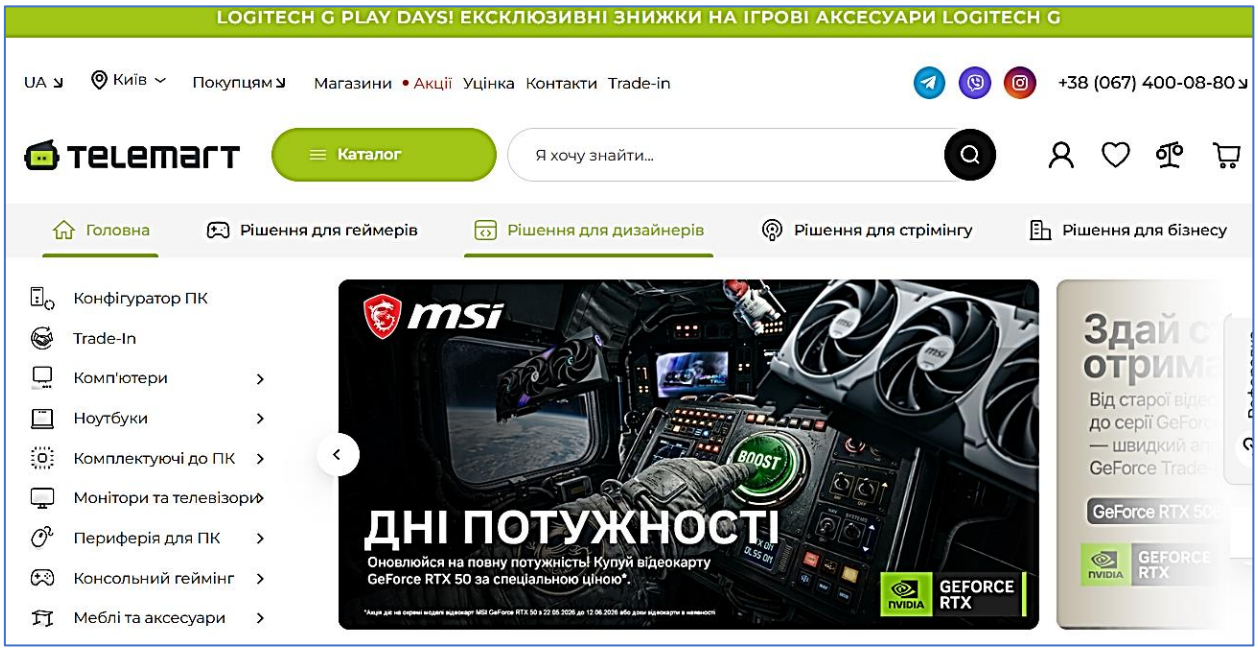


Рисунок *2– Веб-сайт комп'ютерної секції магазину Telemart*

Telemart: Орієнтований на геймерів та кіберспорт. Головна ідея — один із найкращих в Україні онлайн-конфігураторів ПК з автоматичною перевіркою їх сумісності та послуг професійного збирання. Також пропонуємо послугу Trade-In (обмін старих комплектуючих на нові).

- Модель: Спеціалізований магазин комп'ютерної техніки.
- Переваги: Потужний конфігуратор ПК (PC Builder) з перевіркою сумісності компонентів. Орієнтація на геймерів, кіберспортсменів та ентузіастів. Програма Trade-in для комплектуючих.
- Недоліки: Менший асортимент некомп'ютерної периферії з гігантами ринку.

1.2.2 Спеціалізовані магазини комп'ютерної техніки

Brain.com.ua

Brain: Фокусується на B2B-сегменті, корпоративних клієнтах та має розгалужену мережу магазинів. Надають гнучкі умови для системних інтеграторів.

- Модель: Омніканальний рітейл.
- Переваги: Велика мережа фізичних магазинів, розвинена B2B-платформа, гнучка програми лояльності (BrainUp).
- Недоліки: Дизайн та UX сайт дещо застарілі порівняно з конкурентами.

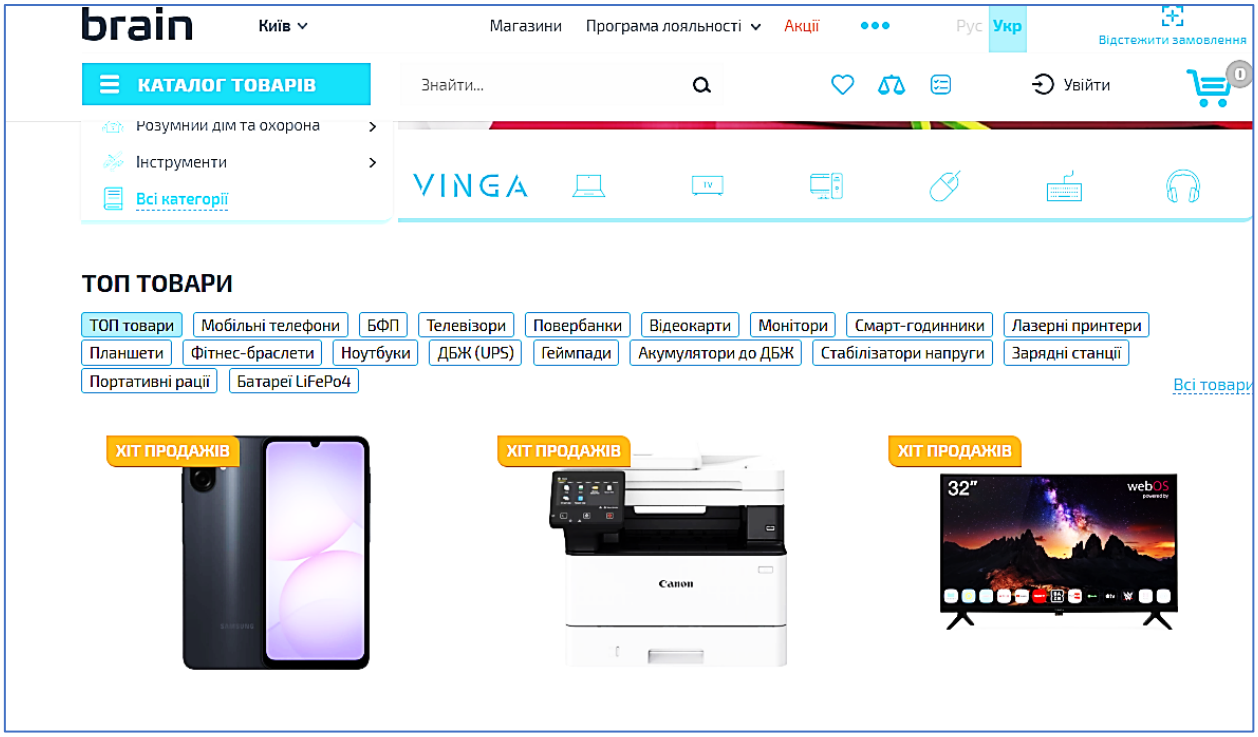


Рисунок *3– Веб-сайт магазину* ***Brain***

Moyo / Comfy / Foxtrot

Moyo / Comfy / Foxtrot: Роблять акцент на омніканальність (поєднання онлайн-вітрин з фізичними магазинами). Їхня сильна сторона — продаж готових пристроїв (ноутбуки, монітори, моноблоки) з можливістю отримати товар того ж дня.

- Модель: Багатоканальні рітейлери широкого профілю.
- Переваги: Широка мережа фізичних магазинів, можливість швидко придбати товар, частина акцій та кредитування на вигідних умовах.
- Недоліки: Глибіна асортимент саме вузькоспеціалізованих комп'ютерних комплектуючих (материнські плати, системи водяного охолодження) значно поступається спеціалізованим магазинам.

Elmir.ua

Elmir (Харків): Магазин із сильною репутацією завдяки дуже привабливим цінам, широкому асортименту рідкісних комплектуючих та чесній гарантійній політиці.

1.2.3 Огляд магазинів комп’ютерної техніки Запоріжжя

itPlanet https://www.itplanet.zp.ua/



Рисунок *4– Інтерфейс магазину itPlanet*

Комп’ютерний всесвіт https://kvshop.com.ua/

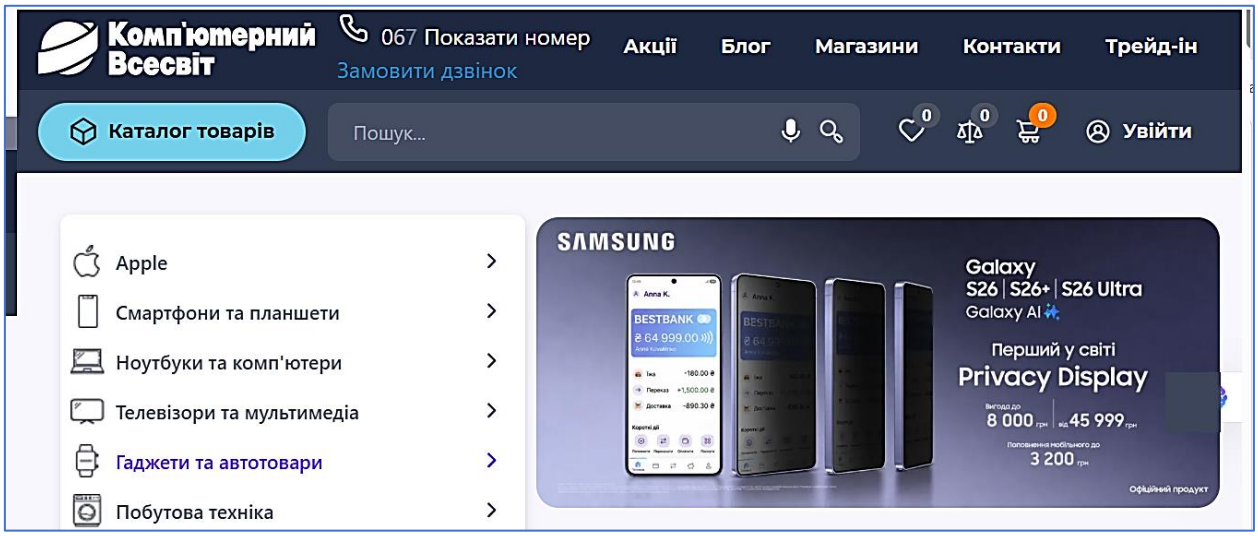


Рисунок *5* – *Інтерфейс магазину Комп’ютерний всесвіт*

Microtron *https://microtron.ua/*

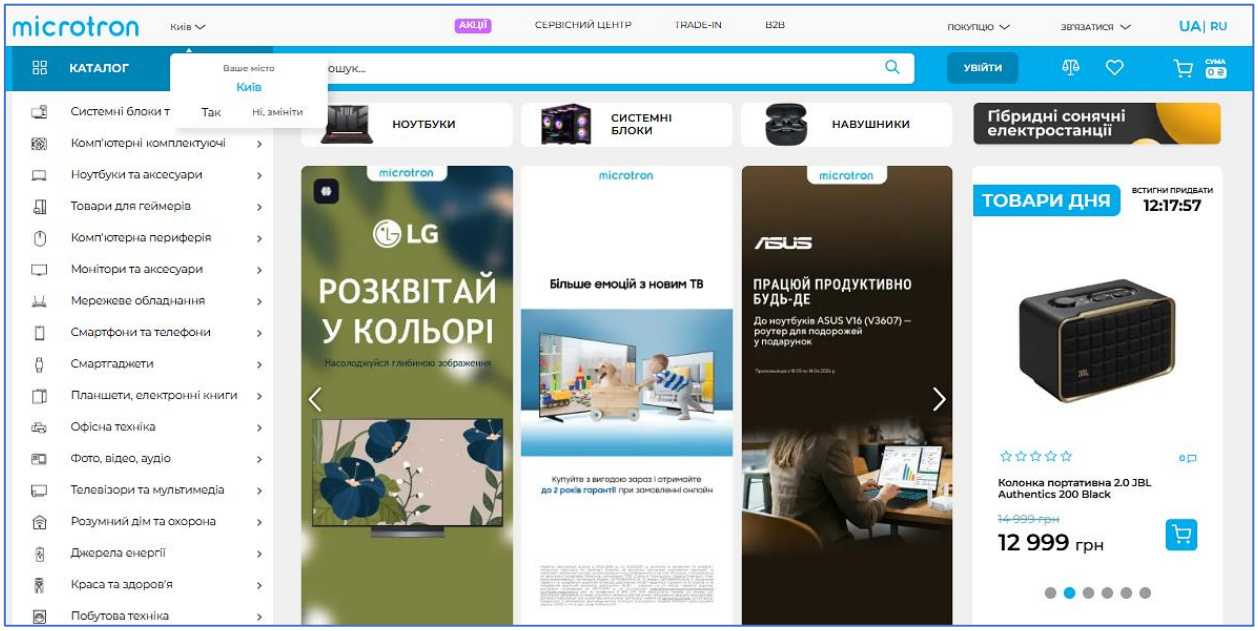


Рисунок *6*– *Інтерфейс магазину Microtron*

1.3 Світові лідери (США та Європа) [12]

Newgg (США) https://www.newegg.com/

Newegg знайшов золотий стандарт для продажу комп'ютерних комплектуючих у світі.

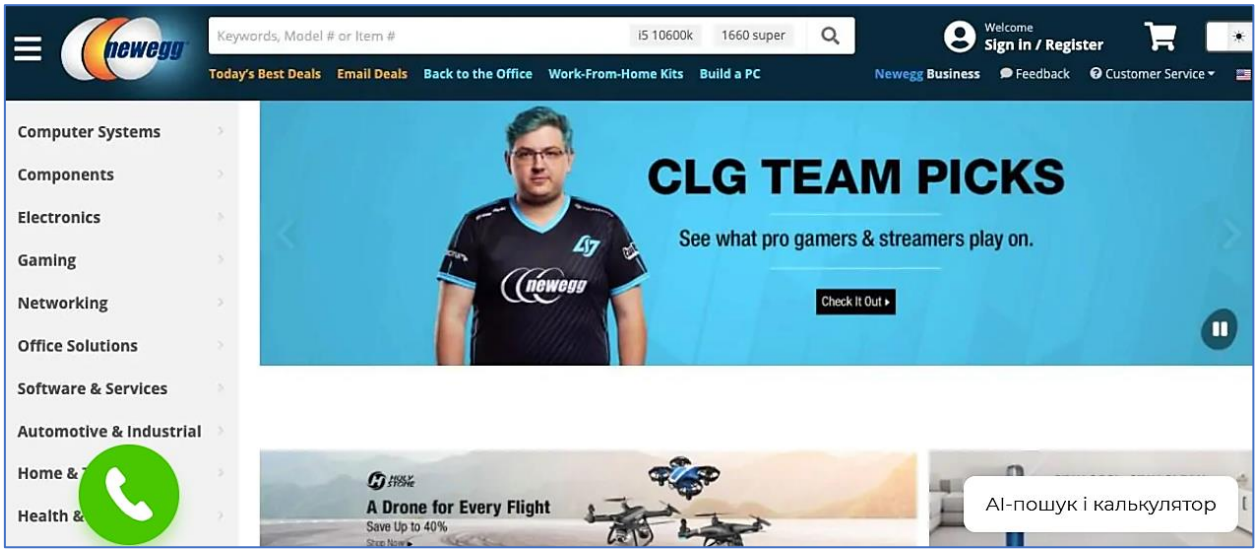


Рисунок *7 – Інтерфейс магазину Newgg*

- Цільова аудиторія: Геймери, ентузіасти, професіонали, які самостійно збирають ПК (DIY-сегмент).
- Особливості: Глибока та надзвичайно деталізована система фільтрації (наприклад, вибір материнської плати не лише за сокетом, а й за недостатню фазу живлення). Вбудований "PC Builder" із автоматичною перевіркою сумісності компонентів. Впровадження оплати криптовалютою.
- Бізнес-модель: Об'єднання власних продажів та моделі маркетплейсу (B2C + B2B).
- Специфіка: Найбільший у світі спеціалізований маркетплейс електроніки.

Microcenter (США) https://www.microcenter.com/

- Специфікація: Гібридна модель (онлайн + потужні офлайн-хаби).
- Особливості: вони часто змушують клієнтів вибирати найдефіцитніші товари (наприклад, новітні відеокарти) у фізичних магазинах, щоб запобігти масовим скупкам скальперами та ботами.

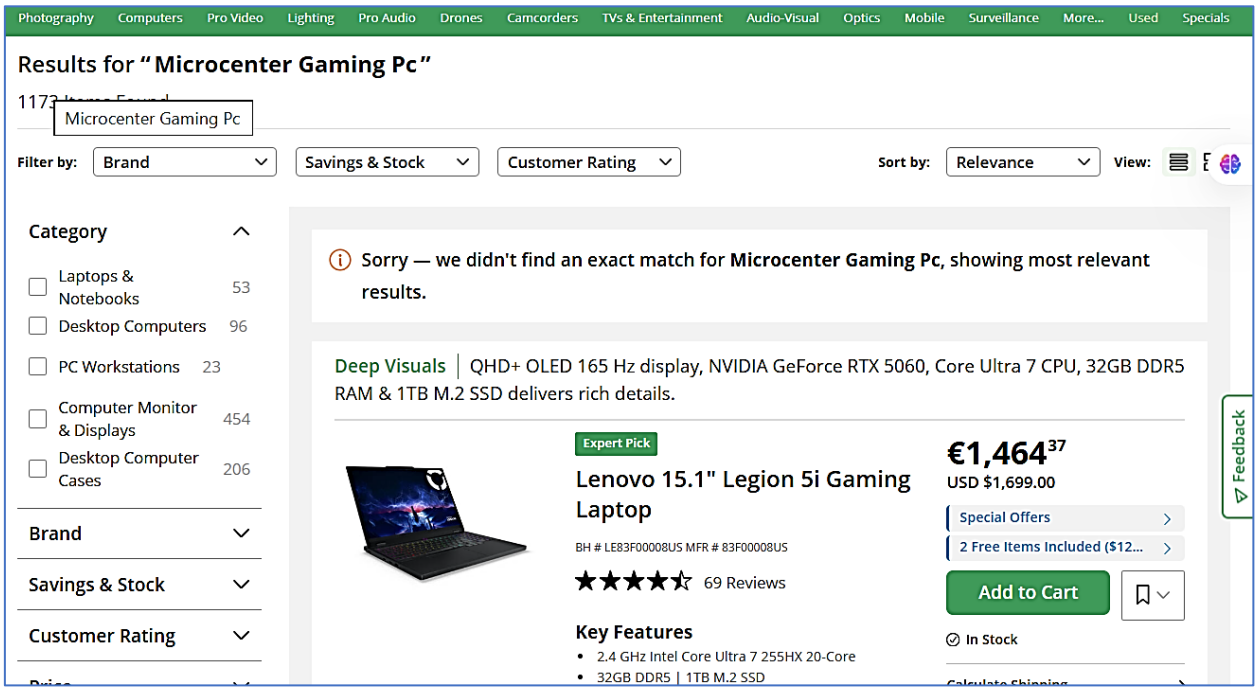


Рисунок *8 – Інтерфейс магазину Microcenter*

Amazon https://www.amazon.com/Computers

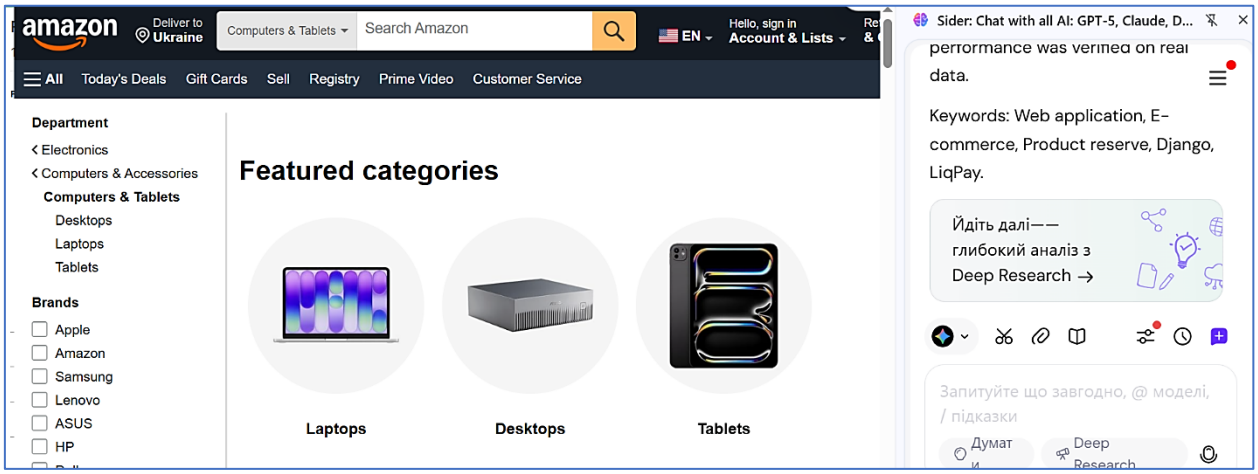


Рисунок *8 – Веб-сайт секції магазину Amazon*

Amazon пропонує величезні обсяги готових ПК та периферії, хоча й продається в нових магазинах у зручності для ентузіастів.

- Конкурентні переваги: Логістика (Amazon Prime), проста політика повернення, величезна кількість реальних відгуків, агресивне ціноутворення.
- Недоліки: Слабка системи технічної фільтрації комплектуючих, ризик натрапити на підготовку від сторонніх продавців.
- Специфіка: Глобальний маркетплейс.
- Особливості: Швидка доставка (Prime), найкраща у світі політика повернення товару, проте пошук та фільтрація самих технічних параметрів часто є незручними.

1.4 Порівняльний аналіз функціоналу магазинів

Конфігурація ПК (PC Builders)

Це ключова функція для спеціалізованих магазинів (Telemart, PCPartPicker). Інтерактивний інструмент, який не дозволяє користувачеві звернути процесор Intel для материнської плати з сокетом AMD.

Конфігуратор має автоматично перевіряти:

- Сумісність сокета процесора та материнської плати.
- Відповідність форм-фактору корпусу та материнської плати/довжини відеокарти.
- Достатність потужності блоку живлення.

Розширена параметрична фільтрація

Це можливість шукати відеокарту лише за брендом, а й за довжиною (в мм), кількістю слотів, тепловиділенням (TDP) тощо Для комп'ютерної техніки критичним є пошук із сотнями параметрів (частота пам'яті, таймінги, типи матриць моніторів, тепловиділення TDP тощо). Найкраще з цим справляються Rozetka та Newegg.Для успішної конкуренції на ринку платформа повинна мати спеціальний функціонал:

Порівняння товарів

Це можливість зручно порівнювати десятки технічних зі збереженням характеристик сесії.

Синхронізація складів (ERP/CRM)

Це відображення реальних залишків на різних складах, механізми резервування товару в кошику (наприклад, на 15 хвилин при оплаті через LiqPay).

1.5 Сучасні тренди електронної комерції в IT-сегменті

Штучний інтелект у рекомендаціях: Підбір супутніх товарів (наприклад, термопаста до процесора або сумісна оперативна пам'ять). Замість класичного "з цим товаром купують", впроваджуються AI-асистенти, які підказують: "Цей блок живлення заслабкий для обраної вами відеокарти".

AR (Доповнена реальність): Можливість віртуально "поставити" системний блок на стіл у кімнаті.

Гейміфікація: Програми лояльності з досягненнями (ачівками) за покупки та відгуки.

Криптовалютні платежі та розстрочки: Інтеграція сервісів платіжними частинами стає критично важливою, оскільки середня вартість ігрового ПК постійно зростає.

1.6 Огляд засобів та технологій розробок інтернет-магазинів комп'ютерної техніки

Розробка інтернет-магазину для продажу комп'ютерної техніки має свою специфіку: складне дерево категорій, величезна кількість характеристик (фільтрація), необхідність інтеграції з обліковими системами та наявність унікального функціоналу (наприклад, конфігуратор збірки ПК

з перевіркою сумісності комплектуючих). Вибір технологічного стеку є важливим етапом, який забезпечує здатність проекту до масштабування.

1.6.1 Готові CMS та SaaS-платформи [12]

Magento (Adobe Commerce)

- Опис: Найпотужніша платформа з відкритим вихідним кодом для enterprise-сегменту.
- Переваги: Ви постійно величезні каталоги товарів, має просунуті інструменти для B2B та B2C, потужну систему кешування. Ідеально підходить для продажу складної електроніки з тисячами атрибутів.
- Недоліки: Високий поріг входу, дорогі розробники, високі вимоги до серверного обладнання.

Shopify

- Опис: Хмарна (SaaS) платформа для швидкого запуску.
- Переваги: Не потрібно адмініструвати сервери, величезна екосистема плагінів, висока безпека та надійність.
- Недоліки: Абонентська плата, комісія з кожної транзакції, складність реалізації спеціального функціоналу (наприклад, складного конфігуратора ПК).

WooCommerce (WordPress)

- Опис: Плагін для WordPress, який перетворює його на магазин.
- Переваги: Безкоштовний, простий у використанні, підходить для малого бізнесу з обмеженим бюджетом.
- Недоліки: При великій кількості товарів (що характерно для комп'ютерних магазинів) та фільтрів повністю суттєво гальмувати, потребує оптимізації.

1.6.2 Фреймворки для Custom-розробки

Якщо готових рішень недостатньо (наприклад, потрібна унікальна логіка резервування товарів, складна програма лояльності чи інтеграція з конкретними постачальниками), використовується індивідуальна розробка.

Backend-розробка:

- Django (Python): Хороший вибір для електронної комерції. Має вбудовану потужну адмін-панель, високий рівень безпеки та дозволяє легко інтегрувати системи машинного навчання (наприклад, для рекомендаційних товарів) [1–3].
- Node.js (Express / NestJS): Забезпечує високу швидкість обробки великої кількості одночасних запитів (наприклад, у дні розпродажів на зразок Чорної П'ятниці). Часто використовується разом з мікросервісною архітектурою.
- Laravel (PHP): Один із найпопулярніших бекенд-фреймворків. Має готові екосистеми електронної комерції (наприклад, Bagisto), на якій легко знайти розробників для підтримки.

Frontend-розробка:

Для забезпечення високої швидкості роботи інтерфейсу (швидка фільтрація без перезавантаження сторінок) зараз домінує підхід SPA (Single Page Application) або SSR (Server-Side Rendering).

- React.js / Next.js: Найпопулярніший вибір. Дозволяє створювати складні інтерактивні елементи. Next.js додатково вирішує проблему SEO-оптимізації, яка критично важлива для отримання трафіку.
- Vue.js / Nuxt.js: Дуже популярний фреймворк, який часто використовується в поєднанні з Laravel та Django.

Бази даних та системи пошуку:

- Реляційні БД (PostgreSQL, MySQL, SQLite): Основа для зберігання замовлень, користувачів, транзакцій та структурованого каталогу товарів. Забезпечують цільність даних (ACID транзакції) [4].
- Redis: база даних у пам’яті. Критично важлива для зберігання сесійних користувачів, кошиків товарів та реалізації механізму тимчасового резервування товарів перед оплатою, а також для кешування результатів баз даних.

- Elasticsearch / Algolia: Спеціалізовані пошукові рушії. Комп'ютерний магазин вимагає розумного морфологічного пошуку, автодоповнення та дуже швидкої роботи фасетних фільтрів (за брендом, сокетом процесора, обсягом пам'яті тощо). Звичайні бази даних справляються з цим значно ширше.

Сторонні інтеграції та API:

- Платіжні системи: LiqPay, WayForPay, Stripe (через REST API з використанням веб-хуків для підтвердження транзакцій) [6–7].
- Служби доставки: API Нової Пошти (генерація ТТН, розрахунок доставки вартості пошуку, відділ у реальному часі).
- Облікові системи (ERP/CRM): 1С, BAS, Бітрікс24 для синхронізації залишків комплектуючих на складах у реальному часі.

1.7 Висновки з розділу 1

1. Ринок інтернет-продажів комп'ютерної техніки чітко розділений на дві великі категорії: маркетплейси широкого профілю (змагаються ціною та логістикою) та нові спеціалізовані магазини (змагаються експертизою, сервісом та інструментами типу ПК-конфігураторів).
2. Під час розробки власного магазину варто орієнтуватися на спеціалізовані інструменти зручні фільтри, збірка ПК, після чого можна конкурувати з гігантами в загальному сегменті. Для створення успішного нового проекту (наприклад, на базі Django) недостатньо просто розробити каталог і кошик. Необхідно інтегрувати розумну бізнес-логіку: перевірку сумісності, надійне резервування (щоб уникнути оверселінгу), автоматизовану логістику та безпечні платіжні шлюзи.
3. Якщо бізнес планує масштабувати, продавати тисячі найменувань, мати складні алгоритми рекомендацій, систему збирання ПК з перевіркою сумісності та високі завантаження, оптимальним рішенням стає

Custom-розробка на базових фреймворках (Django, Laravel або Node.js) у поєднанні з сучасними Frontend-технологіями (React/Vue) та потужними пошуковими системами типу Elasticsearch. В рамках завданнь для кваліфікаційної роботи доцільно обрати фреймворк Django з розробкою простого веб-інтерфейсу застосунку

4. Огляд веб-сайтів магазинів з продажу комп’ютерів дозволив визначити основні функції можливості цих сайтів та визначити основні можливості користувацьких інтерфейсів..

2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ З ПРОДАЖУ КОМП'ЮТЕРНОЇ ТЕХНІКИ

2.1 Розробка архітектури програмної системи

2.1.1 Загальна архітектурна модель MVT

Архітектура розробленого веб-застосунку базується на патерні **MVT (Model-View-Template)** , який є стандартом для фреймворка Django. Цей підхід забезпечує чіткий розподіл відповідальності між даними, бізнес-логікою та інтерфейсом користувача [13–15]:

- **Модель (Model):** Відповідає за структуру даних та взаємодію з базою даних. Описується у вигляді Python-класів, які Django ORM (Object-Relational Mapping) перетворює в SQL-запити.
- **View (Представлення/Контролер):** Містити основну бізнес-логіку програми. Виконує HTTP-запити від користувача, звертається до моделей для отримання/збереження даних, обробляє їх і повертає відповідь (найчастіше — відрендерений HTML-шаблон або JSON для API).
- **Template (Шаблон):** Відповідає за презентаційний шар (User Interface). Використовує мову шаблонів Django для динамічної генерації HTML-сторінок на основі даних, переданих з View.

2.1.2 Основні компоненти системи та їх взаємодія

Для забезпечення високої продуктивності, надійності та можливості масштабування система складається з кількох взаємопов'язаних компонентів:

1. **Клієнтська частина (Frontend):** Браузер користувача, який відображає HTML/CSS/JS. Відповідає за адаптивний дизайн та базову валідність форми.

2. **Веб-сервер та WSGI-контейнер:** приймає запити (наприклад, Nginx) та передає їх до Django-додатку через WSGI-сервер (наприклад, Gunicorn).

3. **Ядро застосунку (Django App):** Обробляє маршрутизацію (URL-адреси), бізнес-логіку кошика, оформлення замовлень та взаємодію з API сторонніх сервісів.

4. **База даних (RDBMS):** Реляційна база даних (рекомендується SQLLite) для зберігання інформації про товаришів, користувачів і замовлення транзакцій. Підтримка PostgreSQL критична для коректної роботи блокування баз даних при резервуванні.

5. **Брокер повідомляє та фонові задачі (Redis + Celery):** Використовується для асинхронного виконання важких або відкладених завдань (наприклад, автоматичне скасування неоплачених замовлень через 30 хвилин та зняття резерву з товару).

6. **Зовнішній платіжний шлюз (API LiqPay):** Сторонній сервіс для безпечної обробки транзакцій.

2.1.3 Проєктування бази даних [4, 9]

Архітектура даних побудована навколо таких основних сутностей:

- Product (Товар): Містити атрибути комп'ютерної техніки (назва, опис, характеристики, ціна, *доступна кількість на складі*).
- Category (Категорія): Ієрархічна структура для класифікації техніки (ноутбуки, комплектуючі, периферія).
- Customer (Користувач): Розширена модель користувача для зберігання персональних даних та історії замовлень.
- Order (Замовлення): Агрегує інформацію про покупця, статус оплати, загальну суму та метадані платежу від LiqPay.
- OrderItem (Елемент замовлення): Зв'язує замовлення з конкретними товарами, фіксуючи ціну на момент покупки та зарезервовану кількість.

- Payment_transaction (Платіжна транзакція): Фіксує статус, дату та суму розрахунку за замовлення.

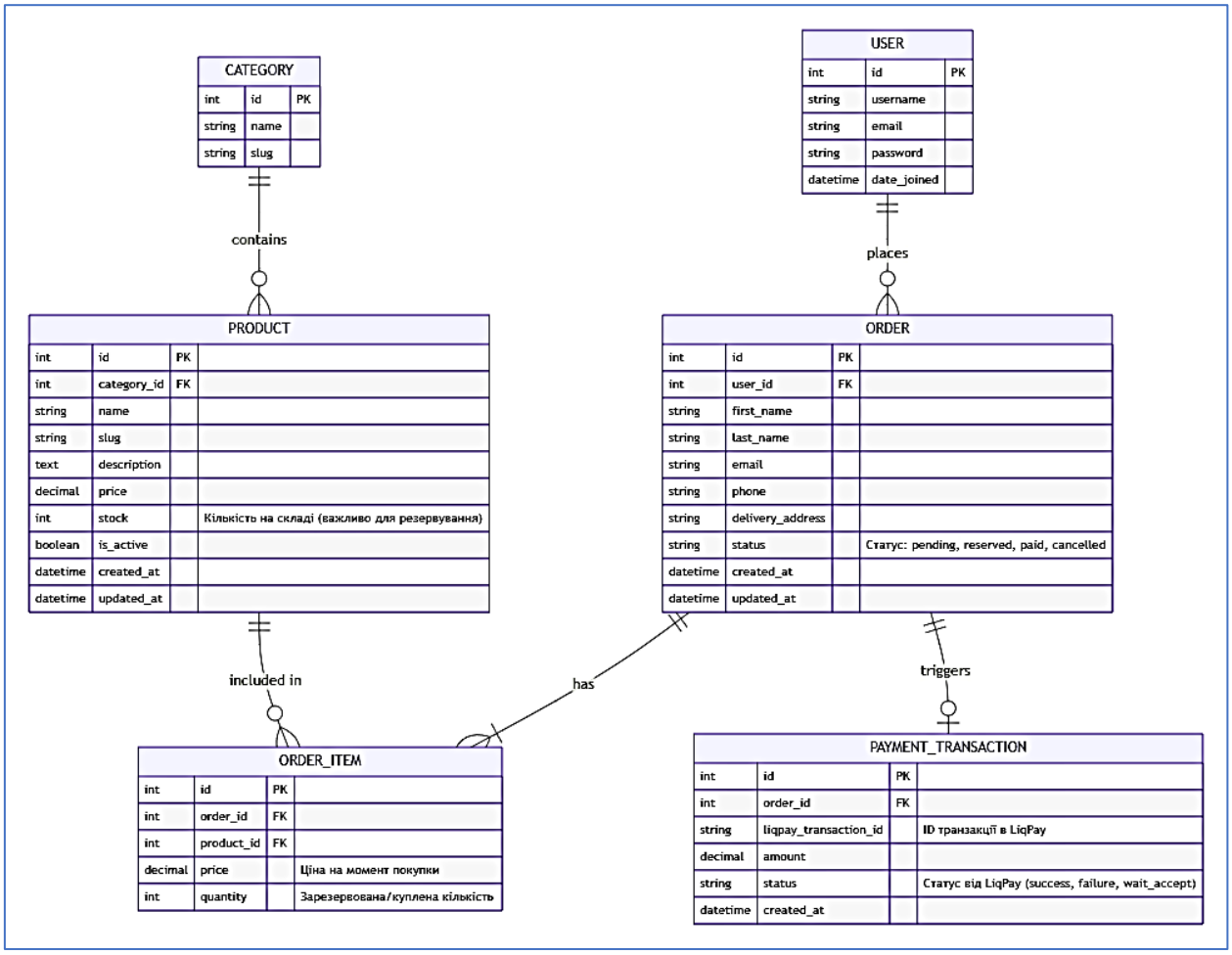


Рисунок *9– ER-діаграму даних для інтернет-магазину*

Ця схема враховує вимоги до застосунку, зокрема:

1. **Резервування товарів**: поле stock у таблиці PRODUCT та поле status у таблиці ORDER (яке може приймати значення reserved на час очікування оплати).
2. **Інтеграція платіжної системи**: окрема таблиця PAYMENT_TRANSACTION для збереження даних від LiqPay, що дозволяє відстежувати статуси транзакцій без втручання в основну таблицю замовлень.

3. **Фіксація цін**: таблиця ORDER_ITEM зберігає ціну на момент формування замовлення, що є критично важливим для e-commerce (якщо ціна товару зміниться пізніше, в історії замовлень вона залишиться коректною).

2.1.4 Архітектура механізму резервування товарів

Одним із архітектурних завдань у проекті є уникнення стану "перегонів" (stace race condition), коли кілька користувачів одночасно намагаються купити останній товар.

Алгоритм рішення:

1. **Ініціалізація замовлення:** Коли користувач натискає "Оформити замовлення", система відкриває **атомарну транзакційну** базу даних (transaction.atomic).
2. **Блокування записів:** Застосовується метод select_for_update(), який блокує рядки товарів у базі даних для інших транзакцій до завершення поточної.
3. **Перевірка залишків:** Система перевіряє, чи достатньо товару на складі.
4. **Списання:** якщо товар достатній, його кількість на складі зменшується (резервується), а транзакція успішно фіксується (commit).
5. **Тайм-аут резерву:** Замовлення отримує статус "Очікує оплату". Одночасно створити відкладену задачу в Celery. Якщо протягом визначеного часу (наприклад, 15 хвилин) оплата не перевищується, Celery-worker змінює статус замовлення на «Скасовано» і повертає зарезервовану кількість товарів назад на склад.

2.1.5 Архітектура платіжної інтеграції (LiqPay Flow)

Взаємодія з платіжною системою побудована за асинхронним принципом з використанням Webhooks (Server-to-Server callbacks):

1. Формування підпису: Після створення замовлення Django-додаток формує пакет даних (сума, валюта, ID замовлення) та генерує криптографічний підпис з використанням приватного ключа LiqPay.
2. Редирект на Checkout: користувач перенаправляється на захищену сторінку LiqPay для введення карткових даних.
3. Обробка Webhook (Callback): Після успішної або неуспішної оплати LiqPay відправляє POST-запит на спеціальну кінцеву точку нашого застосунку.
4. Валідація та оновлення стану: Система перевіряє цифровий підпис отриманого запиту (для підтвердження, що він дійсно від LiqPay). Якщо статус success, замовлення переводиться в статус "Оплачено", користувачу відправляється квітація.

2.1.6 Безпека архітектури

Архітектура враховує сучасні вимоги кібербезпеки:

- Захист від CSRF-атак за допомогою вбудованого проміжного програмного забезпечення Django.
- Екранування даних у шаблонах для запобігання XSS-ін'єкціям.
- Зберігання паролів у хешованому вигляді (алгоритм PBKDF2).
- Використання змінних оточень (змінних середовища) для зберігання секретних ключів (SECRET_KEY, LIQPAY_PRIVATE_KEY) поза репозиторієм коду.

2.2 Проєктування та розробка користувацького інтерфейсу веб-застосунку

Розроблений користувацький інтерфейс (User Interface, UI) інтернет-магазину комп’ютерної техніки, заснований на принципах ергономіки, інтуїтивної зрозумілості та адаптивного веб-дизайну (Responsive Web Design). Головною програмою етапу проектування інтерфейсу було

забезпечено оптимальний користувальницький досвід (User Experience, UX) під час взаємодії з каталогом товарів та процесом оформлення замовлення.

Для реалізації візуальної частини (клієнтського шару) застосована мова гіпертекстової розмітки HTML5 та сучасний CSS-фреймворк Tailwind CSS. Вибір Tailwind CSS, який використовує парадигму функціональних класів (utility-first), зумовлений забезпеченням гнучкості стилізації, зменшення розміру кінцевого CSS-файлу та підвищення швидкості відтворення сторінок у браузері користувача.

У контексті архітектурного патерну MTV (Model-Template-View) фреймворк Django, розроблений інтерфейс виконує роль шаблону (Template). Візуальний макет інтегровано з бекенд-частиною за допомогою механізму шаблонізації Django (Django Template Language), що забезпечує серверний рендеринг (Server-Side Rendering) та динамічне доповнення сторінок даними, які виявляються з реляційної бази даних.

Структурно головна сторінка веб-застосунку поділена на наступні функціональні блоки (Рис. 10):

1. Глобальна панель навігації (Header): Забезпечує наскрізну навігацію веб-застосунком. Містити логотип компанії, інтерактивне меню з основними категоріями техніки (ноутбуки, комплектуючі, периферія) та елемент доступу до кошика.

2. Блок привернення уваги (Hero Section): Розташований у верхній частині сторінки та призначений для демонстрації актуальних пропозицій або флагманських продуктів компанії.

3. Каталог товарів (Product Grid):Основний інформаційний блок, реалізований у конфігурації адаптивної сітки (на базі технології CSS Grid Layout). Сітка автоматично масштабується та змінює кількість колонок залежно від роздільної здатності екрана пристрою, що забезпечує коректне відображення як на настільних моніторах, так і на мобільних телефонах.

4. Картка товару (Product Card): Компонент інтерфейсу, що відображає базову інформацію про конкретну товарну позицію. Ця картка

містить фотографію комп'ютерної техніки, найменування, ключові технічні характеристики, ціну та кнопку додавання до кошика. Взаємодія з цією кнопкою є тригером для запуску бізнес-логіки бекенда (резервування товару в БД).

Модульна структура інтерфейсу забезпечує високий рівень масштабованості фронтенд-частини застосунку: додавання нових блоків або зміна існуючих компонентів не вимагає глибокого рефакторингу коду на всій сторінці.

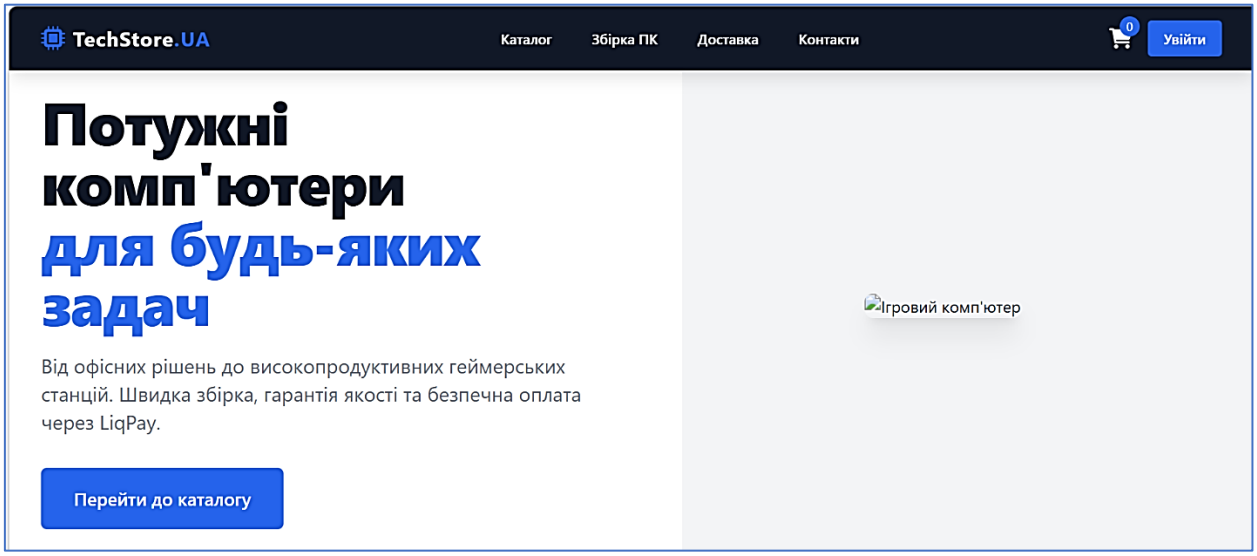


Рисунок *10 – Структурно головна сторінка веб-застосунку*

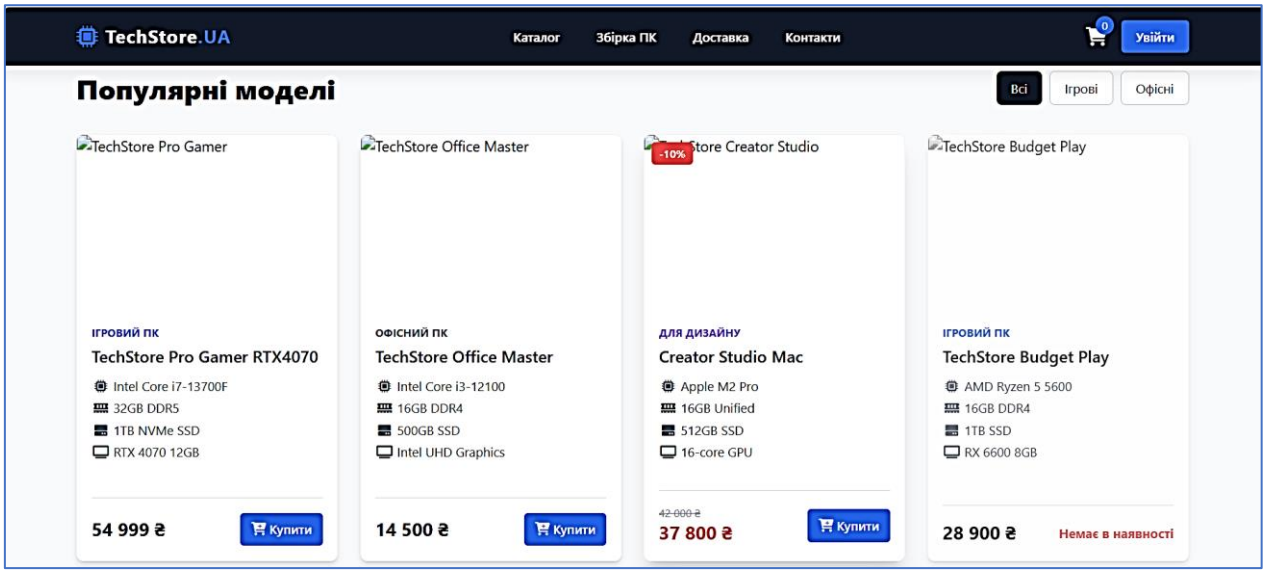


Рисунок *3 – Сторінка каталогу товарів*

2.3 Висновки з розділу 2

1. Архітектура розробленого веб-застосунку базується на патерні MVT (Model-View-Template) , який є стандартом для фреймворка Django.
2. Для забезпечення високої продуктивності, надійності та можливості масштабування система складається з кількох взаємопов’язаних компонентів:
   - клієнтської частини,
   - веб-сервера,
   - ядра застосунку.
4. Розроблена структура реляційної бази даних, що реалізує базові сутності предметної області.
5. Спроєктований інтерфейс веб-застосунку з продажу комп’ютерної техніки.

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ З ПРОДАЖУ КОМП'ЮТЕРНОЇ ТЕХНІКИ

3.1 Створення проєкту веб-сайту в фреймворку Django

зібрати розроблений код в єдиному робочому проекті Django, налаштувати базу даних, фонові задачі (Celery) і підготувати середовище для тестування платежів (LiqPay).

1. Системні вимоги

Перед початком перевірте, що на вашому комп'ютері встановлено:

1. Python (версії 3.10 або вище)
2. Redis (необхідний як брокер повідомляє для Celery).
   - *Windows:* рекомендується інсталювати через WSL2 або використовувати Docker.
3. Ngrok (необхідний для прокидання локального порту в Інтернеті, щоб LiqPay міг надсилати Webhook-запити на ваш локальний комп'ютер).

2. Створення середовища та встановлення залежностей

Відкриваємо термінал і виконуємо наступні команди:

```
# 1. Створюємо директорію проєкту
mkdir computer_shop
cd computer_shop
# 2. Створюємо та активуємо віртуальне середовище
python -m venv venv
: venv\Scripts\activate # Для Windows
# 3. Встановлюємо необхідні бібліотеки
pip install django celery redis liqpay-python
```

3. Ініціалізація проекту та додатку

```
# 1. Створюємо основний проєкт (назвемо його core)
django-admin startproject core .
# 2. Створюємо додаток магазину
```

```
python manage.py startapp shop
```

4. Структура файлів

Розділимо файли, , за відповідними каталогами:

```
computer_shop/
│
├── core/                # Головна директорія проєкту
│   ├── __init__.
│   ├── asgi.py
│   ├── celery.py        # Конфігурація Celery
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
├── shop/                # Наш додаток
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py        # ВСТАВИТИ розроблений код моделей
│   ├── services.py      # ВСТАВИТИ розроблений код логіки резервування
│   ├── tasks.py         # ВСТАВИТИ розроблений код Celery задач
│   ├── urls.py          # ВСТАВИТИ розроблений код маршрутів
│   └── views.py         # ВСТАВИТИ розроблений код вебхука LiqPay
│
├── venv/
└── manage.py
```

3.2 Створення моделей для ORM системи в Django

На основі спроєктованої в Розділі 2 ER-діаграми (Рис.10) створимо об’єктно-орієнтовані моделі для реалізації у вбудованій Django ORM і створення бази даних в СУБД SQLite. У Django використовується власна вбудована ORM— Django ORM. Вона постачається «з коробки» разом із самим фреймворком. Django ORM підтримує роботу з більшістю популярних реляційних баз даних, таких як PostgreSQL, MySQL, SQLite та Oracle, дозволяючи взаємодіяти з ними за допомогою об'єктно-орієнтованого коду на Python замість написання сирих SQL-запитів.

Реалізація рівня доступу до даних (Data Access Layer) у патерні MVT (Model-View-Template), який використовує фреймворк Django. Кожен клас тут відповідає за створення окремої таблиці в реляційній базі даних, а атрибути класів визначають колонки цих таблиць.

Розберемо архітектуру цих моделей детальніше, звернувши увагу на ключові рішення, які є критично важливими для інтернет-магазину.

1. Моделі Category та Product (Каталог товарів)

Клас Category відповідає за ієрархію товарів. Ключовим полем тут є slug (URL-ідентифікатор). У контексті SEO-оптимізації інтернет-магазинів комп'ютерної техніки важливо мати людинозрозумілі адреси (наприклад, /category/laptops/), для чого і використовується тип SlugField.

Клас Product містить базову інформацію про товар:

- Зв'язок category = models.ForeignKey(...) створює відношення "один-до-багатьох" (одна категорія може мати багато товарів). Параметр related_name='products' дозволяє отримувати всі товари категорії через category.products.all().
- Поле price використовує DecimalField. Для фінансових операцій категорично заборонено використовувати тип Float, оскільки він має проблеми з точністю арифметики з плаваючою комою. DecimalField гарантує абсолютну точність до копійок.

- Поле stock (PositiveIntegerField) є фундаментальним для нашого механізму резервування. Воно відображає фактичну кількість товару на фізичному складі.

2. Моделі OrderIs та OrderItem (Оформлення замовлень)

Ці дві моделі розділяють логіку замовлення на загальні дані та конкретні позиції (товари) в кошику.

Клас Order представляє сам факт замовлення. Найважливішим тут є кортеж STATUS_CHOICES. Статус reserved є ключовим для бізнес-логіки: він сигналізує системі, що товар фізично віднято з поля stock моделі Product, але оплата ще очікується. Якщо статус переходить у cancelled, фонова задача (наприклад, Celery) має повернути кількість товару назад на склад.

Клас OrderItem є проміжною сутністю (позицією в чеку):

product = models.ForeignKey(..., on_delete=models.PROTECT).

Тут навмисно використано PROTECT замість CASCADE. Якщо магазин вирішить видалити товар з бази даних, історія замовлень клієнтів, які купили цей товар раніше, не повинна бути видалена чи пошкоджена. База даних заборонить видалення товару, якщо він фігурує в існуючих замовленнях.

price = models.DecimalField(...).

Збереження ціни безпосередньо в OrderItem є стандартом e-commerce. Ціни на комп'ютерну техніку постійно змінюються (через курси валют, акції тощо). Якщо клієнт купив відеокарту за 15 000 грн, а завтра вона коштує 18 000 грн, в історії його замовлення має назавжди залишитись зафіксована ціна на момент покупки.

3. Модель PaymentTransaction (Інтеграція LiqPay)

Цей клас ізолює логіку роботи з платіжним шлюзом від загальної логіки замовлень.

- Зв'язок order = models.OneToOneField(...) гарантує, що одному замовленню відповідає лише одна активна платіжна сесія.

- Поля liqpay_order_id та payment_id зберігають ідентифікатори транзакцій для двосторонньої синхронізації між вашим сервером та сервером LiqPay.
- STATUS_CHOICES тут відрізняється від статусів замовлення. Вони дублюють офіційні статуси вебхуків LiqPay (wait_accept, success, failure, reversed). Це дозволяє вашій системі коректно обробляти callback-запити від банку.

4 Метадані та допоміжні методи

У кожному класі присутній внутрішній клас Meta. Він вказує Django, як називати ці сутності в панелі адміністратора (в однині та множині) і за яким критерієм їх сортувати (наприклад, ordering = ['-created_at'] показуватиме найновіші замовлення чи товари першими). Методи __str__ забезпечують читабельне відображення об'єктів у консолі та адмінці.

Моделі реалізовані у вигляді класів Python в Лістингу 1.

Лістинг *1 – Реалізація моделей для інтернет-магазину*

```
from django.contrib.auth.models import User
from decimal import Decimal

class Category(models.Model):
        name = models.CharField(max_length=255,
verbose_name="Назва категорії")
        slug = models.SlugField(max_length=255,
unique=True, verbose_name="URL-ідентифікатор")
        description = models.TextField(blank=True,
verbose_name="Опис")

        class Meta:
                verbose_name = "Категорія"
                verbose_name_plural = "Категорії"
                ordering = ['name']
```

```
        def __str__(self):
                return self.name

class Product(models.Model):
        category = models.ForeignKey(Category,
related_name='products', on_delete=models.CASCADE,
verbose_name="Категорія")
        name = models.CharField(max_length=255,
verbose_name="Назва товару")
        slug = models.SlugField(max_length=255,
unique=True, verbose_name="URL-ідентифікатор")
        description = models.TextField(blank=True,
verbose_name="Опис товару")
        price = models.DecimalField(max_digits=10,
decimal_places=2, verbose_name="Ціна")
        stock = models.PositiveIntegerField(default=0,
verbose_name="Кількість на складі")
        is_active = models.BooleanField(default=True,
verbose_name="Активний")
        created_at =
models.DateTimeField(auto_now_add=True,
verbose_name="Створено")
        updated_at = models.DateTimeField(auto_now=True,
verbose_name="Оновлено")

        class Meta:
                verbose_name = "Товар"
                verbose_name_plural = "Товари"
                ordering = ['-created_at']

        def __str__(self):
```

```
                return f"{self.name} ({self.price} грн)"

class Order(models.Model):
        STATUS_CHOICES = (
                ('pending', 'Очікує оформлення'),
                ('reserved', 'Зарезервовано (очікує
оплати)'),
                ('paid', 'Оплачено'),
                ('cancelled', 'Скасовано'),
                ('shipped', 'Відправлено'),
        )

        user = models.ForeignKey(User,
related_name='orders', on_delete=models.CASCADE,
verbose_name="Користувач")
        status = models.CharField(max_length=20,
choices=STATUS_CHOICES, default='pending',
verbose_name="Статус")
        total_amount = models.DecimalField(max_digits=10,
decimal_places=2, default=Decimal('0.00'),
verbose_name="Загальна сума")
        created_at =
models.DateTimeField(auto_now_add=True,
verbose_name="Створено")
        updated_at = models.DateTimeField(auto_now=True,
verbose_name="Оновлено")

        class Meta:
                verbose_name = "Замовлення"
                verbose_name_plural = "Замовлення"
                ordering = ['-created_at']
```

```
        def __str__(self):
                return f"Замовлення #{self.id} від
{self.user.username}"

class OrderItem(models.Model):
        order = models.ForeignKey(Order,
related_name='items', on_delete=models.CASCADE,
verbose_name="Замовлення")
        product = models.ForeignKey(Product,
related_name='order_items', on_delete=models.PROTECT,
verbose_name="Товар")
        price = models.DecimalField(max_digits=10,
decimal_places=2, verbose_name="Ціна під час покупки")
        quantity = models.PositiveIntegerField(default=1,
verbose_name="Кількість")

        class Meta:
                verbose_name = "Елемент замовлення"
                verbose_name_plural = "Елементи
замовлення"

        def get_cost(self):
                return self.price * self.quantity

        def __str__(self):
                return f"{self.id} -
{self.product.name}"

class PaymentTransaction(models.Model):
        STATUS_CHOICES = (
                ('wait_accept', 'Очікує підтвердження'),
                ('success', 'Успішно'),
```

```
                ('error', 'Помилка'),
                ('failure', 'Невдало'),
                ('reversed', 'Повернено'),
        )

        order = models.OneToOneField(Order,
related_name='payment', on_delete=models.CASCADE,
verbose_name="Замовлення")
        liqpay_order_id =
models.CharField(max_length=255, unique=True,
verbose_name="ID замовлення в LiqPay")
        payment_id = models.CharField(max_length=255,
blank=True, null=True, verbose_name="ID платежу
(LiqPay)")
        status = models.CharField(max_length=50,
choices=STATUS_CHOICES, default='wait_accept',
verbose_name="Статус платежу")
        amount = models.DecimalField(max_digits=10,
decimal_places=2, verbose_name="Сума транзакції")
        currency = models.CharField(max_length=3,
default='UAH', verbose_name="Валюта")
        created_at =
models.DateTimeField(auto_now_add=True,
verbose_name="Створено")
        updated_at = models.DateTimeField(auto_now=True,
verbose_name="Оновлено")

        class Meta:
                verbose_name = "Транзакція"
                verbose_name_plural = "Транзакції"
                ordering = ['-created_at']
```

```
        def __str__(self):
                return f"Транзакція
{self.liqpay_order_id} - {self.status}"
```

Щоб цей код моделей запрацював і перетворився на реальні таблиці в базі даних, виконаємо кілька кроків у терміналі (командному рядку).

Django використовує механізм, який називається **міграції** (migrations). Міграції автоматично перетворюють класи з файлу models.py на SQL-запити для створення таблиць.

Покрокова інструкція, як була запущена міграція (ці команди потрібно виконувати в терміналі, перебуваючи в головній папці проєкту, де знаходиться файл manage.py, при активованому віртуальному середовищі):

Крок 1. Створення файлів міграцій

Спочатку потрібно сказати Django проаналізувати ваш файл models.py і підготувати інструкції для бази даних:

```
(шлях до папки проєкту)
python manage.py makemigrations
```

Успішний результат: В консолі бачимо повідомлення про створення міграцій (наприклад, Create model Category, Create model Product тощо).

Крок 2. Застосування міграцій до бази даних

Тепер потрібно виконати інструкції, щоб фізично створити таблиці у базі даних (за замовчуванням Django використовує SQLite файл db.sqlite3):

```
(шлях до папки проєкту)
python manage.py migrate
```

Успішний результат: Бачимо список застосованих міграцій з позначкою [OK] біля кожної.

Крок 3. Реєстрація моделей в адмін-панелі

Щоб побачити таблиці, створювати товари та переглядати замовлення через зручний веб-інтерфейс, вам потрібно зареєструвати ці

моделі у файлі admin.py (який знаходиться в тій же папці, що й models.py).

В файл admin.py і додаємо код:

Лістинг *2 – Зміст файлу admin.py*

```
from django.contrib import admin
from .models import Category, Product, Order, OrderItem, PaymentTransaction

admin.site.register(Category)
admin.site.register(Product)
admin.site.register(Order)
admin.site.register(OrderItem)
admin.site.register(PaymentTransaction)
```

Крок 4. Створення ролі адміністратора

Щоб зайти в адмін-панель, вам потрібен акаунт:

```
(шлях до папки проєкту)
python manage.py createsuperuser
```

Система попросить email (можна залишити порожнім) та пароль.

Крок 5. Запуск сервера

Тепер запускаємо локальний сервер розробки:

```
(шлях до папки проєкту)
python manage.py runserver
```

Після запуску відкриваємо браузер і переходимо за адресою http://127.0.0.1:8000/admin/. Увійдіть за допомогою створеного логіна і пароля — і бачимо повноцінну систему управління інтернет-магазином, де можна додавати категорії та товари.

3.3 Створення логіки безпечного резервування товарів

Модуль services.py, який містить логіку безпечного резервування товарів під час оформлення замовлення.

Головні архітектурні принципи, які роблять цей код надійним для високонавантаженого інтернет-магазину:

1. **transaction.atomic() (Все або нічого):** Увесь блок коду виконується як єдина транзакція. Якщо клієнт замовив процесор і материнську плату, але плати не виявилось на складі, система згенерує помилку (ValidationError). Завдяки атомарності процесор *не буде* списаний зі складу, база даних миттєво відкотить усі зміни до початкового стану.
2. **select_for_update() (Захист від Race Condition):** Це найважливіший рядок. Коли система виконує цей запит, вона фізично блокує відповідні рядки в таблиці товарів на рівні бази даних (Row-Level Locking). Якщо в цю ж мілісекунду інший клієнт натисне "Купити" на той самий останній ноутбук, його запит зачекає завершення першої транзакції. Таким чином, баланс складу ніколи не піде в "мінус".
3. **order_by('id') (Захист від Deadlocks):** Коли ми блокуємо декілька товарів одночасно, дуже важливо блокувати їх у фіксованому порядку. Якщо транзакція А спробує заблокувати Товар 1, а потім Товар 2, а транзакція Б — Товар 2, а потім Товар 1, вони можуть "зависнути", очікуючи одна одну. Сортування за ID повністю виключає цю проблему.

Лістинг *3 – Логіка безпечного резервування товарів*

```
#File services.py

import  logging
from  django.db  import  transaction
```

```
from django.core.exceptions import ValidationError
from .models import Product, Order, OrderItem

logger = logging.getLogger(__name__)

def create_order_and_reserve_products(user, cart_items):
        """
        Створює замовлення та безпечно резервує товари на складі.

        :param user: Об'єкт User, який робить замовлення.
        :param cart_items: Список словників у форматі [{'product_id': 1, 'quantity': 2}, ...]
        :return: Об'єкт Order
        """

        # Витягуємо всі ID товарів з кошика для одного запиту до БД
        product_ids = [item['product_id'] for item in cart_items]

        # Створюємо словник для швидкого пошуку кількості товару в кошику
        cart_quantities = {item['product_id']: item['quantity'] for item in cart_items}

        # Початок атомарної транзакції
        with transaction.atomic():
                # 1. БЛОКУВАННЯ РЯДКІВ (select_for_update)
                # order_by('id') критично важливий для уникнення deadlocks (взаємних блокувань),
                # якщо дві транзакції намагатимуться заблокувати однакові товари в різному порядку.
                products = Product.objects.select_for_update().filter(id__in=product_ids).order_by('id')

                # Перевіряємо, чи всі товари з кошика існують
                if len(products) != len(product_ids):
                        raise ValidationError("Один або кілька товарів з кошика не знайдено.")

                order_items_to_create = []
                total_amount = 0
```

```
                # 2. ПЕРЕВІРКА ЗАЛИШКІВ ТА ЗНЯТТЯ ЗІ
СКЛАДУ
                for product in products:
                        requested_quantity =
cart_quantities[product.id]

                        if product.stock <
requested_quantity:
                                # Якщо хоча б одного
товару не вистачає, транзакція перерветься,
                                # і жоден товар не
буде списано (завдяки transaction.atomic).
                                raise
ValidationError(f"Недостатньо товару '{product.name}' на
складі. Доступно: {product.stock}")

                        # Віднімаємо кількість зі
складу
                        product.stock -=
requested_quantity
                        product.save()

                        # Розраховуємо загальну суму
                        item_cost = product.price *
requested_quantity
                        total_amount += item_cost

                        # Підготовка даних для
OrderItem
                        order_items_to_create.append({
                                'product': product,
                                'price': product.price,
                                'quantity':
requested_quantity
                        })

                # 3. СТВОРЕННЯ ЗАМОВЛЕННЯ
                # Статус 'reserved' означає, що товари
списані зі складу, але оплата ще не підтверджена
                order = Order.objects.create(
                        user=user,
                        status='reserved',
                        total_amount=total_amount
                )

                # 4. СТВОРЕННЯ ЕЛЕМЕНТІВ ЗАМОВЛЕННЯ
(Bulk Create для оптимізації)
                order_items = [
```

```
                        OrderItem(
                                order=order,
                                product=item['product'],
                                price=item['price'],
                                quantity=item['quantity']
                        )
                        for  item  in
order_items_to_create
                ]
                OrderItem.objects.bulk_create(order_items)

                logger.info(f"Замовлення  #{order.id}
успішно  створено.  Товари  зарезервовано.")

                return  order
```

Цей фрагмент коду реалізує патерн сервісний шар (Service Layer) і відповідає за критично важливу бізнес-логіку інтернет-магазину — надійне оформлення замовлення та списання залишків без ризику продати один і той самий товар двом різним клієнтам.

Покрокова робота патерну сервісний шару:

Крок 1. Підготовка даних

На початку функція приймає об'єкт користувача та список товарів у кошику. Замість того, щоб робити окремий запит до бази даних для кожного товару в кошику, код одразу витягує всі ідентифікатори товарів у масив product_ids. Це суттєво зменшує навантаження на базу даних, готуючи систему до виконання одного загального запиту.

Крок 2. Атомарна транзакція (transaction.atomic)

Блок with transaction.atomic(): гарантує принцип "все або нічого" (ACID). Усі операції з базою даних всередині цього блоку розглядаються як єдина неподільна дія. Якщо на будь-якому етапі виникне помилка (наприклад, не вистачить товару), система автоматично скасує всі попередні зміни в цьому блоці. Це гарантує, що база даних ніколи не опиниться в частково оновленому, некоректному стані.

Крок 3. Блокування на рівні рядків (select_for_update)

Це найважливіший механізм захисту від стану гонитви (race condition). Метод select_for_update() вказує базі даних застосувати песимістичне блокування (pessimistic lock) до вибраних товарів. Якщо в цей самий момент інший покупець спробує оформити замовлення на ті ж товари, його процес буде призупинено доти, доки перша транзакція не завершиться.

Крок 4. Захист від взаємних блокувань (order_by('id'))

При одночасному блокуванні кількох записів є ризик виникнення взаємного блокування (deadlock). Наприклад, транзакція А блокує товар 1 і чекає на товар 2, а транзакція Б блокує товар 2 і чекає на товар 1. Сортування товарів за ідентифікатором (order_by('id')) гарантує, що всі паралельні транзакції завжди намагатимуться блокувати товари в однаковій послідовності, що повністю виключає можливість deadlocks.

Крок 5. Валідація залишків

У циклі система перевіряє, чи достатньо фізично товару на складі (product.stock < requested_quantity). Якщо товару менше, ніж просить клієнт, генерується виняток ValidationError. Завдяки атомарній транзакції, викидання цього винятку миттєво зупиняє процес і повертає залишки всіх попередньо перевірених товарів до їхнього початкового стану.

Крок 6. Оптимізація запитів (bulk_create)

Після успішного створення об'єкта замовлення (Order), системі потрібно створити записи для кожного товару в чеку (OrderItem). Замість того, щоб зберігати кожен рядок окремим SQL-запитом INSERT, код збирає їх у список і використовує метод bulk_create. Це дозволяє записати всі позиції чека в базу даних за один єдиний запит, що значно підвищує продуктивність системи при великих замовленнях.

Щоб запустити написаний код (models.py та services.py) і перевірити, як працює механізм резервування товарів на практиці, потрібно інтегрувати їх у ваш Django-проєкт.

Найкращий спосіб протестувати цю логіку без створення повноцінного веб-інтерфейсу (frontend) — це скористатися інтерактивною консоллю Django Shell або створити тестовий скрипт.

Функція select_for_update(), яка використовується для блокування рядків, повноцінно працює з базами даних, що підтримують блокування на рівні рядків (Row-Level Locking), наприклад**,** PostgreSQL **або** MySQL. В СУБД SQLite), select_for_update() працює без помилок, але заблокує *всю базу даних* на час транзакції, а не окремі рядки.

Крок 1: Розміщення файлів

Переконайтеся, що ви створили додаток (наприклад, shop) у вашому Django-проєкті:

```
bash
python manage.py startapp shop
```

1. Додати додаток shop до списку INSTALLED_APPS у файлі settings.py.
2. Помістіть створений раніше код моделей у shop/models.py.
3. Створіть файл shop/services.py і вставити туди код логіки резервування.

Крок 2: Створення таблиць у базі даних

Django має згенерувати SQL-команди на основі ваших моделей і застосувати їх. Виконати в терміналі:

```
(шлях до робочої папки)
python manage.py makemigrations shop
python manage.py migrate
```

Крок 3: Тестування логіки через Django Shell

Тепер запустимо консоль Django, яка дозволяє взаємодіяти з базою даних використовуючи наш код:

```
(шлях до робочої папки)
python manage.py shell
```

Після відкриття консолі (з'явиться >>>), треба скопіювати та вставити наступний скрипт. Він створить тестового користувача, категорію, товари, а потім викличе сервіс (Лістинг 4):

Лістинг *4 – Скрипт для тестування логіки надійного замовлення*

```
from django.contrib.auth.models import User
from shop.models import Category, Product, Order
from shop.services import create_order_and_reserve_products
from decimal import Decimal

# 1. Створюємо тестового користувача
user, created = User.objects.get_or_create(username='test_buyer', email='buyer@test.com')

# 2. Створюємо категорію
category, created = Category.objects.get_or_create(
        name='Ноутбуки',
        slug='laptops'
)

# 3. Створюємо тестові товари (задаємо кількість на складі)
product1, created = Product.objects.get_or_create(
        category=category,
        name='MacBook Pro 14',
        slug='macbook-pro-14',
        defaults={'price': Decimal('85000.00'), 'stock': 5}
```

```
)

product2, created = Product.objects.get_or_create(
        category=category,
        name='Миша Apple Magic Mouse',
        slug='magic-mouse',
        defaults={'price': Decimal('3500.00'),
'stock': 10}
)
print(f"Початковий залишок: MacBook -
{product1.stock} шт., Миша - {product2.stock} шт.")

# 4. Формуємо віртуальний "кошик"
cart_items = [
        {'product_id': product1.id, 'quantity': 1},
# Купуємо 1 макбук
        {'product_id': product2.id, 'quantity': 2}
# Купуємо 2 мишки
]

# 5. ЗАПУСКАЄМО НАШУ ЛОГІКУ
try:
        order =
create_order_and_reserve_products(user=user,
cart_items=cart_items)
        print(f"\n✅ Успіх! Замовлення #{order.id}
створено.")
        print(f"Сума замовлення: {order.total_amount}
грн. Статус: {order.status}")
except Exception as e:
```

```
        print(f"\n❌ Помилка: {e}")

# 6. Перевіряємо залишки на складі після резервування
product1.refresh_from_db()
product2.refresh_from_db()
print(f"\nЗалишок після замовлення: MacBook - {product1.stock} шт., Миша - {product2.stock} шт.")
```

Щоб побачити, як працює захист (відкат транзакції), після попереднього коду виконати нове замовлення, де товару потрібно більше, ніж є:

Лістинг 5 – Скрипт для перевірки роботи надійної логіки продажу

```
huge_cart = [{'product_id': product1.id, 'quantity': 10}]
# Хочемо 10 макбуків, а лишилось 4

try:
        order_fail = create_order_and_reserve_products(user=user, cart_items=huge_cart)
except Exception as e:
        print(f"Очікувана помилка спрацювала: {e}")
```

3.4 Створення інтерфейсів користувачів (frontend)

3.4.1 Інтерфейс адміністратора

Фреймворк Django має вбудований розвинутий інтерфейс адміністратора з широкою функціональністю. Інтерфейс адміністратора (Admin panel) у Django — це потужна вбудована підсистема, яка автоматично

генерується на основі моделей даних (ORM) і дозволяє керувати контентом сайту без необхідності писати додатковий код для бекенду.

Для проєкту «Веб-сайт з продажу комп'ютерів» інтерфейс адміністратора має наступний вигляд та функціонал:

1. Авторизація та головна панель (Dashboard)

Вхід: Для доступу до панелі адміністратора в Django при запущеному сервері відкрити сторінку за адресо. *http://127.0.0.1:8000/admin* .

- Головна сторінка: Відображає список усіх зареєстрованих додатків (apps) та їхніх моделей. Наприклад: додаток «Каталог» (товари, категорії), додаток «Замовлення», додаток «Користувачі».



Рисунок *11 – Головна панель адміністратора*

2. Управління каталогом (Товари та Категорії)

Це основний розділ для контент-менеджера магазину.

- Категорії (Categories): Можливість створювати та редагувати категорії (наприклад, "Ноутбуки", "ПК у зборі", "Комплектуючі", "Периферія").
- Комп'ютери/Товари (Products): При додаванні або редагуванні товару адміністратор бачить форму з такими полями:

- *Основна інформація:* Назва (наприклад, "Ігровий ПК Asus ROG"), артикул (SKU), опис, ціна.
- *Характеристики:* Процесор, обсяг ОЗП, відеокарта, обсяг накопичувача (SSD/HDD). У Django Admin це можна реалізувати через пов'язані таблиці або JSON-поля.
- Зручності: У списку товарів адміністратор бачить таблицю з колонками (ID, Назва, Ціна, Залишок). Є панель фільтрів (праворуч) для відбору товарів за категорією чи брендом, а також рядок пошуку за назвою.

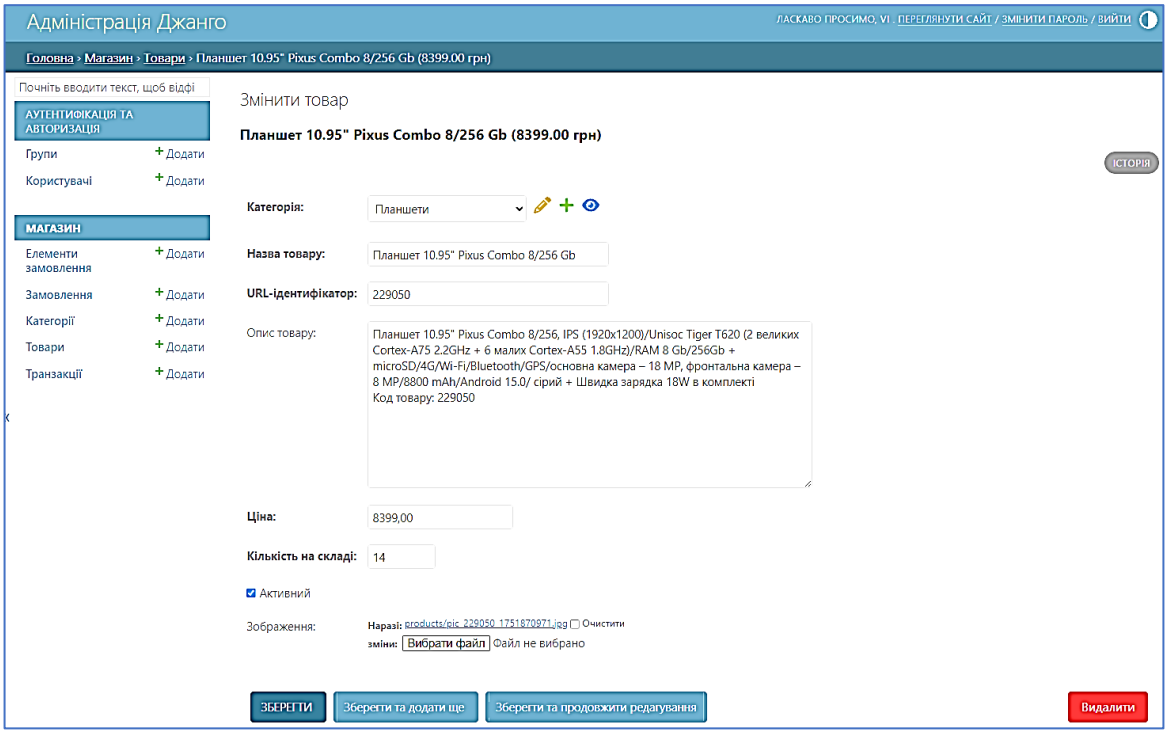


Рисунок *12– Панель управління каталогом*

3. Управління замовленнями (Orders)

Критично важливий розділ для обробки покупок.

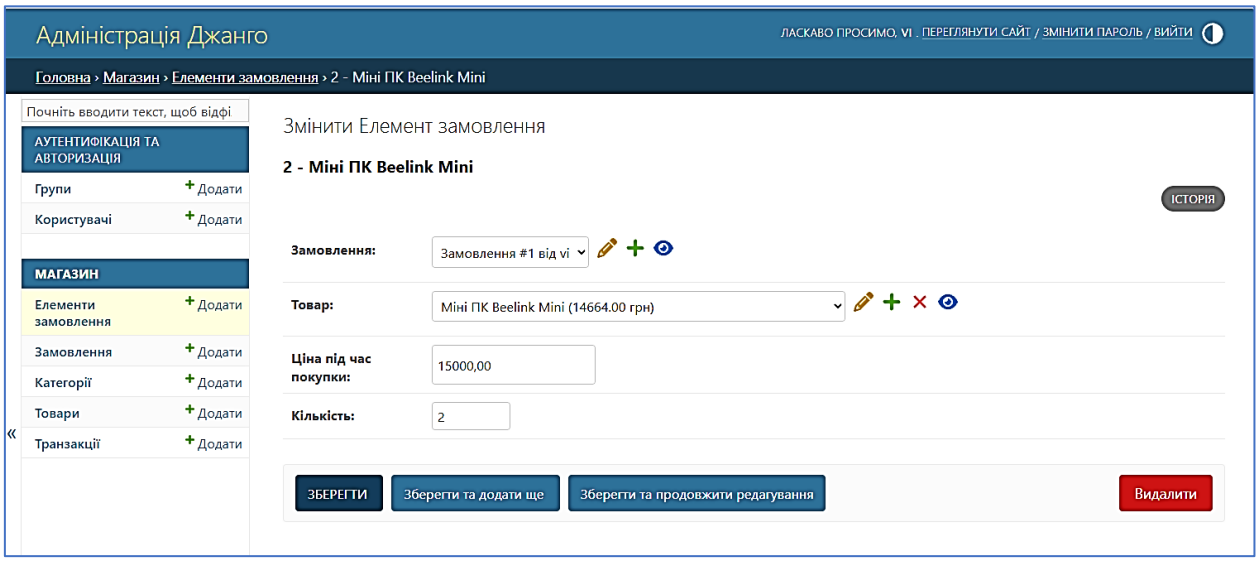


Рисунок *13 – Панель управління замовленням*

- Список замовлень: Відображає номер замовлення, дату створення, email клієнта, загальну суму та поточний статус (наприклад: "Нове", "Оплачено", "Відправлено", "Виконано", "Скасовано").
- Деталі замовлення: При переході в конкретне замовлення адміністратор бачить:
  - Дані покупця (ПІБ, телефон, адреса доставки).
  - Перелік замовлених комп'ютерів (через зв'язок *Inline* — товари відображаються прямо на сторінці замовлення).
- Інтеграція з платіжною системою (LiqPay): В адмінці можна переглядати статус транзакції (наприклад, payment_id від LiqPay, статус успішності платежу), щоб менеджер міг переконатися, що гроші зараховано, перш ніж відправляти товар.

4. Управління користувачами (Користувачі та групи)

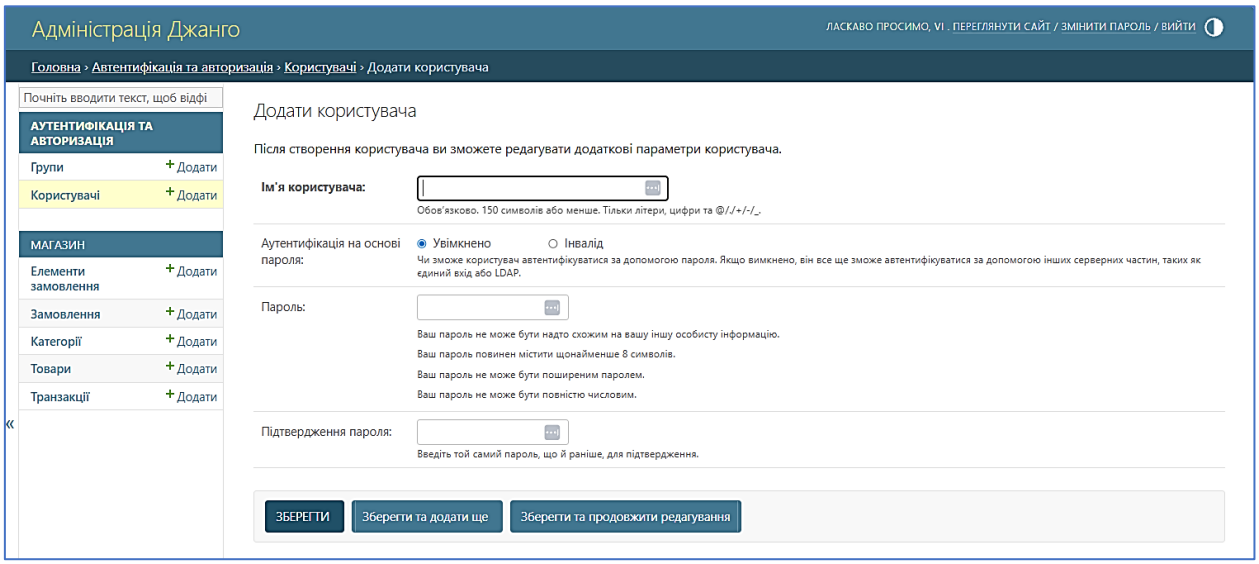


Рисунок *14 – Панель для додавання користувачів*

Використовується стандартна модель User з Django:

- Перегляд зареєстрованих клієнтів магазину.
- Призначення прав доступу (наприклад, можна створити групу "Менеджери", які матимуть доступ лише до замовлень, але не зможуть видаляти товари чи міняти налаштування сайту).

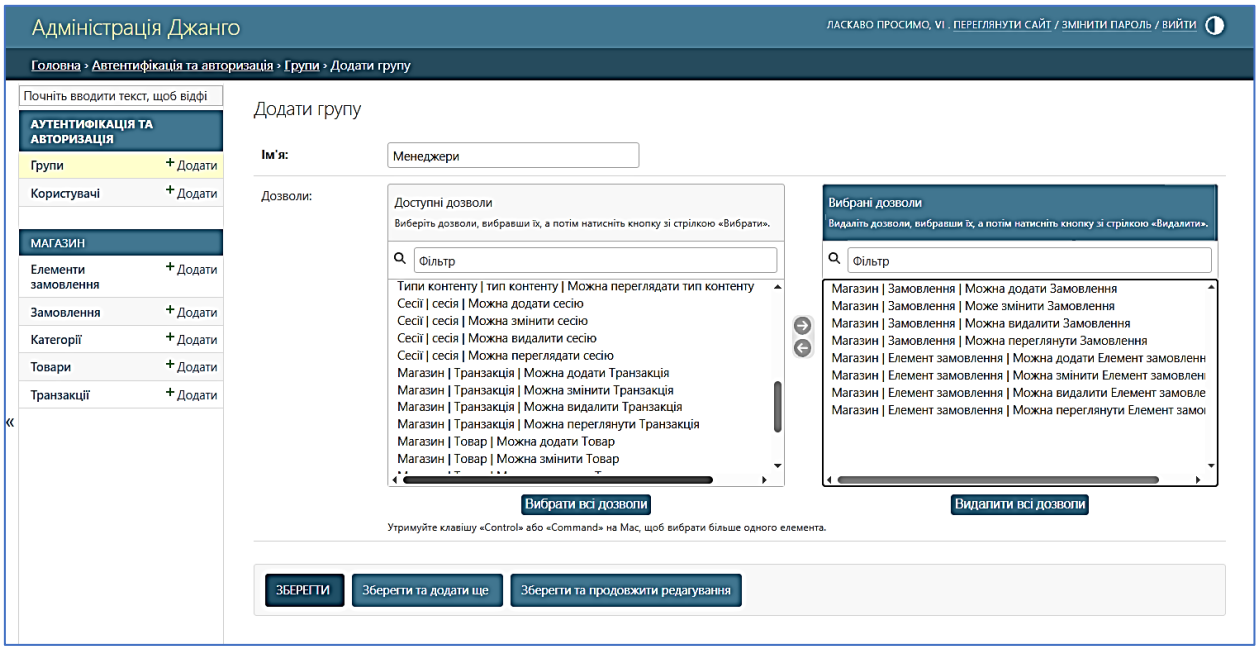


Рисунок *15 – Панель керування групами користувачів*

Загалом, інтерфейс адміністратора Django дозволяє власнику інтернет-магазину комп'ютерів повністю контролювати весь життєвий цикл продажу: від наповнення вітрини до контролю залишків, обробки платежів через LiqPay та закриття замовлень.

3.4.2 Інтерфейси користувачів

У контексті архітектурного патерну MVT (Model- View - Template) фреймворк Django, розроблений інтерфейс виконує роль шаблону (Template). Візуальний макет інтегровано з бекенд-частиною за допомогою механізму шаблонізації Django (Django Template Language), що забезпечує серверний рендеринг (Server-Side Rendering) та динамічне доповнення сторінок даними, які виявляються з реляційної бази даних.

Для створення інтерфейсу користувачів використаний Bootstrap — популярний і зручний набір готових стилів.

Він дає змогу:

- Перетворити товари на сучасні картки.
- Розмістити їх у вигляді акуратної сітки (наприклад, по 3-4 товари в ряд).
- Додати навігаційне меню (шапку сайту) зверху.
- Зробити так, щоб сайт добре виглядав і на комп'ютері, і на мобільному телефоні.

Створимо один базовий шаблон **base.html**, а всі інші сторінки будуть наслідувати з нього дизайн (Лістинг 6).

Лістинг 6 – Базовий шаблон для сторінок

```
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Наш Комп'ютерний Магазин</title>
        <!-- Підключаємо CSS Bootstrap -->
```

```
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css
/bootstrap.min.css" rel="stylesheet">
    </head>
    <body class="bg-light">

        <!-- Навігаційне меню (Шапка) -->
        <nav class="navbar navbar-expand-lg navbar-dark bg-
dark mb-4">
            <div class="container">
                <a class="navbar-brand" href="{% url
'product_list' %}">💻 ComputerShop</a>
            </div>
        </nav>

        <!-- Головний контейнер, куди буде вставлятися
контент інших сторінок -->
        <div class="container">
            {% block content %}
            <!-- Контент сторінок буде тут -->
            {% endblock %}
        </div>

        <!-- Підключаємо JS Bootstrap -->
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/b
ootstrap.bundle.min.js"></script>
    </body>
    </html>
```

На основі цього шаблону створимо сайт каталогу товарів **product_list.html** :

Лістинг 7– Створення сайту каталогу товарів

```
    {% extends 'shop/base.html' %}

    {% block content %}
        <h1 class="mb-4">Наш каталог комп'ютерної
техніки</h1>

        <div class="row">
            <!-- ЛІВА КОЛОНКА: Категорії (займає 3
колонки) -->
            <div class="col-md-3 mb-4">
                <div class="card shadow-sm">
                    <div class="card-header bg-primary
text-white fw-bold">
```

```
                        Категорії
                    </div>
                    <div class="list-group list-group-
flush">
                        <!-- Посилання на "Всі товари"
-->
                        <a href="{% url 'product_list'
%}" class="list-group-item list-group-item-action {% if
not request.GET.category %}active{% endif %}">
                            Усі товари
                        </a>

                        <!-- Виводимо список категорій
з бази даних -->
                        {% for category in categories
%}
                            <a href="?category={{
category.id }}" class="list-group-item list-group-item-
action {% if request.GET.category ==
category.id|stringformat:'s' %}active{% endif %}">
                                {{ category.name }}
                            </a>
                        {% endfor %}
                    </div>
                </div>
            </div>

            <!-- ПРАВА КОЛОНКА: Товари (займає 9
колонок) -->
            <div class="col-md-9">
                <div class="row">
                    {% for product in products %}
                        <!-- Змінили col-md-4 (по 3 в
ряд) на col-md-4, щоб вміщувалось красиво в меншому
блоці -->
                        <div class="col-md-4 mb-4">
                            <div class="card h-100
shadow-sm">
                                <a href="{% url
'product_detail' pk=product.pk %}">
                                    {% if product.image
%}
                                        <img src="{{
product.image.url }}" class="card-img-top" alt="{{
```

```
product.name }}" style="height: 200px; object-fit:
contain; padding: 10px;">
                                    {% endif %}
                                </a>
                                <div class="card-body
d-flex flex-column">
                                    <h5 class="card-
title">
                                        <a href="{% url
'product_detail' pk=product.pk %}" class="text-
decoration-none text-dark">
                                            {{
product.name }}
                                        </a>
                                    </h5>
                                    <p class="card-text
text-danger fs-5 fw-bold mt-auto">{{ product.price }}
грн</p>
                                    <a href="{% url
'product_detail' pk=product.pk %}" class="btn btn-
primary w-100">Детальніше</a>
                                </div>
                            </div>
                        </div>
                    {% empty %}
                        <!-- Повідомлення, якщо в
категорії ще немає товарів -->
                        <div class="col-12">
                            <div class="alert alert-
info">У цій категорії поки немає товарів.</div>
                        </div>
                    {% endfor %}
                </div>
            </div>
        </div>
    {% endblock %}
```

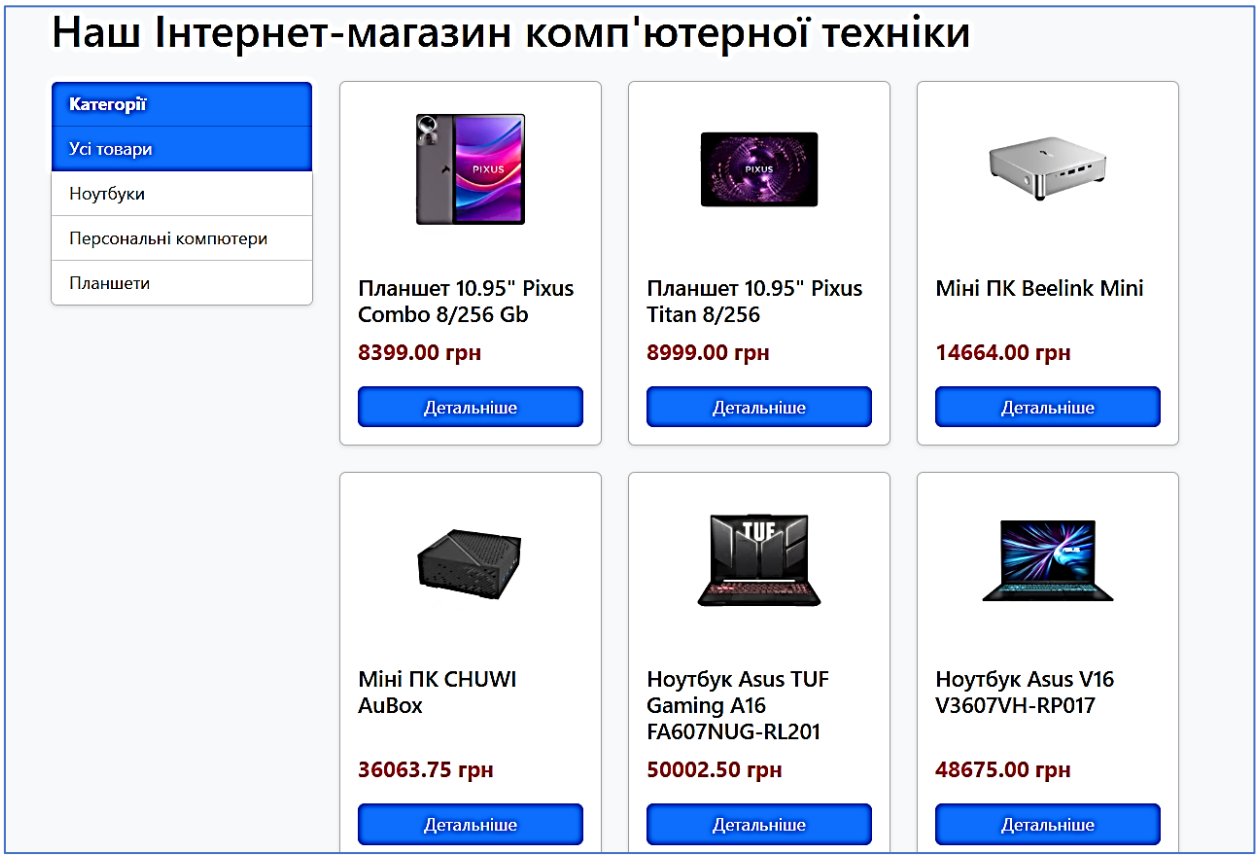


Рисунок *16 – Сторінка каталогу товарів*

Створимо сторінку деталей товару з кнопкою повернення до каталогу (Лістинг 8).

Лістинг *8 – Створення сайту каталогу товарів*

```
    {% extends 'shop/base.html' %}

    {% block content %}
    <div class="row mt-4 mb-5">

        <!-- Ліва колонка: Фотографія -->
        <div class="col-md-6 mb-4 text-center">
            <div class="card shadow-sm p-3 h-100 d-flex
align-items-center justify-content-center">
                {% if product.image %}
                    <img src="{{ product.image.url }}"
class="img-fluid" alt="{{ product.name }}" style="max-
height: 450px; object-fit: contain;">
                {% else %}
                    <div class="p-5 text-muted">Фото
відсутнє</div>
```

```
                {% endif %}
            </div>
        </div>

        <!-- Права колонка: Деталі товару -->
        <div class="col-md-6">
            <h1    class="display-5    fw-bold    mb-3">{{
product.name }}</h1>

            <div class="mb-4">
                <span  class="text-danger  fs-1  fw-bold">{{
product.price }}</span>
                <span class="fs-4 text-muted">грн</span>
            </div>

            <!-- Якщо у вашій моделі є поле description
(опис), воно виведеться тут -->
            {% if product.description %}
                <div class="mb-4">
                    <h5             class="text-muted">Опис
товару:</h5>
                    <p  class="fs-5">{{  product.description
}}</p>
                </div>
            {% endif %}

            <hr class="my-4">

            <!-- Кнопки дій -->
            <div class="d-grid gap-3 col-lg-8">
                <!-- Кнопка "Додати в кошик" (поки що просто
візуальна, функціонал зробимо пізніше) -->
                <button   class="btn   btn-success   btn-lg"
type="button">
                    Додати в кошик 🛒
                </button>

                <!-- Кнопка повернення до каталогу -->
                <a   href="{%   url   'product_list'   %}"
class="btn btn-outline-secondary btn-lg">
                    ← Повернутися до каталогу
                </a>
            </div>
        </div>

    </div>
    {% endblock %}
```

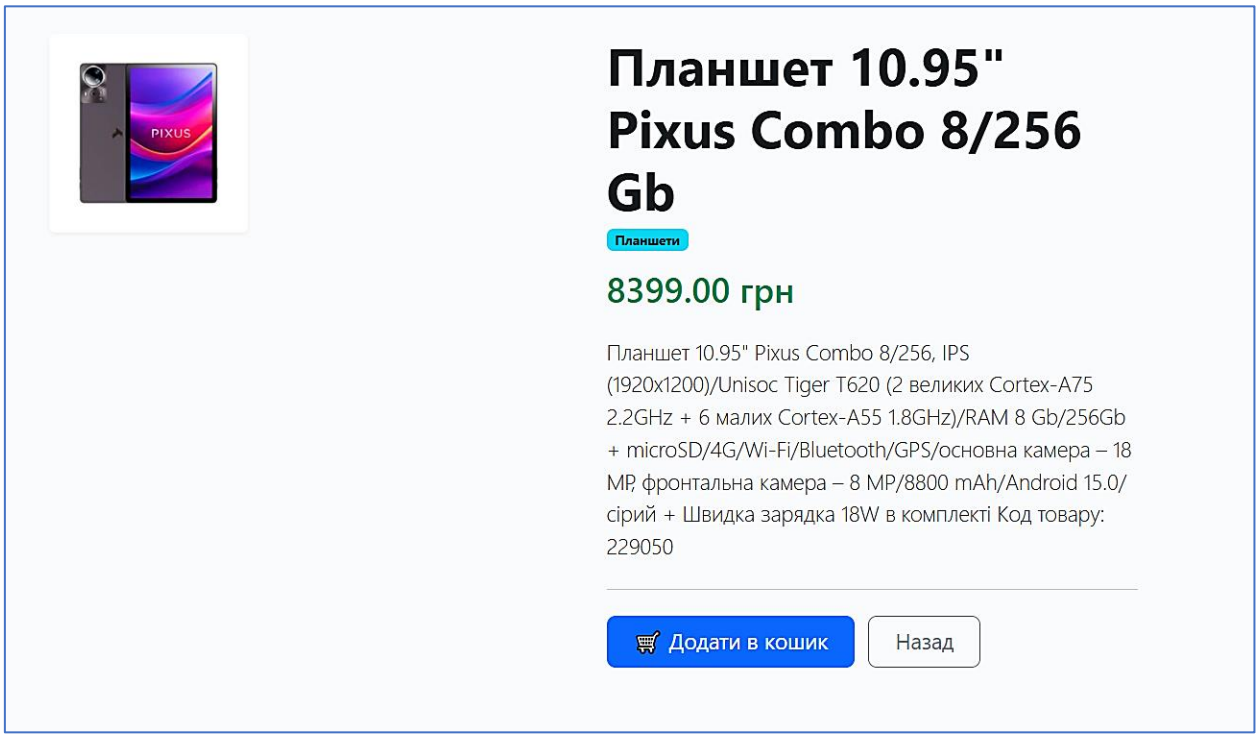


Рисунок *16 – Сторінка деталей товару*

Створимо логіку кошика окремо від файлу представлення views.py. Це файл **cart.py** (Лістинг 9).

Лістинг *9 – Створення логіки кошика*

```
from decimal import Decimal
from django.conf import settings
from .models import Product

class Cart:
    def __init__(self, request):
        """Ініціалізація кошика"""
        self.session = request.session
        # Намагаємося отримати кошик з поточної сесії
        cart = self.session.get('cart')
        if not cart:
```

```
                # Якщо кошика немає, створюємо порожній
словник
                cart = self.session['cart'] = {}
            self.cart = cart

        def add(self, product, quantity=1):
            """Додавання товару в кошик або оновлення його
кількості"""
            product_id = str(product.id)
            if product_id not in self.cart:
                self.cart[product_id]  =  {'quantity':  0,
'price': str(product.price)}

            self.cart[product_id]['quantity'] += quantity
            self.save()

        def save(self):
            """Позначаємо сесію як змінену, щоб Django її
зберіг"""
            self.session.modified = True

        def __iter__(self):
            """Перебираємо товари в кошику і отримуємо їх з
бази даних"""
            product_ids = self.cart.keys()
            products                                        =
Product.objects.filter(id__in=product_ids)

            cart = self.cart.copy()
            for product in products:
                cart[str(product.id)]['product'] = product
```

```
            for item in cart.values():
                item['price'] = Decimal(item['price'])
                item['total_price']   =   item['price']   *
item['quantity']
                yield item

        def get_total_price(self):
            """Підрахунок  загальної  вартості  товарів  у
кошику"""
            return       sum(Decimal(item['price'])       *
item['quantity'] for item in self.cart.values())
```

Тепер створимо сторінку кошика (Лістинг 10).

Лістинг *10 – Створення сторінки кошика* ***cart_detail.html***

```
{% extends "shop/base.html" %}
{% block title %}Ваш кошик{% endblock %}

{% block content %}
<div class="container mt-4">
    <h1 class="mb-4">Ваш кошик</h1>

    {% if cart %}
        <div class="table-responsive">
            <table class="table table-bordered align-
middle">
                <thead class="table-light">
                    <tr>
                        <th>Товар</th>
                        <th>Кількість</th>
                        <th>Ціна</th>
                        <th>Загалом</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in cart %}
```

```
                                {% with product=item.product %}
                                <tr>
                                    <td>{{ product.name }}</td>
                                    <td>{{ item.quantity }}
шт.</td>
                                    <td>{{ item.price }}
грн</td>
                                    <td>{{ item.total_price }}
грн</td>
                                </tr>
                                {% endwith %}
                            {% endfor %}
                        </tbody>
                        <tfoot class="table-light fw-bold">
                            <tr>
                                <td colspan="3" class="text-
end">Загальна вартість:</td>
                                <td>{{ cart.get_total_price }}
грн</td>
                            </tr>
                        </tfoot>
                    </table>
                </div>

                <div class="d-flex justify-content-between mt-
4">
                    <a href="{% url 'shop:product_list' %}"
class="btn btn-outline-secondary">Продовжити покупки</a>
                    <!-- Кнопка оформлення замовлення поки що
нікуди не веде, зробимо це пізніше -->
                    <a href="#" class="btn btn-
success">Оформити замовлення</a>
                </div>
            {% else %}
                <div class="alert alert-info" role="alert">
                    Ваш кошик порожній. <a href="{% url
'shop:product_list' %}" class="alert-link">Перейти до
каталогу товарів</a>.
                </div>
            {% endif %}
```

```
    </div>
    {% endblock %}
```

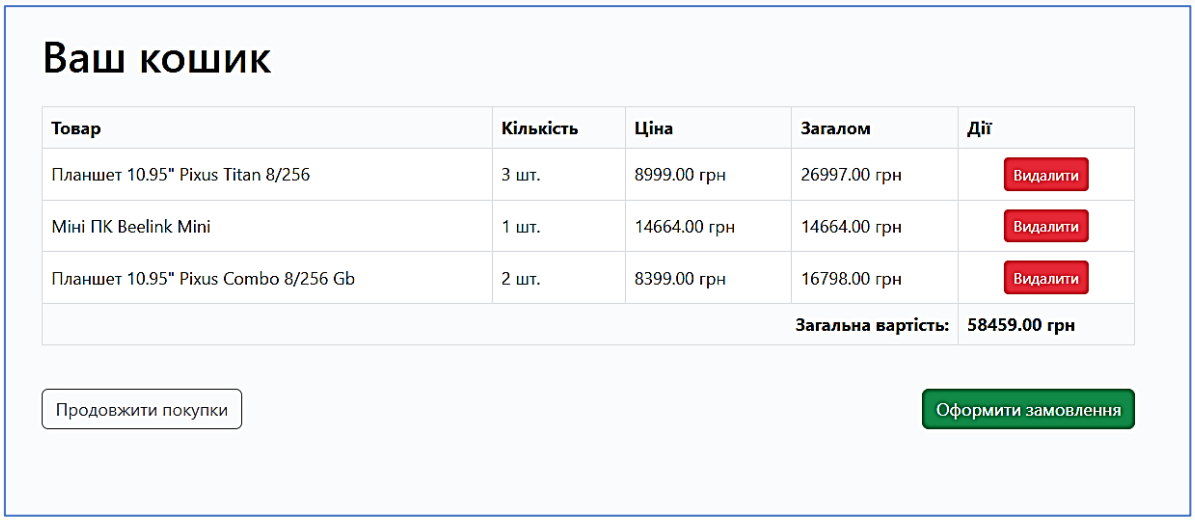


Рисунок *16 – Сторінка кошику*

3.5 Підключення платіжної системи LiqPay

Підключення платіжної системи LiqPay (від ПриватБанку) для інтернет-магазину комп'ютерної техніки — це стандартний процес, який складається з організаційної та технічної частин.

Покрокова інструкція, як це зробити:

Крок 1. Підготовка сайту (Вимоги LiqPay)

Перед тим як подавати заявку, ваш інтернет-магазин має відповідати вимогам безпеки та законодавства. Модератори LiqPay перевірятимуть сайт, тому на ньому обов'язково мають бути:

- Товари з цінами: На сайті вже мають бути додані комп'ютери, комплектуючі тощо, з актуальними цінами та можливістю додати їх у кошик.
- Контактна інформація: Номер телефону, email, адреса (фізична або юридична).

- Договір публічної оферти (Умови використання): Опис умов продажу.
- Політика повернення та обміну: Оскільки ви продаєте техніку, обов'язково чітко пропишіть умови гарантії та повернення товару протягом 14 днів згідно із законом.
- Умови оплати та доставки: Окремі сторінки або розділи з цією інформацією.

Крок 2. Реєстрація та створення компанії в LiqPay

1. Зайдіть на офіційний сайт LiqPay та авторизуйтесь (зазвичай за допомогою фінансового номера телефону або Приват24).

2. Перейдіть у розділ «Бізнес» і натисніть «Створити компанію» (або «Підключити інтернет-магазин»).

3. Заповніть дані про ваш магазин:

   - Назва, адреса сайту (URL).
   - Категорія товарів (Електроніка / Комп'ютерна техніка).
   - Реквізити для зарахування коштів (зазвичай це IBAN вашого рахунку ФОП або ТОВ. LiqPay працює з рахунками будь-яких українських банків, не лише ПриватБанку, але для рахунків у ПриватБанку зарахування відбувається швидше і часто без додаткових комісій).

Крок 3. Отримання API-ключів

Після створення компанії в кабінеті LiqPay вам будуть доступні ключі для інтеграції.

1. Знайдіть у налаштуваннях компанії розділ API (або «Ключі API»).
2. Скопіюйте Public Key (публічний ключ) та Private Key (приватний ключ).
3. Нікому не передавайте Private Key, крім вашого розробника (якщо він є).
4. Public Key (публічний ключ) та Private Key (приватний ключ) внесіть у файл core\settings.py:

```
LIQPAY_PUBLIC_KEY = 'ваш_публічний_ключ'
```

```
LIQPAY_PRIVATE_KEY = 'ваш_приватний_ключ'
```

3.6 Висновки з розділу 3

1. В цьому розділі розглянуто створення проєкту веб-застосунку у фреймворку Django та представлена його реалізація.

2. Реалізовані моделі, що описують предметну область та через Django ORM створена база даних в СУБД SQLite.

3. Представлена розробка та створення інтерфейсів адміністратора та користувачів веб-сайту з продажу комп’ютерів.

4. Показано, як підключити створений інтернет-магазин до платіжної системи Приват банку LiqPay/

ВИСНОВКИ

1. Розробка веб-застосунку з продажу комп’ютерної техніки є актуальною задачею. Під час аналізу технологій для створення таких додатків було складено список платформ, які відповідають найсучаснішим стандартам. Також було визначено набір технологій та інструментів розробки, їх продуктивність, рівень інтеграції та зручність використання.
2. Проведено дослідження та аналіз сучасних аналогів та методів створення магазинів електронної комерції.
3. В результаті виконання кваліфікаційної роботи було створено веб-застосунок з продажу комп’ютерів для компаній, що спеціалізується на продажу комп’ютерної техніки. Для реалізації використано мову Python, фрейворк Django, патерн проєктування MVT та вбудована система ORM.
4. Проєкт передбачав планування структури, налаштування середовища розробки, створення базової файлової архітектури та маршрутизації. Дані зберігаються у базі даних СУБД SQLite, створеної на основі моделей з допомогою ORM.
5. Для організації доступу до конфіденційної інформації реалізовано авторизацію адміністратора через спеціальну вбудовану в Django панель адміністратора. Це дозволяє захистити перегляд та керування інтернет-магазином.
6. Інтерфейс веб-сайту реалізовано з використанням CSS, що забезпечує адаптивний дизайн та зручну взаємодію з користувачем. Всі сторінки структуровані та відповідають вимогам до сучасного веб-дизайну.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Melé A. (2022). Django 4 By Example. Packt Publishing.Forth Editing. 456 p.

2. Vincent, W. S. (2022). Django for Professionals: Production-ready web development. Leanpub. 454 p/

3. Greenfeld, D. R., & Greenfeld, A. R. (2020). Two Scoops of Django 3.x: Best Practices for the Django Web Framework. Two Scoops Press. 568 p.

4. Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.765 p.

5. Django Software Foundation. The Django documentation. URL: https://docs.djangoproject.com/ (дата звернення 20.05.2026).

6. LiqPay API Documentation. Офіційна документація для розробників. URL: https://www.liqpay.ua/documentation/(дата звернення 20.05.2026).

7. LiqPay Python SDK. Офіційний репозиторій бібліотеки на GitHub. URL: https://github.com/liqpay/sdk-python (дата звернення 20.05.2026).

8. Celery Project. Celery - Distributed Task Queue.URL: https://docs.celeryq.dev/ Необхідно для розуміння налаштування фонових завдань (зняття неоплаченого резерву через 15 хвилин).

9. Microsoft Architecture Center. E-commerce reference architecture. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/ecommerce/ (дата звернення 20.05.2026).

10. AIN.UA. Аналітика українського ринку E-commerce. URL: https://ain.ua/ (дата звернення 20.05.2026).

11. Retailers.ua. Новини роздрібної торгівлі та e-commerce. URL: https://retailers.ua/ (дата звернення 20.05.2026).

12.Statista. E-commerce worldwide - Statistics & Facts. URL: https://www.statista.com/markets/413/e-commerce/ (дата звернення 20.05.2026).

13. Django Documentation. URL: https://docs.djangoproject.com/ (дата звернення 20.05.2026).

14. MDN Web Docs. URL: https://developer.mozilla.org/ (дата звернення 20.05.2026).

15. Python Documentation. URL: https://docs.python.org/ (дата звернення 20.05.2026).

16. Іваненко В. Розробка веб-застосунку з продажу комп'ютерів. Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, університетська конференція «МОЛОДА НАУКА-2026» Том III «Секції Інженерного навчально-наукового інституту ім. Ю. М. Потебні Запорізького національного університету» : м. Запоріжжя, 20–24 квітня 2026 р. Запоріжжя, 2026. С. 95–96.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Іваненко Вікторія Сергіївна , студентка 4 курсу, форми навчання заочної, Інженерного навчально-наукового інституту ім. Ю. М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми «Програмне забезпечення систем», адреса електронної пошти svetka_ivanenko@ukr.net , — підтверджую, що написана мною кваліфікаційна робота на тему «Розробка веб-застосунку з продажу комп'ютерів» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

-згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 20.05.2026 _____ Іваненко Вікторія Сергіївна
(студентка)

Дата 20.05.2026 _____Безверхий Анатолій Ігорович
(керівник)