Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 «Інженерія програмного забезпечення»

На тему: *Розробка веб-орієнтованої інформаційної системи маркетплейсу (електроніки) з застосуванням JAMstack підходу*

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-21-2
_____ *Ковтун М.О.*

Керівник кваліфікаційної роботи _____ */ Гриценко А. М. /*

Завідувач кафедри _____ */ Стрюк А. М. /*

Кривий Ріг
2025

Криворізький національний університет
Факультет: *Інформаційних технологій*
Кафедра: *Моделювання та програмного забезпечення*
Освітньо-кваліфікаційний рівень: *бакалавр*
Спеціальність: *121 "Інженерія програмного забезпечення"*

ЗАТВЕРДЖУЮ
Зав. кафедри
_____*Стрюк А. М.*
«___»_____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу
студенту групи ІПЗ-21-2 Ковтуну Максиму Олександровичу

1. Тема: *Розробка веб-орієнтованої інформаційної системи маркетплейсу (електроніки) з застосуванням JAMstack підходу*
затверджено наказом по КНУ №119 від «10» квітня 2025 р.
2. Термін подання студентом закінченого проекту «16» червня 2025 р.
3. Вихідні дані по роботі: *Пояснювальна записка: 64 сторінок, 23 рисунків, 1 додаток, 21 використаних у роботі джерел*
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Огляд предметної області в галузі електронної комерції. Проектування та розробка інформаційної системи маркетплейсу.
5. Перелік графічного демонстраційного матеріалу: Актуальність предметної області. Приклади існуючих програмних рішень. Архітектура системи. Структура бази даних. Приклади роботи розробленої системи (інтерфейс користувача програми). Результати тестування.
Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	*Формулювання мети та задач роботи*	*13.02.2025-20.02.2025*
2	*Аналіз інформаційних джерел*	*21.02.2025-09.03.2025*
3	*Визначення вимог до програмного забезпечення*	*10.03.2025-18.03.2025*
4	*Розробка архітектури програмного продукту*	*19.03.2025-23.03.2025*
5	*Розробка інтерфейсу*	*24.03.2025-31.03.2025*
6	*Розробка бази даних*	*01.04.2025-14.04.2025*
7	*Розробка програмного забезпечення*	*15.04.2025-13.05.2025*
8	*Розгортання та тестування програмного забезпечення*	*14.05.2025-20.05.2025*
9	*Оформлення пояснювальної записки*	*21.05.2025-12.06.2025*
10	*Розробка демонстраційних матеріалів*	*13.06.2025-17.06.2025*

Дата видачі завдання: «_____»_____ 2025 р.

Студент _____ Ковтун М.О.

Керівник роботи _____ Гриценко А. М.

РЕФЕРАТ

JAMSTACK, HEADLESS CMS, STATIC SITE GENERATION, INCREMENTAL STATIC REGENERATION, NEXT.JS, STRAPI, SERVERLESS, CDN, МУЛЬТИМОВНІСТЬ

Пояснювальна записка:64 с., 8 табл., 32 рис., 1 дод., 24 джерела.

Метою роботи є розробка високопродуктивної, безпечної та масштабованої JAMstack-системи маркетплейсу аксесуарів до комп'ютерної техніки.

Проведено огляд технологій побудови e-commerce-рішень — від монолітних WordPress/WooCommerce і SaaS-платформи Shopify до headless-фреймворків Snipcart, Commerce Layer і open-source Medusa.js.

Спроєктовано логічну та фізичну архітектуру з чітким розділенням даних і представлення, мультимовністю (укр. / англ.) та адаптивністю під різні пристрої. Розроблено ER-модель, прототип інтерфейсу у Figma і компонентний фронтенд на Next.js із Static Site Generation та Incremental Static Regeneration, а також API-роути через REST/GraphQL.

Реалізовано ключовий функціонал: каталог із фільтрацією і пошуком, картки та детальний перегляд товару з галереями й відгуками, кошик і адмін-панель Strapi. Налаштовано CI/CD-конвеєр з автоматичним білдом і деплоєм при кожному пуші, а також webhook-тригер з CMS для перегенерації сторінок.

Проведено комплексне тестування — перевірку основних юзкейсів, адаптивності та аудит продуктивності через Lighthouse (Performance ≥ 90). Результати підтвердили відповідність рішення бізнес-вимогам і технічним стандартам сучасних e-commerce-платформ.

Отриманий результат готовий до комерційного впровадження з можливістю розширення функціоналу (оплата, аналітика, персоналізація) та адаптації до інших ринків.

ABSTRACT

JAMSTACK, HEADLESS CMS, STATIC SITE GENERATION, INCREMENTAL STATIC REGENERATION, NEXT.JS, STRAPI, SERVERLESS, CDN, MULTILINGUALISM

Explanatory note:64 p., 8 tables, 32 figures, 1 appendix, 24 sources.

The purpose of the work is to develop a high-performance, secure and scalable JAMstack marketplace system for computer accessories.

A review of e-commerce solution building technologies was conducted - from monolithic WordPress/WooCommerce and SaaS platform Shopify to headless frameworks Snipcart, Commerce Layer and open-source Medusa.js.

A logical and physical architecture was designed with a clear separation of data and presentation, multilingualism (Ukrainian / English) and adaptability to different devices. Developed an ER model, a prototype interface in Figma and a component frontend on Next.js with Static Site Generation and Incremental Static Regeneration, as well as API routes via REST/GraphQL.

Implemented key functionality: catalog with filtering and search, cards and detailed product view with galleries and reviews, shopping cart and Strapi admin panel. Configured a CI/CD pipeline with automatic build and deployment for each push, as well as a webhook trigger with CMS for page regeneration.

Comprehensive testing was conducted - checking the main use cases, adaptability and performance audit via Lighthouse (Performance ≥ 90). The results confirmed that the solution meets business requirements and technical standards of modern e-commerce platforms.

The result is ready for commercial implementation with the possibility of expanding the functionality (payment, analytics, personalization) and adapting to other markets.

ЗМІСТ

ВСТУП

Сучасне суспільство дедалі глибше інтегрує цифрові технології у всі сфери життєдіяльності, і торгівля не є винятком. Розвиток електронної комерції відбувається стрімкими темпами, що обумовлено як глобальними тенденціями цифровізації, так і зміною споживчих звичок. Все більше користувачів віддають перевагу онлайн-покупкам, оскільки це зручно, швидко та дозволяє отримати доступ до широкого асортименту товарів незалежно від місця перебування. У цьому контексті маркетплейси — електронні торгові платформи, які об’єднують численних продавців і покупців — стали ключовими гравцями на ринку електронної комерції.

Особливої популярності набувають вузькоспеціалізовані маркетплейси, які орієнтовані на окремі категорії товарів. Такий підхід дозволяє не лише краще задовольнити потреби цільової аудиторії, але й забезпечити вищий рівень зручності у користуванні та якості обслуговування. У цьому контексті розробка маркетплейсу, орієнтованого на продаж аксесуарів до комп’ютерної техніки, є доцільною та перспективною. Ця категорія товарів користується стабільним попитом серед широкого кола користувачів — від офісних працівників до геймерів та ентузіастів техніки.

Зі зростанням вимог до продуктивності веб-додатків, швидкості завантаження сторінок, безпеки та масштабованості особливої актуальності набувають сучасні архітектурні підходи до розробки. Одним із таких є JAMstack — новітній підхід до побудови веб-застосунків, який ґрунтується на використанні JavaScript, API та статичного генератора контенту (Markup). На відміну від традиційної серверно-орієнтованої архітектури, JAMstack забезпечує кращу продуктивність, гнучкість, захищеність та простоту масштабування. Саме ці характеристики роблять JAMstack ідеальним вибором для реалізації веб-орієнтованої інформаційної системи маркетплейсу.

Актуальність теми даної кваліфікаційної роботи обумовлена необхідністю створення високопродуктивного, безпечного та масштабованого

рішення для онлайн-продажу аксесуарів до комп’ютерної техніки, яке відповідатиме сучасним вимогам до UI/UX дизайну, доступності, мультимовності та мобільності. Вибір JAMstack підходу дає змогу задовольнити ці вимоги та створити конкурентоспроможний веб-застосунок із мінімальними витратами на підтримку інфраструктури.

У рамках даної роботи буде розглянуто теоретичні засади створення JAMstack-додатків, здійснено огляд існуючих аналогів, проаналізовано функціональні вимоги до системи та побудовано програмну архітектуру маркетплейсу. Також буде здійснено вибір інструментів реалізації та створено інтерфейсний прототип, який демонструє основні можливості системи: перегляд каталогу товарів, пошук, фільтрацію, мультимовний інтерфейс та базову взаємодію з користувачем.

Результати цієї кваліфікаційної роботи можуть бути використані як основа для подальшої розробки повноцінного комерційного рішення, а також як навчальний приклад побудови сучасного JAMstack-застосунку у сфері електронної комерції.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ В ГАЛУЗІ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Актуальність предметної області

Електронна комерція є однією з найдинамічніших галузей цифрової економіки. Глобальні продажі в сегменті онлайн-торгівлі демонструють стійкий двозначний ріст майже кожного року, що наочно відображено на діаграмі світових обсягів e-commerce за період 2017–2023 рр. (див. рис. 1.11). Пандемія COVID-19 лише пришвидшила цей тренд, змусивши підприємства різних масштабів інтенсивно переводити свої сервіси в режим онлайн. У результаті маркетплейси стали не просто майданчиками для продажу, а справжніми цифровими екосистемами, які об’єднують численні бізнес-процеси в єдину платформу з високим ступенем автоматизації.

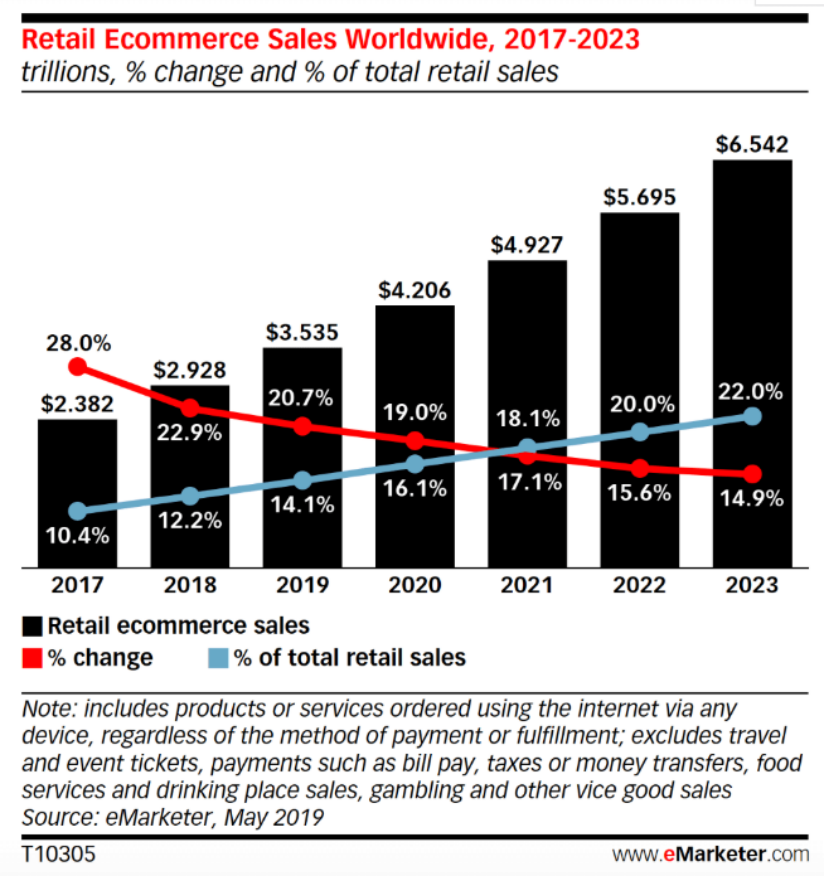

Рисунок 1.1 - Діаграма зростання світових обсягів онлайн-продажів, 2017–2023 рр.

Однією з ключових переваг маркетплейсів є можливість консолідації асортименту від різних продавців у межах єдиної інтерфейсної оболонки. Це дає змогу покупцям порівнювати характеристики й ціни візуально та інформаційно, а продавцям — отримувати доступ до розширеної аудиторії без значних інвестицій у власну інфраструктуру. Такий підхід особливо ефективний для нішевих ринків, як-от аксесуари до комп'ютерної техніки, де асортимент швидко оновлюється, а користувачі шукають конкретні за параметрами товари.

Водночас традиційні серверно-орієнтовані рішення виявляються малоефективними для підтримки високих навантажень, забезпечення швидкодії, безпеки та зручного мобільного інтерфейсу. У відповідь на ці виклики у веб-розробці формується архітектурна парадигма JAMstack, що передбачає відокремлення фронтенду, попередню генерацію markup-сторінок і звернення до динамічної логіки через API (див. рис. 1.2). Завдяки цьому рішення гарантує блискавичне завантаження з CDN, зниження навантаження на бекенд та посилення захисту від атак, оскільки статичний контент не потребує безпосереднього доступу до бази даних.

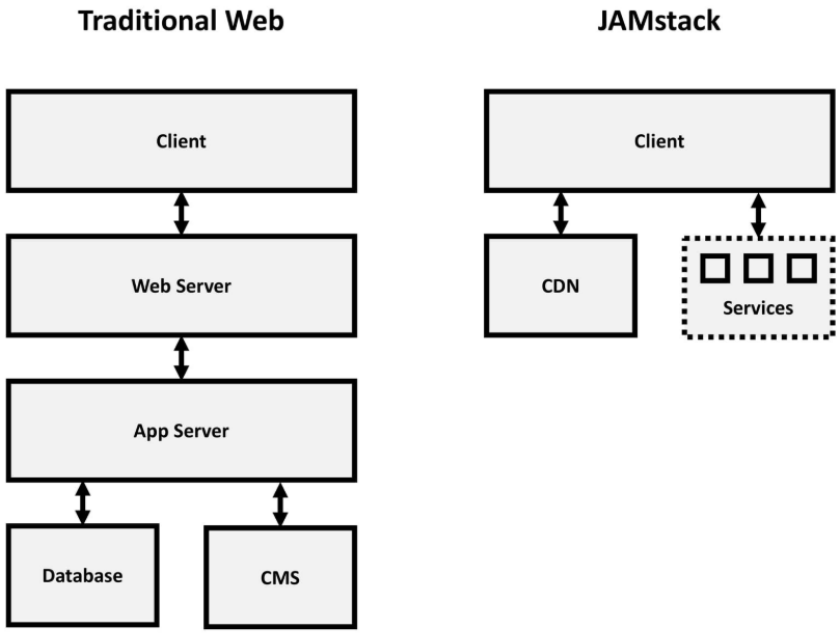

Рисунок 1.2 - Порівняння традиційної веб-архітектури та JAMstack

Також JAMstack бездоганно масштабується у глобальних мережах CDN, що критично для e-commerce, де кожна секунда затримки може зменшити конверсію. Поєднання з Headless CMS забезпечує гнучке управління контентом: структуровані дані й переклади зберігаються незалежно від фронтенд-логіки, що спрощує розробку мультимовних інтерфейсів.

Обрана предметна область — маркетплейс аксесуарів — характеризується частими оновленнями асортименту, великою кількістю постачальників та імпульсивними покупками. Фокус на JAMstack дозволяє створити інформаційну систему, яка відповідає сучасним вимогам бізнесу: висока продуктивність, безпека, масштабованість та зручність адміністрування контенту.

1.2 Визначення особливостей предметної області

Ринок аксесуарів до комп’ютерної техніки є одним із найдинамічніших сегментів IT-галузі. Він включає широкий спектр товарів: від периферійних пристроїв (мишки, клавіатури, геймпади, монітори) до супутнього обладнання (підставки, кабелі, адаптери, очищувачі частина категорій наведена на рис. 1.3) і компонентів для кастомізації (підсвітка, змінні кейкапи, охолоджувачі тощо). Усі ці товари об'єднує загальна характеристика — велика різноманітність моделей, брендів, технічних характеристик і цінових категорій.

Особливістю цього ринку є часті оновлення модельного ряду. Наприклад, виробники периферії постійно випускають нові моделі з покращеними характеристиками або зміненим дизайном. Також варто враховувати активну присутність на ринку не лише великих міжнародних брендів, а й малих компаній, що спеціалізуються на нішевих аксесуарах. Це створює потребу в постійному оновленні контенту — карток товарів, описів, зображень, технічних даних.

Рисунок 1.3 – Різноманіття категорій електроніки сумасних торгівельних систем

Інформаційна система для такого маркетплейсу повинна підтримувати гнучку категоризацію та фільтрацію товарів за ключовими параметрами — тип пристрою, сумісність, інтерфейс підключення, розміри, колір, матеріал тощо. Це суттєво полегшує навігацію для користувачів та скорочує час на пошук потрібного товару. Крім того, через велику кількість брендів і моделей виникає потреба в ефективному механізмі пошуку, який враховує як точні запити, так і синонімічні варіанти або часткові співпадіння.

У контексті електронної торгівлі важливу роль відіграє візуальна презентація товару. Для аксесуарів часто вирішальне значення має вигляд, ергономіка, стиль — тому система повинна підтримувати галереї зображень високої якості, збільшення за наведенням, 3D-огляди (опціонально). Також корисними є відгуки покупців, рейтингування, та можливість залишити

коментар, що впливає на довіру до товару та покращує взаємодію з клієнтом, зразок такої сторінки наведено на рисунку 1.4.

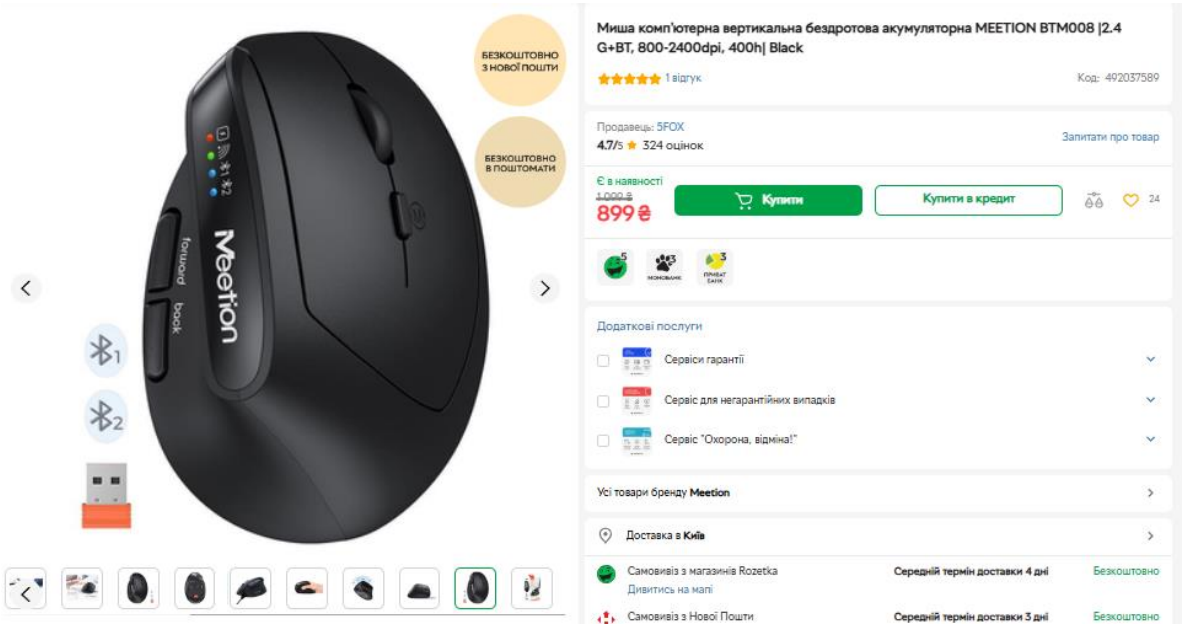

Рисунок 1.4 – Візуальне представлення товару на сторінці

Ще однією специфікою є потреба у мультимовній підтримці. Багато користувачів хочуть мати змогу переглядати контент українською, англійською чи іншими мовами. Тому система повинна мати можливість локалізації інтерфейсу, описів товарів, категорій і фільтрів. Бажано, щоб ця локалізація була реалізована централізовано — через CMS або конфігураційні файли.

Оскільки маркетплейс орієнтований як на кінцевого споживача (B2C), так і на бізнес-клієнтів (B2B), інтерфейс має враховувати два типи взаємодії. Для пересічного користувача — це зручність, візуальна привабливість, простота. Для B2B-сегменту можуть бути корисними функції групового замовлення, швидкого оформлення або завантаження прайсів (опціонально на наступному етапі розвитку).

Особливістю JAMstack-реалізації у цьому контексті є відокремлення даних і представлення. Наприклад, контент (товари, описи, локалізації) зберігається у headless CMS, а сторінки генеруються під час побудови або на

вимогу (через Serverless-функції). Це дозволяє досягти максимальної швидкодії, зменшити залежність від сервера, а також забезпечити масштабованість без ускладнень.

На основі вищезазначеного, до маркетплейсу електроніки висуваються наступні вимоги:

- підтримка великого обсягу товарів та оновлень;
- можливість зручної категоризації та фільтрації;
- ефективний пошук;
- мультимовність;
- адаптивний, естетичний інтерфейс;
- підтримка високоякісних зображень;
- можливість гнучкого управління контентом.

Таким чином, особливості предметної області диктують потребу у створенні сучасної, швидкої, масштабованої та зручної для адміністрування веб-орієнтованої інформаційної системи, що дозволяє реалізувати ці особливості оптимально як з технічної, так і з економічної зору.

1.3 Огляд існуючих систем в галузі електронної комерції

В умовах швидкого розвитку електронної комерції та підвищених вимог до продуктивності, безпеки та масштабованості сучасні компанії все частіше звертаються до спеціалізованих платформ і фреймворків для побудови маркетплейсів. Серед найпоширеніших підходів можна виділити класичні монолітні системи на базі CMS, SaaS-рішення, а також повністю headless чи JAMstack-архітектури. Кожне з цих рішень має як сильні сторони, так і обмеження, які варто враховувати при виборі платформи для конкретного проєкту.

Однією з найпоширеніших платформ для побудови невеликих чи середніх маркетплейсів залишається комбінація WordPress та WooCommerce. Завдяки колосальній екосистемі плагінів і тем оформлення, а також простоті встановлення, ця зв’язка підходить для швидкого запуску MVP. Проте, із

ростом асортименту і навантаження серверу така архітектура починає “гальмувати”: типові труднощі пов’язані із запитами до бази даних, необхідністю кешування, складнощами з оновленнями та ризиками вразливостей через надмірну кількість сторонніх розширень. Прикладом великого майданчика на WooCommerce є сайт групи роздрібних магазинів “Easy Buy”, проте його власники неодноразово стикалися з проблемами продуктивності при сезонних розпродажах.

У категорії SaaS-рішень лідирує Shopify. Платформа пропонує готову інфраструктуру, зручну адміністративну панель та інтегровані платежі “з коробки”, що робить її привабливою для бізнесів, які прагнуть мінімізувати технічне адміністрування. Прикладом великого маркетплейсу на базі Shopify є спортивний бренд Gymshark (див. рис. 1.5), який демонструє стабільні високі продажі при одночасному обслуговуванні сотень тисяч користувачів. Проте цей комфорт супроводжується обмеженою кастомізацією і vendor lock-in: складні функції доводиться реалізовувати через платні додатки, а перехід на іншу платформу потребує значних ресурсів.

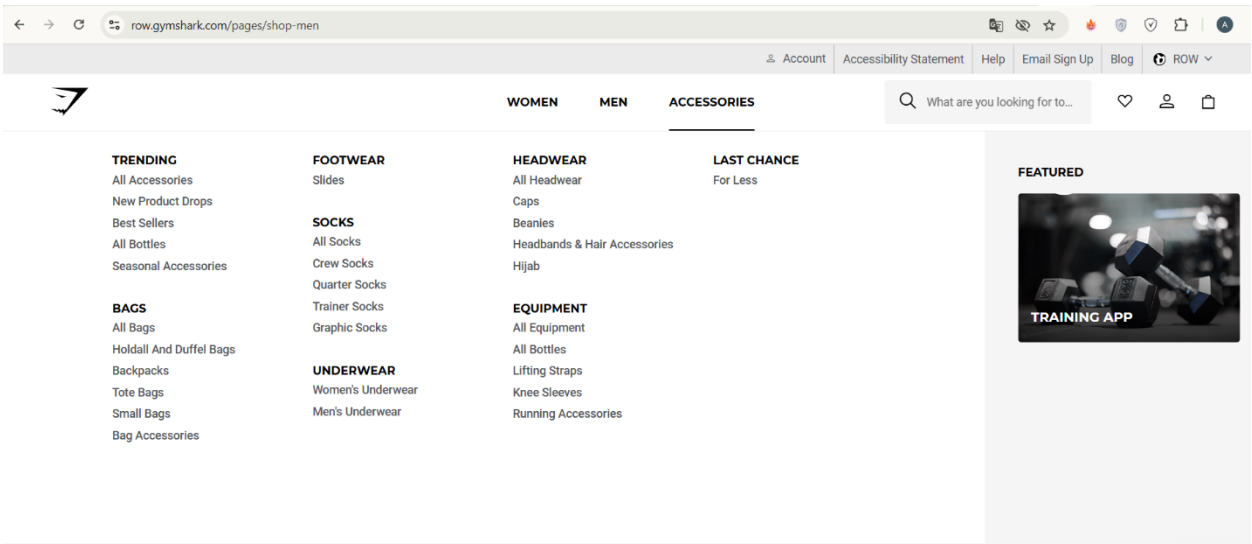

Рисунок 1.5 – Каталог товарів маркетплейсу Gymshark

Інший цікавий підхід — поєднання легкості JAMstack-фронтенду з можливостями кошика від Snipcart або аналогічних сервісів. Такі рішення

дозволяють стартапам і невеликим проєктам швидко інтегрувати базову e-commerce функціональність у статичні сайти. Однак Snipcart (див. рис. 1.6) не є повноцінним маркетплейсом і не надає складних механізмів управління кількома продавцями або диспетчеризації складських залишків.

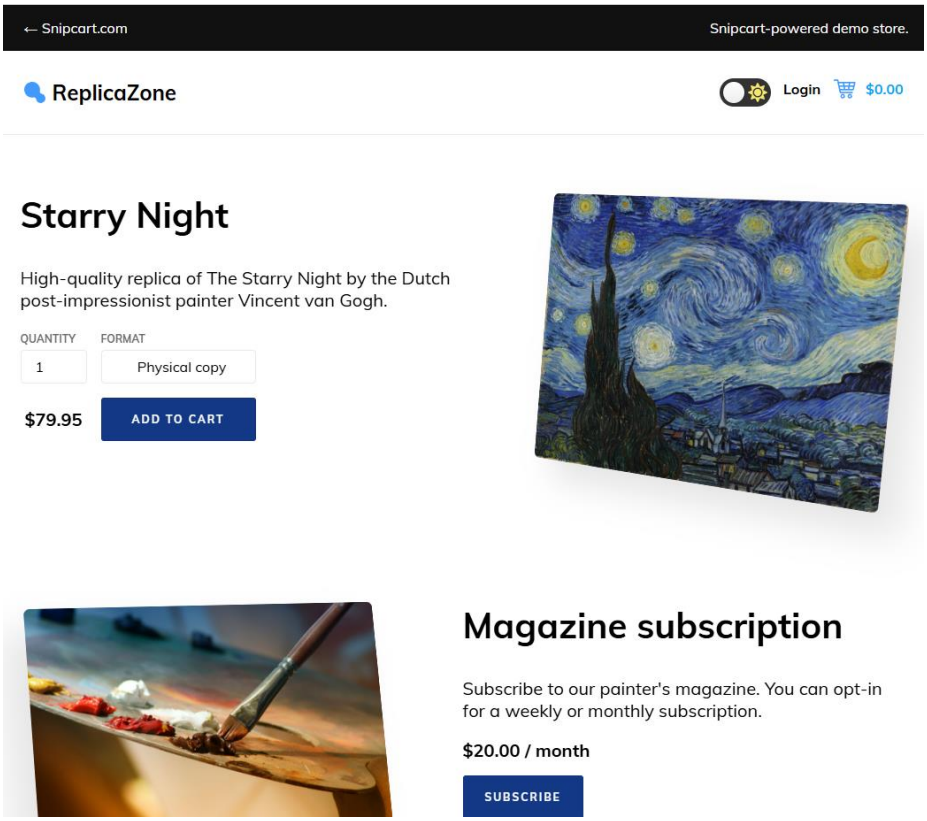

Рисунок 1.6 – Демо сторінка системи Snipcart

Найбільш гнучким і сучасним серед open-source-рішень вважається Medusa.js — headless e-commerce платформа, що надає повний набір API для каталогу товарів, кошика, замовлень і клієнтів. Завдяки готовому backend-ядру та можливості використання будь-якого фронтенду (Next.js, Gatsby тощо), Medusa забезпечує повну кастомізацію логіки та дизайну. На Ілюстрації 3.2 показано стартовий шаблон Medusa з Next.js, який демонструє потенціал швидкого створення преміального інтерфейсу. Крім того, Medusa активно використовується в проєктах малого та середнього бізнесу, наприклад, інтернет-магазині гарнітур для геймерів “ProAudioGear” (див. рис. 1.7).

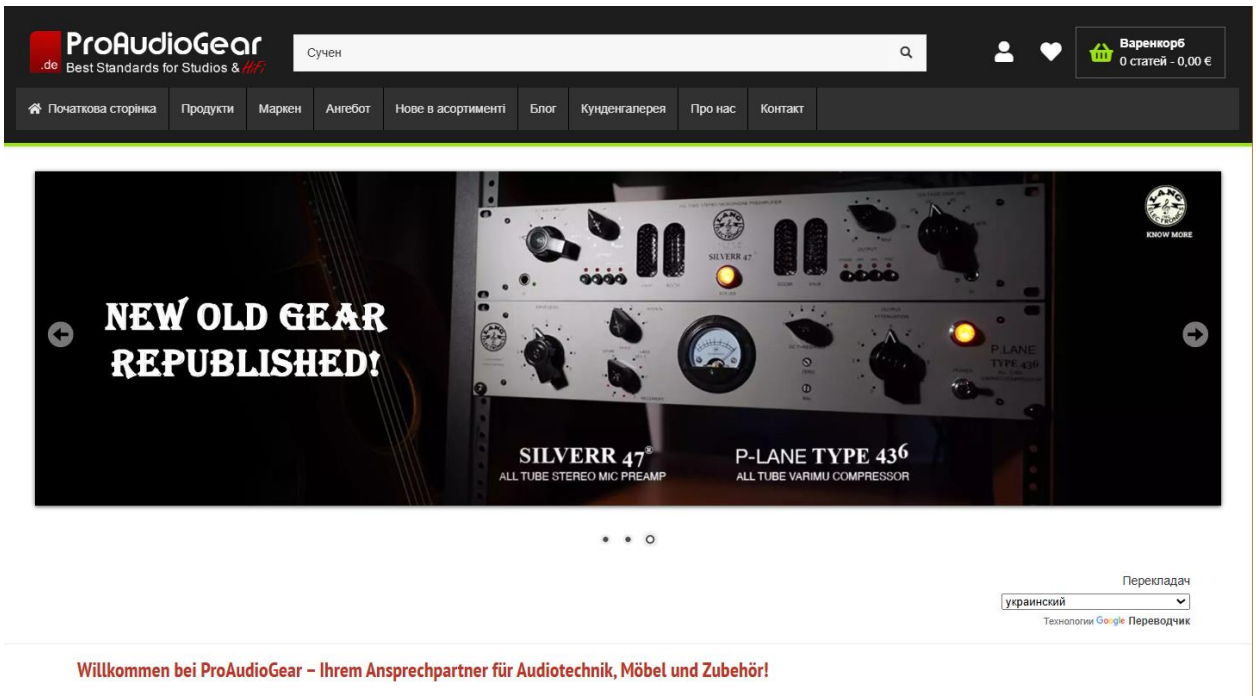

Рисунок 1.7 – Головна сторінка інтернет-магазину гарнітур ProAudioGear

Платформи класу Commerce Layer або Moltin, що також базуються на headless-архітектурі, орієнтовані на великі B2B-й B2C-екосистеми з багатомовністю, складною ціновою політикою та мультисайтами. Вони надають потужні API для управління цінами, локалями, складськими запасами та інтеграцією з ERP-системами. Проте рівень складності впровадження та вартість ліцензій можуть перевищувати бюджет невеликих проєктів.

У підсумку, класичні монолітні рішення доречні для простих магазинів з обмеженим асортиментом, але вони втрачають актуальність при масштабуванні. Shopify пропонує миттєву інфраструктуру, але обмежує свободу розробки. Snipcart і подібні сервіси підходять для мінімального e-commerce на статичних сайтах, але не замінюють повноцінний маркетплейс. Найбільш збалансований підхід для масштабного та кастомного маркетплейсу — headless-платформи на кшталт Medusa.js або Commerce Layer у поєднанні з JAMstack-фронтендом.

Обираючи рішення для створення маркетплейсу аксесуарів до комп’ютерної техніки, слід зважати на потребу підтримки великого каталогу, мультимовності, адаптивності й високої продуктивності. У цьому контексті

поєднання Medusa.js (або іншої headless-системи) із Next.js як фронтендом і CDN-доставкою через Netlify або Vercel забезпечує оптимальний баланс між швидкістю розробки, гнучкістю кастомізації та якістю роботи під навантаженням. Для невеликих MVP можна розпочати з Snipcart або Strapi+Gatsby, а згодом еволюціонувати до повноцінного headless-маркетплейсу, зберігши ключові принципи JAMstack-архітектури.

1.4 Узагальнення функціональних вимог до системи

Для розробки ефективної веб-орієнтованої інформаційної системи маркетплейсу необхідно чітко визначити перелік функціональних вимог, які повинна реалізовувати система. Функціональні вимоги — це опис того, що саме повинна робити система, щоб задовольнити потреби кінцевого користувача та адміністратора ресурсу. Вони є основою для подальшого проєктування архітектури, інтерфейсу та вибору технологій.

1. Публічний інтерфейс користувача (Front-End)

1.1. Перегляд каталогу товарів

Користувач повинен мати змогу переглядати товари, згруповані за категоріями: клавіатури, миші, кабелі, адаптери, підставки тощо. Кожна категорія може містити підкатегорії.

1.2. Розширена фільтрація

Необхідна можливість фільтрації товарів за параметрами: бренд, інтерфейс підключення (USB, Bluetooth), колір, ціновий діапазон, сумісність (з Windows / MacOS / іншим), тип матеріалу та інші технічні характеристики, специфічні для кожної категорії.

1.3. Пошук товарів

Інформаційна система повинна містити механізм пошуку за назвою, артикулом, ключовими словами та частковими співпадіннями. Бажано реалізувати автозаповнення (autocomplete) запитів.

1.4. Сторінка товару

Повна інформація про товар: назва, опис, технічні характеристики, галерея зображень, ціна, відгуки, рейтинг. Для зручності — функція збільшення зображення при наведенні.

1.5. Багатомовність

Усі інтерфейсні елементи, категорії, фільтри, описи товарів повинні мати переклади щонайменше українською та англійською мовами. Мовний перемикач має бути доступним на будь-якій сторінці.

1.6. Адаптивність

Інтерфейс має коректно працювати на різних пристроях: десктопах, планшетах, смартфонах. Застосовується принцип Mobile First або Responsive Design.

2. Функціональність адміністратора / контент-менеджера

2.1. Управління товарами

Адміністратор повинен мати змогу додавати, редагувати, видаляти товари, змінювати їхні характеристики, завантажувати зображення, задавати наявність, встановлювати ціну.

2.2. Управління категоріями та фільтрами

Платформа має дозволяти створювати ієрархію категорій, додавати нові параметри фільтрації залежно від типу товару. Це необхідно для адаптації під змінний асортимент.

2.3. Мультимовне редагування

Адміністратор повинен мати змогу додавати контент різними мовами: опис товару, характеристики, назви категорій та інше. Система має забезпечити зручну локалізацію.

2.4. Управління відгуками (опційно)

Можливість модерувати або приховувати нецензурні, спамні чи недоречні відгуки користувачів.

3. Інтеграції та технічна функціональність

3.1. Підключення до API / CMS

Система повинна працювати у зв’язці з Headless CMS (наприклад, Strapi, Sanity, Contentful або ін.), через яку здійснюється керування вмістом. Контент передається через REST або GraphQL API.

3.2. Статична генерація сторінок (SSG)

Для покращення швидкодії маркетплейс має використовувати попередню генерацію сторінок (Static Site Generation). При зміні даних у CMS — сторінки мають перегенеровуватись автоматично або за запитом.

3.3. Пошукова індексація

Система повинна бути оптимізованою для SEO: семантична верстка, підтримка OpenGraph/Schema.org, адаптивні URL тощо.

3.4. Кешування та CDN

Використання глобального CDN для розповсюдження статичного контенту — важлива вимога для забезпечення швидкого доступу до ресурсу з будь-якої точки світу.

4. Прототип інтерфейсу

У рамках цієї кваліфікаційної роботи також передбачається розробка високорівневого прототипу користувацького інтерфейсу, який демонструє:

- головну сторінку з банером, категоріями;
- сторінку каталогу з фільтрами;
- сторінку товару з детальним описом;
- мовний перемикач;
- адаптацію під мобільні пристрої.

Таким чином, веб-орієнтована інформаційна система маркетплейсу повинна поєднувати зручність навігації, візуальну привабливість, технічну оптимізацію та простоту адміністрування. Функціональні вимоги, викладені вище, становлять основу для подальшого проєктування архітектури системи та вибору засобів реалізації.

2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ МАРКЕТПЛЕЙСУ

2.1. Загальна концепція та архітектура програмного забезпечення

Проектування програмного забезпечення маркетплейсу аксесуарів до комп'ютерної техніки повинно базуватися на сучасних архітектурних підходах, які забезпечують високу продуктивність, гнучкість, масштабованість і безпеку. У межах даної кваліфікаційної роботи було обрано архітектурну парадигму JAMstack як оптимальне рішення для побудови сучасної веб-орієнтованої інформаційної системи.

JAMstack — це скорочення від JavaScript, API, Markup. У цій архітектурі клієнтська частина реалізується на JavaScript-фреймворку (наприклад, React або Next.js), контент зберігається у вигляді попередньо згенерованих статичних сторінок (Markup), а динамічна функціональність реалізується через зовнішні API (наприклад, CMS, пошукові сервіси або оплати).

Ключові переваги JAMstack у контексті маркетплейсу:

- швидкість: статичні сторінки доставляються користувачеві через CDN;
- безпека: відсутній сервер як точка атаки, всі запити відбуваються через захищені API;
- незалежність від серверного навантаження, ефективна робота з великою кількістю відвідувачів;
- легка інтеграція з будь-яким зовнішнім сервісом через API.

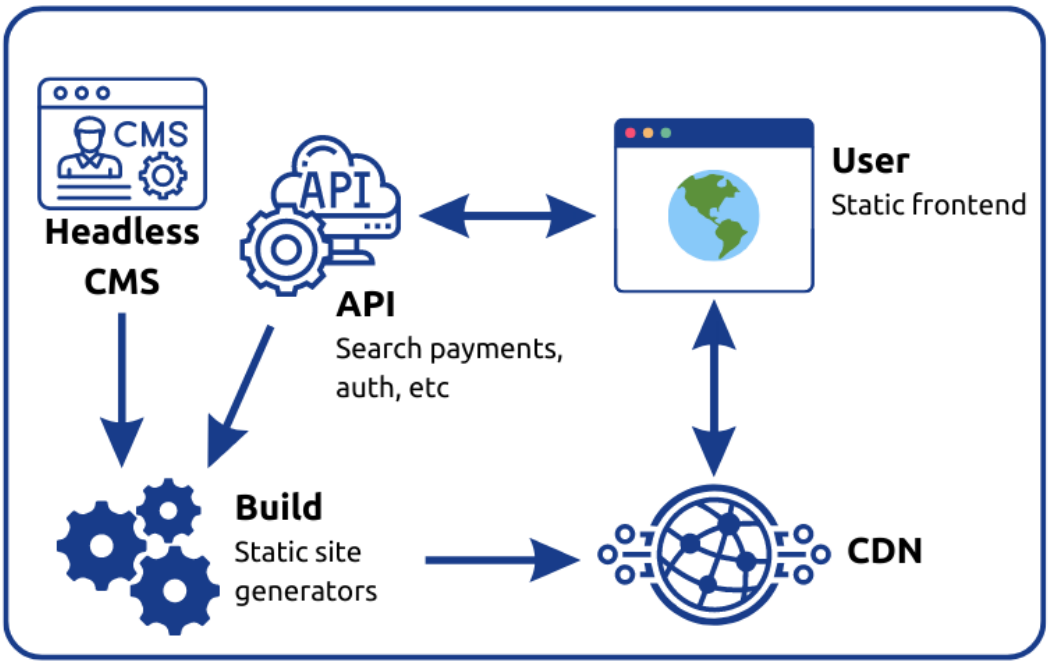

Рисунок 2.1 — Загальна схема JAMstack-архітектури:

Інформаційна система маркетплейсу складається з таких основних компонентів:

а) фронтенд (Next.js):

- відповідає за рендеринг інтерфейсу користувача;
- сторінки каталогу, карток товару, фільтрації, пошуку та мовного перемикання реалізуються як React-компоненти;
- підтримує статичну генерацію сторінок (Static Site Generation; Incremental Static Regeneration) для SEO та швидкості.

б) Headless CMS (Strapi / Contentful):

- служить сховищем контенту — категорій, товарів, описів, зображень;
- має зручний інтерфейс для адміністраторів (додавання/редагування контенту).
- працює через REST або GraphQL API.

в) API-шар:

- забезпечує зв’язок між фронтендом і CMS.

- може включати додаткові зовнішні API (наприклад, для пошуку або аналітики).

- у випадку необхідності динамічної логіки (наприклад, генерація sitemap) — застосовуються cервepless-функції.

г) CDN та хостинг (Netlify / Vercel):

- використовуються для доставки сайту по всьому світу з мінімальною затримкою.

- підтримують автоматичне оновлення сайту при зміні даних у CMS через webhooks.

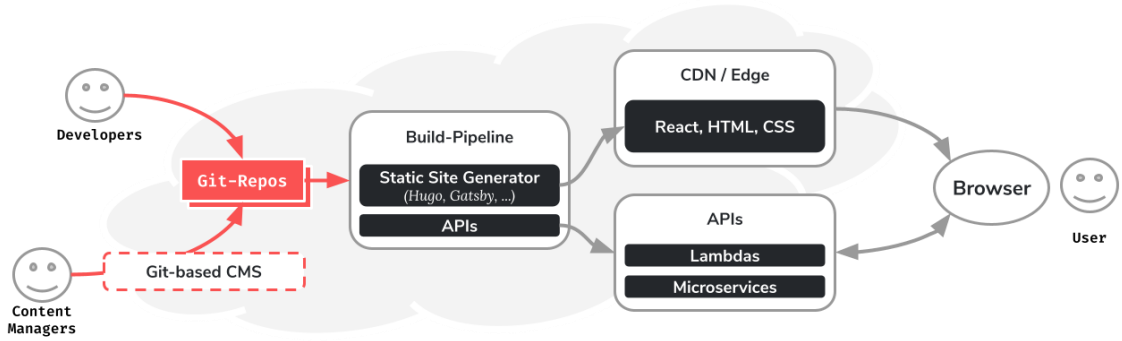

Рисунок 2.2 — Архітектурна модель JAMstack-додатку з CMS:

Архітектура побудована за принципом відокремлення рівнів:

- Користувач звертається до CDN за сторінкою (яка вже згенерована під час білду або ISR);
- Фронтенд-додаток (Next.js) при потребі надсилає запити до API CMS для отримання динамічного контенту;
- CMS повертає дані у форматі JSON через REST або GraphQL.

При оновленні контенту CMS надсилає webhook, який тригерить оновлення статичних сторінок (build hook) — це забезпечує актуальність даних.

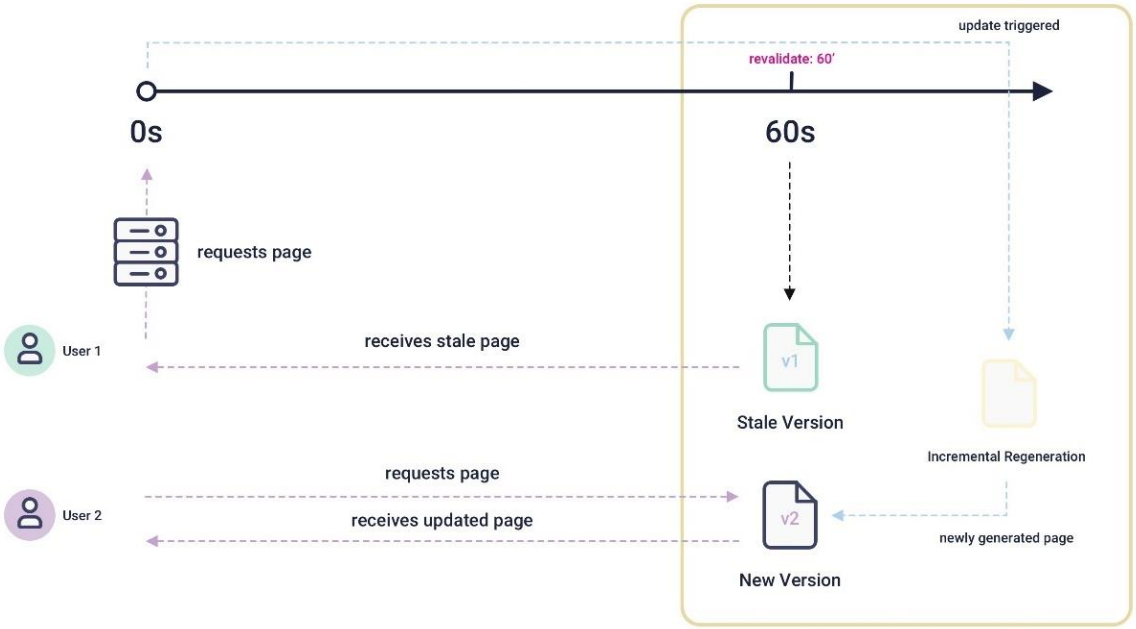

Рисунок 2.3 — Потік даних у JAMstack-системі з ISR:

Інтеграція системи відбувається наступним чином:

1) розробка фронтенду ведеться локально, після чого відбувається білд (збирання) сайту;
2) файли завантажуються на хостинг (Vercel або Netlify), де відразу стають доступними з CDN;
3) адміністратор змінює контент у CMS;
4) CMS автоматично надсилає webhook-запит для тригеру білду;
5) Нові сторінки генеруються та оновлюються на CDN.

Цей підхід дозволяє досягти поєднання продуктивності статичного сайту та гнучкості динамічного контенту, без потреби у постійному запиті до сервера при кожному перегляді сторінки.

Вибір JAMstack-підходу дозволяє створити архітектуру, яка повністю відповідає вимогам до маркетплейсу: висока швидкодія, підтримка мультимовності, гнучке управління контентом, безпека та масштабованість. Така архітектура є не лише актуальною, а й перспективною для розгортання в умовах реального бізнес-середовища, з можливістю подальшого розвитку (інтеграція оплати, аналітики, особистого кабінету тощо).

2.2. Моделювання предметної області

Для побудови ефективної, масштабованої та зручною у використанні інформаційної системи маркетплейсу необхідно провести моделювання предметної області — тобто визначити основні інформаційні сутності, їх атрибути, взаємозв’язки та ролі у системі. Це моделювання є критичним етапом, оскільки на його основі формується структура бази даних (або CMS), API-запити, а також логіка фронтенду при відображенні, пошуку та фільтрації товарів.

У контексті маркетплейсу аксесуарів до комп’ютерної техніки було визначено наступні ключові сутності:

1) товар (Product) - це основна одиниця інформації в системі. Кожен товар має атрибути наведені у таблиці 2.1:

Таблиця 2.1 – Структура сутності Product

Атрибут	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор
Назва	Рядок	Назва товару
Опис	Текст	Детальний опис
Ціна	Число	Вартість у базовій валюті
Зображення	URL	Посилання на зображення
Категорія	UUID	Зовнішній ключ на сутність «Категорія»
Характеристики	JSON	Параметри, специфічні для товару
Мова	Код мови	Мова опису (наприклад, "uk", "en")

2) Сутність User зберігає дані зареєстрованих користувачів системи. Вона необхідна для персоналізації функціоналу, збереження кошика, відстеження відгуків та (опціонально) замовлень.

Таблиця 2.4 – Структура сутності User

Назва поля	Тип даних	Опис
user_id	UUID / Integer	Унікальний ідентифікатор користувача
name	String (100)	Ім’я користувача
email	String (100)	Email-адреса (використовується для авторизації)
password_hash	String (255)	Хешований пароль користувача
role	String (20)	Роль користувача (наприклад, "user", "admin")
created_at	Timestamp	Дата та час створення облікового запису

3) Сутність Review дає змогу користувачам залишати оцінки та коментарі до товарів. Вона підвищує довіру до товару й надає іншим користувачам важливу інформацію під час прийняття рішення.

Таблиця 2.6 – Структура сутності Review

Поле	Тип даних	Опис
review_id	UUID / int	Унікальний ідентифікатор відгуку
user_id	UUID / int	Посилання на автора відгуку (FK до User)
product_id	UUID / int	Посилання на товар, до якого належить відгук
rating	int (1–5)	Оцінка товару
comment	text	Текстовий коментар
created_at	datetime	Дата створення відгуку
is_visible	boolean	Прапорець модерації (відображати/приховати)

4) категорія (Category) використовується для структурування товарів

Категорії є логічними групами товарів і можуть мати ієрархічну структуру (наприклад, "Клавіатури" → "Механічні клавіатури"). Вони допомагають організувати навігацію користувача. Структура сутності наведена в таблиці 2.2.

Таблиця 2.2 – Структура сутності Category

Атрибут	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор категорії
Назва	Рядок	Назва категорії
Батьківська категорія	UUID (nullable)	Посилання на батьківську категорію
Мова	Код мови	Мова, якою подається категорія

5) Сутність CartItem відображає товари, які користувач тимчасово додав до кошика. Вона дозволяє зберігати стан вибраних товарів до моменту оформлення замовлення.

Таблиця 2.5 – Структура сутності CartItem

Поле	Тип даних	Опис
cartitem_id	UUID / int	Унікальний ідентифікатор позиції в кошику
user_id	UUID / int	Посилання на користувача (FK до User)
product_id	UUID / int	Посилання на товар (FK до Product)
quantity	int	Кількість одиниць обраного товару
added_at	datetime	Дата й час додавання до кошика

6) Фільтр / Атрибут (Filter) описує додаткові характеристики товару, що можуть використовуватись для фільтрації:

Таблиця 2.3 – Структура сутності Category

Атрибут	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор фільтра
Назва фільтра	Рядок	Назва, що відображається в інтерфейсі
Тип	Рядок	checkbox / select / range
Категорія	UUID	Зв'язок із категорією

Сутності утворюють між собою наступні логічні зв’язки:

- один товар належить до однієї категорії;
- один товар має декілька атрибутів (бренд, колір, інтерфейс тощо);
- один товар має багатомовні поля, залежно від обраної мови;

-Один товар може мати декілька відгуків;

Фільтри використовуються у каталозі, але зберігаються окремо у структурі даних.

Таблиці демонструють логічну цілісність моделі даних: кожен користувач може залишити відгук на товар, а також формувати власний кошик. Вони критично важливі для персоналізованої взаємодії та базової функціональності будь-якого маркетплейсу. Ці зв’язки забезпечують можливість реалізації функціоналу: фільтрація, огляди товарів, кошик користувача.

Побудован ER-діаграма (див. рис. 2.4) демонструє сутності та взаємозв’язки, побудовані за UML-нотацією. Воно ілюструє центральну сутність "Product" зі зв’язками до категорій, фільтрів, відгуків та кошика.

Ця діаграма допоможе при реалізації SQL-запитів, проектуванні API та формуванні схеми CMS.

Ілюстрація на рисунку 2.4 демонструє логічні зв’язки між сутностями та їх атрибути, з позначенням типів зв’язків (1:N, N:M).

Оскільки система базується на JAMstack-підході, для зберігання контенту обрано headless CMS (Strapi). У такому середовищі кожна сутність є контентним типом, а поля (атрибути) задаються через візуальний інтерфейс. Взаємозв’язки реалізуються через типи зв’язків:

- One-to-Many (1:N): Категорія → Товари;
- Many-to-Many (N:M): Товари ↔ Атрибути (наприклад, бренд, сумісність);
- Localized Fields: поля, які дублюються у кількох мовах з прив’язкою до локалі.

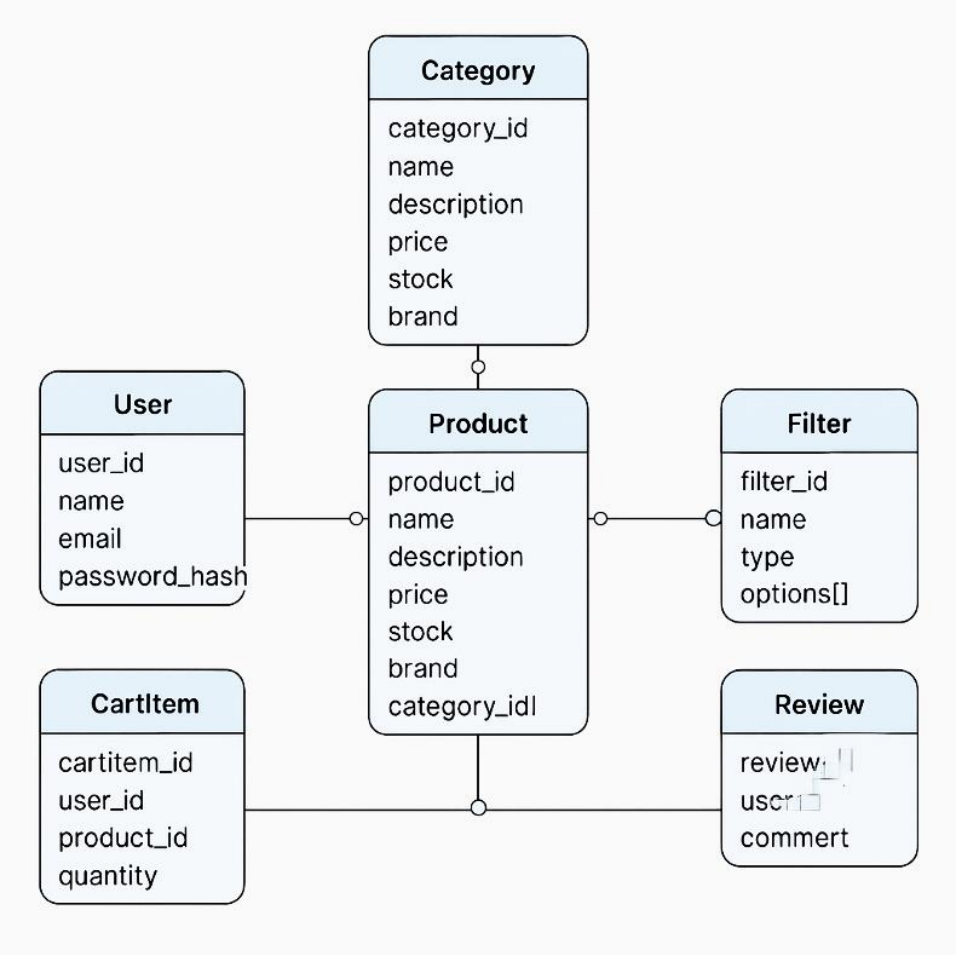

Рисунок 2.4 - ER-діаграма сутностей маркетплейсу

Таке структурування дозволяє гнучко редагувати дані через панель аміністрування CMS, швидко оновлювати контент без повторного розгортання коду, а також ефективно формувати API-запити.

Моделювання предметної області дало змогу сформувати чітке уявлення про структуру системи, необхідні сутності та зв’язки між ними. Обрана структура відповідає потребам маркетплейсу, що підтримує категоризацію, фільтрацію, багатомовність та масштабованість. Вона стане основою для створення API, конфігурації CMS та побудови інтерфейсу на фронтенді.

2.3. Проектування користувацького інтерфейсу

Ефективне проектування користувацького інтерфейсу є одним із найважливіших етапів розробки інформаційної системи маркетплейсу. Основна мета цього етапу — створити інтуїтивно зрозуміле, візуально привабливе й функціонально зручне середовище взаємодії між користувачем і системою. Важливо забезпечити, щоб навігація, фільтрація, перегляд товарів, додавання в кошик і доступ до додаткових можливостей були максимально простими та логічними. Особливу увагу слід приділити адаптивності, оскільки значна частина користувачів здійснює покупки з мобільних пристроїв.

На основі функціональних вимог було спроектовано макет основних сторінок маркетплейсу: головної сторінки, каталогу товарів, детальної сторінки продукту, кошика та мультимовного перемикача. На рисунку 2.5 представлено базовий вайрфрейм головної сторінки системи. Макет побудовано з урахуванням класичної логіки електронної комерції: верхнє горизонтальне меню містить логотип, панель пошуку, посилання на акаунт користувача та кошик. Одразу під ним розташована навігаційна панель категорій. У центральній частині екрану розміщується hero-блок — візуально виразний банер з короткою інформацією або акцією. Нижче реалізовано блоки “Рекомендовані товари” та “Аксесуари”, кожен з яких складається з карток товарів із зображеннями, назвами, цінами, рейтингами та кнопкою "Додати в кошик".

Рисунок 2.5 - Макет головної сторінки

Бічна панель праворуч (sidebar) у wireframe призначена для реалізації фільтрації — одна з найважливіших функцій для маркетплейсу аксесуарів. Користувач може швидко обрати товари за брендом, кольором, типом інтерфейсу чи діапазоном цін. Ці елементи реалізуються у вигляді чекбоксів, радіокнопок або повзунків. Інтерфейс має бути зрозумілим з першого погляду, без потреби в додаткових поясненнях.

Особливістю JAMstack-підходу є те, що інтерфейс повністю формується на фронтенді, а динамічний контент надходить із CMS через API. Це визначає технічну структуру реалізації: компоненти, які відповідають за відображення товарів, фільтрів, рейтингу тощо, повинні бути спроєктовані як ізольовані й повторно використовувані модулі. Для цього найчастіше використовується фреймворк React (в нашому випадку — Next.js), який дозволяє ефективно будувати адаптивні компоненти та оптимізувати взаємодію користувача з системою.

Під час проєктування також враховано потребу в адаптивному дизайні. У мобільній версії (див. рис. 2.6-2.7) навігаційне меню згортається в бургер-іконку, фільтри реалізуються у вигляді згортаних списків, а картки товарів

автоматично переходять у сітку 1–2 колонки, щоб забезпечити зручність перегляду. Таке проєктне рішення є критичним для підвищення конверсії з мобільних пристроїв.

Ще однією важливою вимогою було забезпечення мультимовності. На кожній сторінці інтерфейсу має бути доступна панель перемикання мов (наприклад, у вигляді іконки прапора). Усі статичні елементи (написи, кнопки, повідомлення) реалізуються через систему локалізації, інтегровану в фреймворк. Текстовий контент (опис товарів, категорії) надходить із CMS, де кожна мовна версія зберігається окремо.

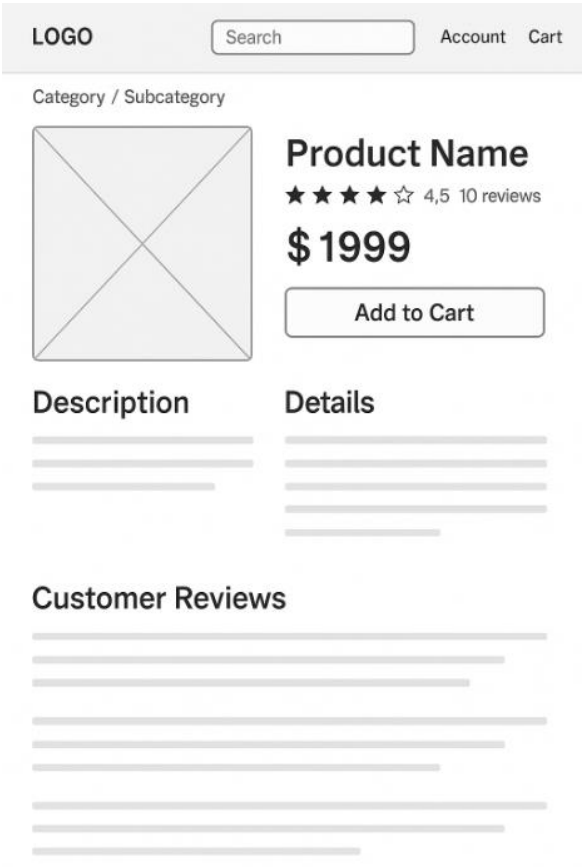

Рисунок 2.6 – Шаблон сторінки товару в мобільній версії

Рисунок 2.7 – Шаблон сторінки товару в мобільній версії

Таким чином, запропонований інтерфейс відповідає сучасним стандартам дизайну, зручності користування та технологічної реалізації. Він дозволяє ефективно реалізувати ключові функції маркетплейсу, забезпечуючи водночас швидкодію, естетичність і зрозумілу навігацію для користувача. У наступному розділі буде розглянуто вибір засобів реалізації, що забезпечують ефективне втілення цього інтерфейсу в реальному коді.

2.4. Вибір засобів реалізації та середовища розробки

При підборі технологічного стеку для розробки маркетплейсу аксесуарів до комп’ютерної техніки були враховані ключові вимоги: висока продуктивність, безпека, простота інтеграцій, можливість масштабування та швидка реалізація MVP. Нижче наведено опис і обґрунтування вибору основних технологій.

Основним фреймворком для реалізації клієнтської частини було обрано Next.js (див. рис. 2.8) завдяки своїй двопінговій підтримці Static Site Generation (SSG) та Server-Side Rendering (SSR), а також можливості використовувати Serverless-функції для API. React дозволяє створювати компонентний

інтерфейс, що легко масштабувати, підтримувати та повторно використовувати. Ще одна суттєва перевага — оптимізоване управління зображеннями та маршрутизацією, що покращує SEO і швидкість роботи.

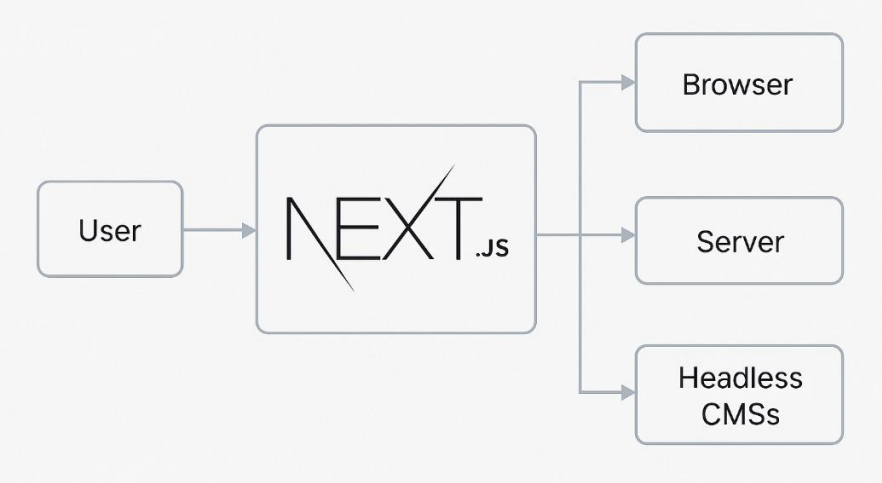

Рисунок 2.8 - — Архітектура Next.js як фронтенду:

Для управління контентом обрано підхід Headless CMS. Серед розглянутих рішень — Strapi, Contentful та Sanity.

Strapi (open-source) дозволяє розгорнути CMS самостійно, має гнучку модель даних та локальний плагін i18n.

Contentful — хмарний SaaS з інтуїтивним редактором, багатомовністю, готовими SDK.

Sanity — схожий на Contentful, із сильним акцентом на налаштувану CMS-модель і real-time редагування.

Вибір між ними ґрунтується на потребі кастомізації (Strapi) і готових хмарних сервісах (Contentful/Sanity). Незалежно від обраного CMS, Next.js легко інтегрується через REST/GraphQL API.

API-логіка реалізується у вигляді serverless-функцій на Node.js, що дозволяє розгортати їх разом з фронтендом (на Vercel/Netlify). Це ідеальний варіант для обробки webhook з CMS, реалізації авторизації чи обчислення кошика без постійного сервера. Такий підхід забезпечує високий рівень безпеки і продуктивності з мінімальними витратами.

Для розгортання обрано Vercel — оригінального розробника Next.js. Платформа забезпечує автоматичний деплой при пуші у Git, глобальний CDN та підтримку SSG/ISR/SSR .

Як альтернативу розглянуто Netlify, що також підтримує serverless-функції та CDI для інших фреймворків.

Для обґрунтованого вибору стеку технологій було проведено порівняння доступних рішень за ключовими критеріями, які мають вирішальне значення для реалізації веб-орієнтованого маркетплейсу: продуктивність, безпека, простота інтеграції, масштабованість та зручність підтримки. Кожна з розглянутих технологій оцінювалась у контексті її відповідності архітектурі JAMstack, вимогам до швидкодії інтерфейсу, обробки даних, мультимовності та адміністративної гнучкості. У результаті такого порівняння сформовано узагальнюючу таблицю 2.7, яка наочно демонструє переваги обраного стеку (Next.js, Headless CMS, Serverless API, Vercel/Netlify) над альтернативами та дозволяє підтвердити його доцільність для реалізації цілей проекту що розробляється в рамках кваліфікаційної роботи.

Таблиця 2.7 - Порівняння технологій по критеріям

Критерій	Next.js / React	Headless CMS	Node.js + Serverless	Vercel / Netlify
Продуктивність	Висока (SSG + ISR)	Залежить від API	Мінімальна латентність	CDN — висока швидкість
Безпека	Обмежений SSR, XSS	Захищена адміністрація	Ізольовані функції	HTTPS, автоматичні SSL
Інтеграція	Легка через API	Багато пакетів і плагінів	API для фронтенду	Git-інтеграція, deploy hooks
Масштабованість	Горизонтальна, CDN	Масштабовані API	Автоматичне масштабування	Автоматичне масштабування
Простота підтримки	Висока, спільнота	Адмін-панелі, UI	Налаштування функцій	Zero-config деплой

Обраний стек — Next.js + Headless CMS + Serverless API + Vercel — повністю відповідає вимогам до сучасного маркетплейсу. Він забезпечує:

- проміжок швидкодії;
- простоту інтеграції мультимовності та контенту;
- безпеку за рахунок ізоляції;
- мінімальні витрати на інфраструктуру;
- можливість швидког о прототипування і подальшого масштабування.

Це дозволяє реалізувати MVP якісно, фокусуючись на UX/UI та ключовому функціоналі без надмірного технічного боргу.

2.5. Реалізація основного функціоналу системи

Розробка основного функціоналу веб-орієнтованої інформаційної системи маркетплейсу є ключовою частиною програмного проєкту, оскільки саме вона визначає, як кінцевий користувач взаємодіє із системою, переглядає товари, здійснює пошук, фільтрацію, перемикає мову інтерфейсу, формує замовлення тощо. У межах цього підрозділу описано реалізацію ключових можливостей фронтенд-частини з використанням сучасного фреймворку Next.js, інтеграцію з CMS для керування вмістом, а також технічні аспекти побудови інтерфейсу з урахуванням JAMstack-архітектури. Особлива увага приділена якості взаємодії клієнта з динамічним контентом, швидкодії, адаптивності та розширюваності функціональних модулів (див. рис. 2.9).

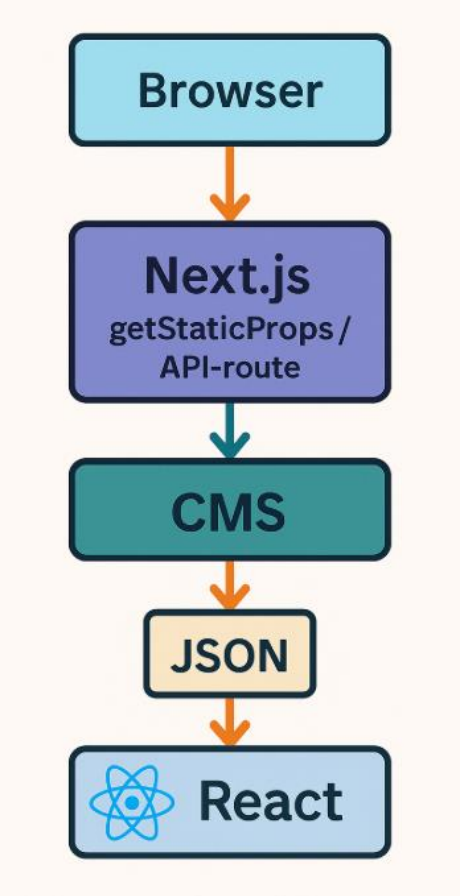

Рисунок 2.9 — Потік запиту клієнта до CMS через апі-клієнт

Описано реалізації ключових функціональних модулів веб-маркетплейсу — від виведення каталогу товарів до інтеграції мультимовності та адмін-панелі, — з акцентом на комплексний підхід до фронтенду, побудованого за допомогою Next.js та React, і бекенду, представленого Headless CMS та serverless-функціями.

Фронтенд-частина реалізована за допомогою Next.js, що дозволяє використовувати SSG, ISR, а також динамічні апі-роути. Структура проекту відображає модульний підхід: каталоги, фільтри, сторінки товарів, кошик, аккаунт — кожен сегмент реалізовано як окремий компонент. Взаємодія з CMS відбувається через централізований клієнт (api.ts), що інкапсулює запити до Strap.

Виведення каталогу товарів

Каталог формується на базі функції getStaticProps, яка робить запит до https://cms.example.com/api/products?lang=uk для отримання локалізованого контенту. Отримані дані передаються у компонент CatalogPage (див. рис. 2.10).

```
export async function getStaticProps({ locale }) {
  const res = await fetch(`${CMS_URL}/products?lang=${locale}`);
  const products = await res.json();
  return { props: { products }, revalidate: 60 };
}
```

Рисунок 2.10 – Програмний код формування каталогу товарів

Таке рішення дозволяє генерувати актуальний сортований набір продуктів, що згодом передається в React-компоненти для рендерингу.

У компоненті CatalogPage.tsx використовується мапінг по масиву об'єктів, кожен з яких рендериться через ProductCard (див. рис. 2.11).

```
{products.map(p => (
  <ProductCard key={p.id} product={p} />
))}
```

Рисунок 2.11 – Фрагмент коду рендерингу товару

Цей компонент займається візуалізацією — зображення, назва, ціна, кнопка "Додати в кошик".

Фронтенд-фільтрація виконується за допомогою React-стану, пов’язаного з фільтровими умовами. Наприклад, фільтр за брендом використовує конструкцію:

```
const [brandFilter, setBrandFilter] = useState("");
const filtered = products.filter(p => p.brand.includes(brandFilter));
```

Рисунок 2.12 – Організація фільтрації товарів засобами React

Щоб реалізувати пошук, додано input, який оновлює стан searchQuery і фільтрує товари за назвою або описом. Якщо фільтрів більше (мінімальне та максимальне значення ціни або технічні характеристики), клієнт може або фільтрувати на місці, або виконувати запит до бекенда:

Деталізація реалізована через динамічний роутинг 'pages/product/[slug].tsx'. Спершу формується список getStaticPaths зі шляхами на основі slug, отриманих із CMS:

```
export async function getStaticPaths() {
  const products = await fetch(`${CMS_URL}/products`).then(res => res.json());
  const paths = products.map(p => ({ params: { slug: p.slug } }));
  return { paths, fallback: 'blocking' };
}
```

Рисунок 2.13 – Організація динамічного роутингу

Далі в getStaticProps по slug витягується детальна інформація — опис, галерея, характеристики, рейтинг, відгуки:

```
export async function getStaticPaths() {
  const products = await fetch(`${CMS_URL}/products`).then(res => res.json());
  const paths = products.map(p => ({ params: { slug: p.slug } }));
  return { paths, fallback: 'blocking' };
}
```

Рисунок 2.14 – Організація завантаження детальної інформації по товару

На сторінці відображається галерея зображень, короткий опис, ціна, технічні дані, кнопка для додавання до кошика та блок відгуків, які витягуються як частина даних(див. рис. 2.15)

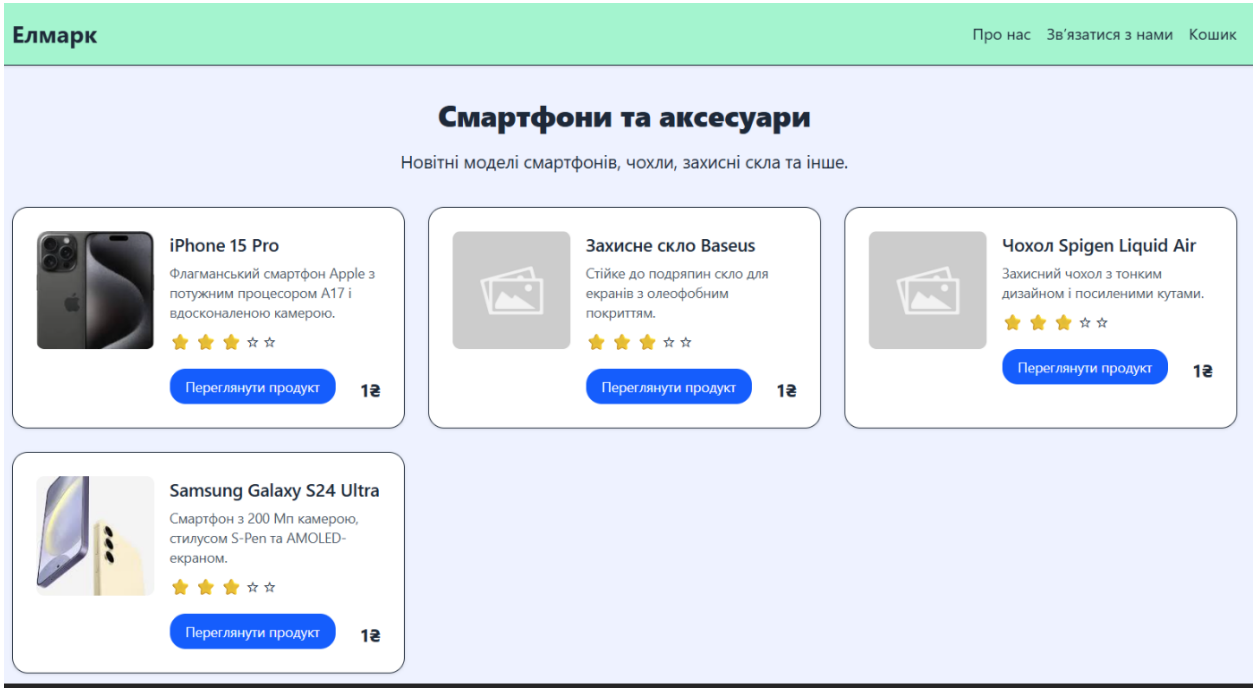

Рисунок 2.15 – Сторінка каталогу товарів

Мультимовність реалізовано за допомогою бібліотеки i18next, що дозволяє працювати з перекладами інтерфейсу через JSON-файли (див. рис. 2.16).

```
import { useTranslation } from 'next-i18next';
const { t } = useTranslation('common');
<h1>{t('catalog.title')}</h1>
```

Рис. 2.16 – Фрагмент організація мультимовного інтерфейсу засобами бібліотеки i18next

Аналіз показує, що тексти в UI та назви елементів витягуються з локалізованих файлів, а контент також надходить з CMS у відповідній мовній гілці.

Кошик реалізований як контекст або redux-store, що обробляє CartItem: кожний товар додається у кошик з ідентифікатором, кількістю і ціною. Компонент CartPage.tsx робить запит до локального store, а також попередньо

зберігає стан у localStorage. Візуальне представлення кошика наведено на рисунку 2.17

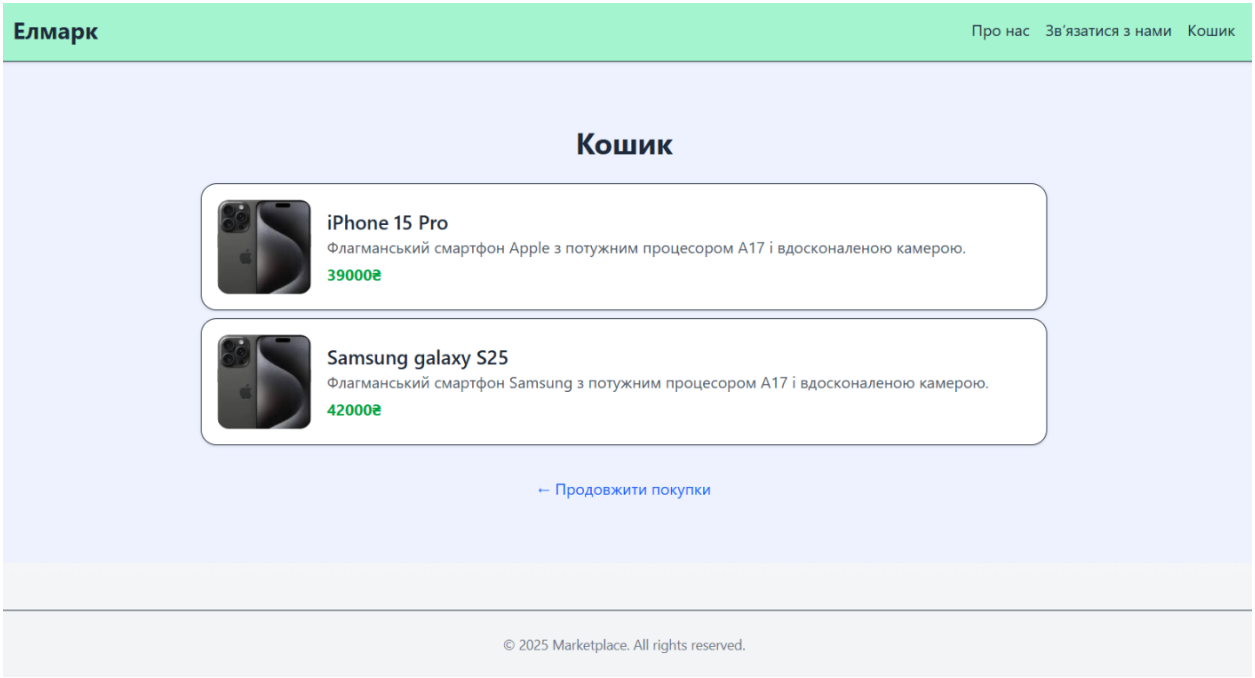

Рисунок 2.17 – Сторінка кошику користувача

Для забезпечення гнучкості контент-менеджеру обрана Headless CMS, така як Strapi. Контент-менеджер отримує змогу самостійно додавати або редагувати поля, переклади, технічні характеристики, зображення. Після збереження CMS ініціює webhook на Vercel чи Netlify, який запускає перегенерацію сторінок. Це гарантує, що нові товари або оновлення інтерфейсу відображаються на сайті без необхідності ручного запуску build:

```
Strapi Admin UI → Post save → Webhook → Vercel/JAMstack Build → CDN (updated pages)
```

ext

Дерево директорій проєкту представлено на рис. 2.18.

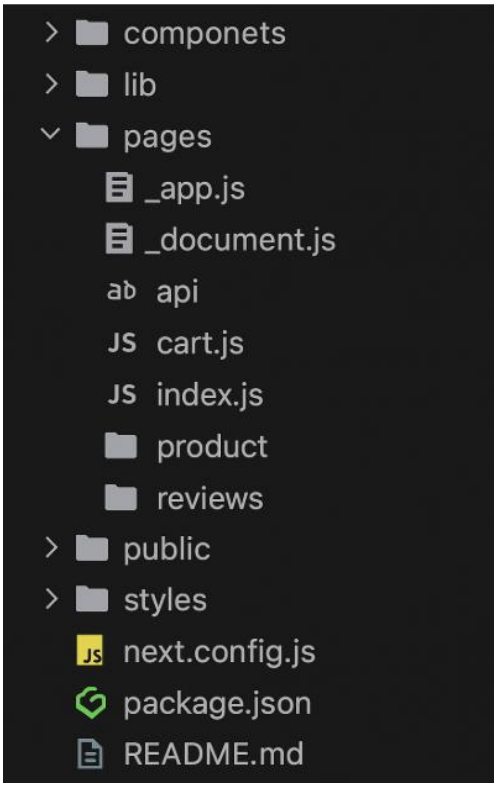

Рисунок 2.18 – Дерево каталогів та файлів проекту

Загалом, реалізація демонструє цілісний цикл: витяг контенту з CMS, статична генерація каталогу та сторінок, клієнт-фільтрація і пошук, підтримка мультимовності, управління кошиком, зображення товарів і галереї, а також відгуків та особистого кабінету користувача. Функціональність охоплює базові потреби B2C маркетплейсу з акцентом на високу швидкість, UX і можливість майбутнього розширення (авторизація, оплата). Такий підхід забезпечує легкість масштабування без навантаження на сервер і спрощує подальше управління контентом.

2.6. Побудова та розгортання системи

Процес побудови та розгортання JAMstack-маркетплейсу поєднує унікальні можливості статичної генерації, автоматизованих білдів і глобального CDN, що забезпечує блискавичну доставку контенту користувачам. У цьому підрозділі описано етапи побудови фронтенду, деплою

на платформи Netlify або Vercel, інтеграцію з CMS для оновлення контенту та механізми автоматичної регенерації сторінок через ISR і вебхуки.

На етапі побудови (див. рис. 2.19) Next.js у CLI або платформах Vercel/Netlify виконує скрипт npm run build, який запускає процедури попередньої генерації усіх необхідних статичних сторінок. З урахуванням налаштованої опції revalidate у getStaticProps, частина запитів обробляється за допомогою Incremental Static Regeneration, що гарантує актуальність контенту без повного ребілду. В результаті виходить готовий каталог HTML-файлів, JavaScript-пакетів та статичних ресурсів (зображень, CSS, шрифтів), збережених у директорії .next або out (якщо використовується next export).

На рисунку 2.19 показано результат виконання команди npm run build у терміналі Visual Studio Code. Лог збірки демонструє кроки створення оптимізованої production-зв'язки Next.js: завантаження змінних оточення, генерацію статичних сторінок, фіналізацію оптимізації, а також розмір згенерованих ресурсів і повідомлення про успішну компіляцію клієнтської та серверної частин.

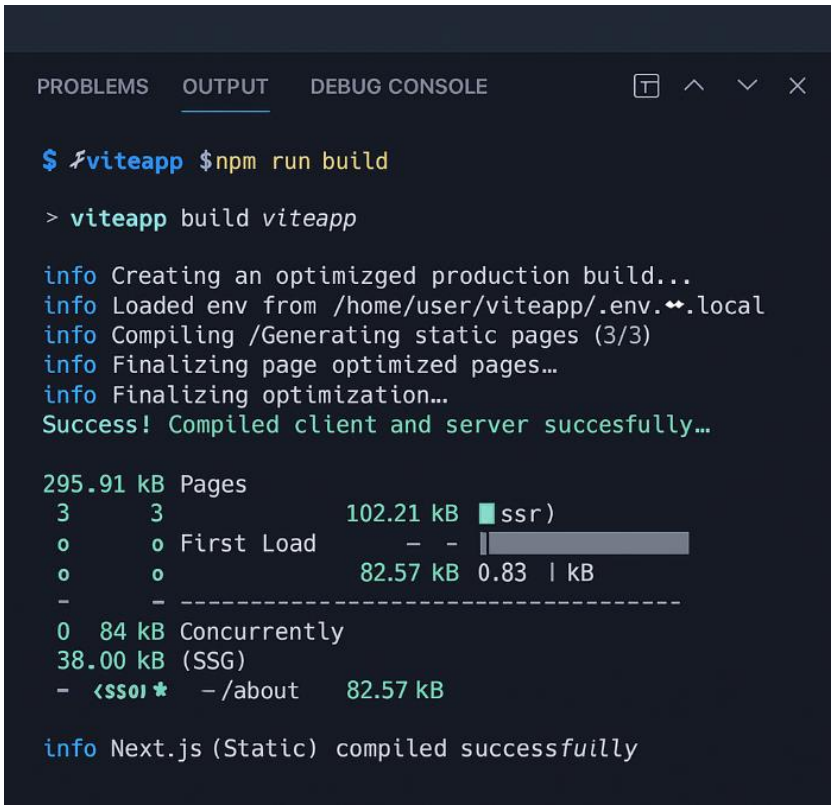

Рисунок 2.19 — Лог побудови фронтенду в IDE

Після успішної збірки проекту файли автоматично завантажуються на CDN платформи розгортання. Netlify та Vercel забезпечують безшовну інтеграцію з Git-репозиторієм: кожен пуш у головну гілку ініціює новий білд і deploy. Інтерфейс Netlify демонструє статуси деплоїв (Production deploys, Deploy previews, Branch deploys) та дозволяє вручну тригерити деплой або відкотитися до попередньої версії. Аналогічно, Vercel автоматично створює production- та preview-деплоя при кожному коміті.

На рисунку 2.20 відображено розділ Deploys Netlify: активний production deploy, список preview та branch deploys з їх статусами та часом публікації. Кнопка Trigger deploy дозволяє вручну запускати білд. цієї частини дипломного звіту.

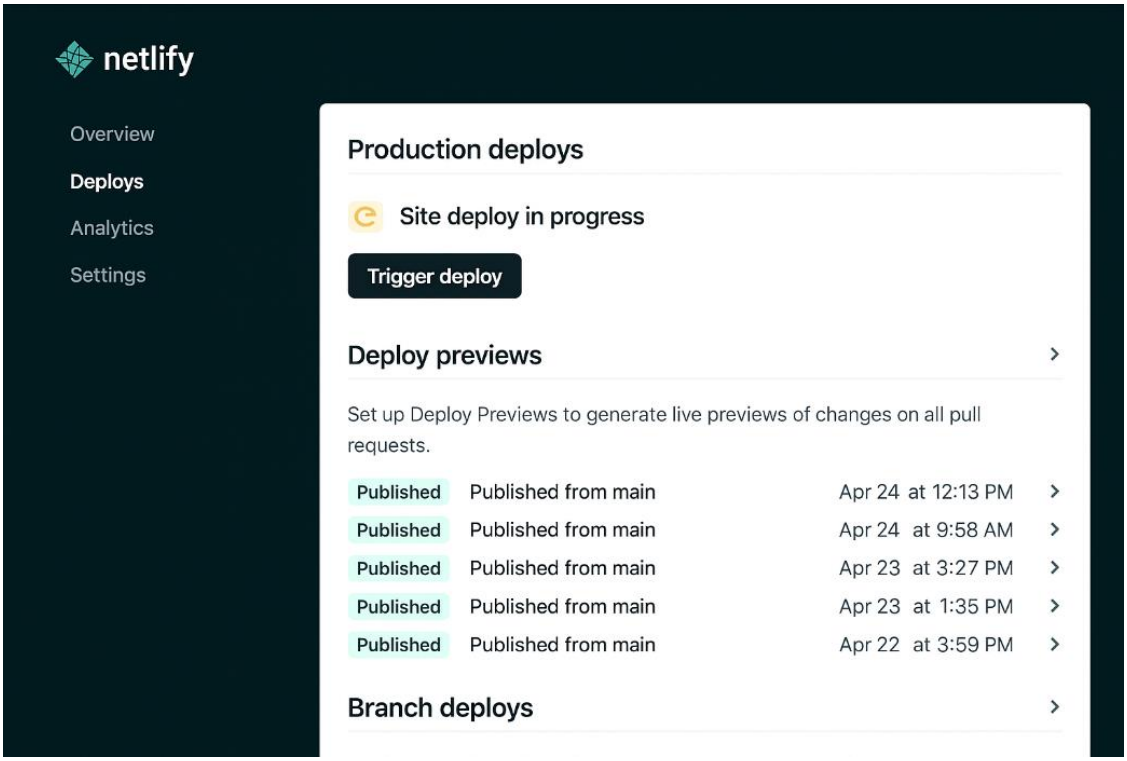

Рисунок 2.20 — Панель деплою Netlify

Ця система гарантує, що кінцевий користувач завжди отримує контент найновішої версії сайту з найближчого вузла CDN, що зменшує час завантаження і підвищує задоволеність від користування.

Оскільки контент магазинної частини — товари, описи та локалізовані поля — зберігається в Headless CMS, після внесення змін у адмін-панелі CMS

надсилає webhook на платформу розгортання. Цей webhook автоматично запускає новий білд, в результаті якого оновлені дані підтягуються в процесі SSG/ISR.

На рисунку 2.21 представлено розділ Webhooks у адмін-панелі Strapi: ліворуч — навігаційне меню з меню “Webhooks” під Settings, праворуч — таблиця наявних вебхуків з їх URL та статусом увімкнення/вимкнення і кнопкою Create new webhook.

Таким чином, адміністратор не потребує ручного втручання у процес деплою — оновлення контенту відбувається одразу після збереження в CMS, а користувачі бачать актуальні дані вже за кілька секунд.

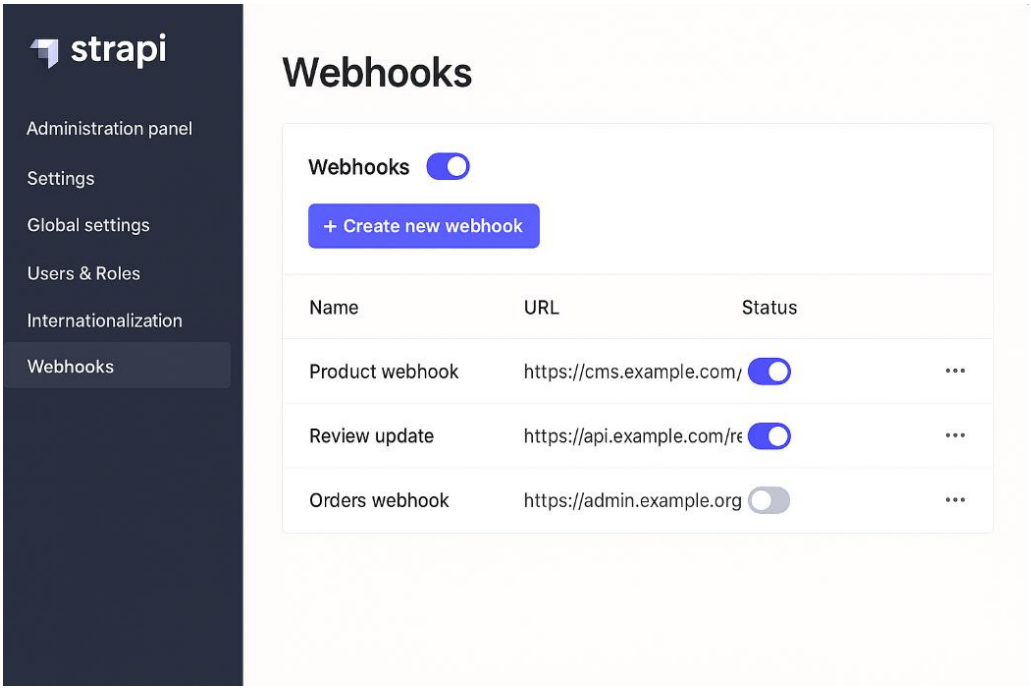

Рисунок 2.21 — Панель Strapi з webhook-налаштуваннями

Механізм Incremental Static Regeneration дозволяє налаштувати кожну статичну сторінку так, щоб, отримавши запит і вичерпавши заданий інтервал revalidate, Next.js у фоновому режимі перегенерувала сторінку без перебоїв у роботі. Це дає змогу комбінувати переваги статичних сайтів (швидкість, безпека) із актуальністю даних, властивою динамічним системам.

Налаштування вебхуків із CMS забезпечує додатковий тригер регенерації не за часом, а за подією, що відбувається у адмінці.

Після завершення деплою проект отримує URL виду jamstack-electronics-marketplace.vercel.app або кастомний домен, налаштований у панелі Vercel/Netlify. Домени прив’язуються через DNS-записи, а платформи автоматично подбають про SSL-сертифікати.

На рисунку 2.22 зображено розділ Domains у панелі Vercel: кнопка Add для додавання нового домену, перелік існуючих доменів із їхнім статусом (наприклад, Production або Invalid) та посилання на документацію для налаштування DNS.

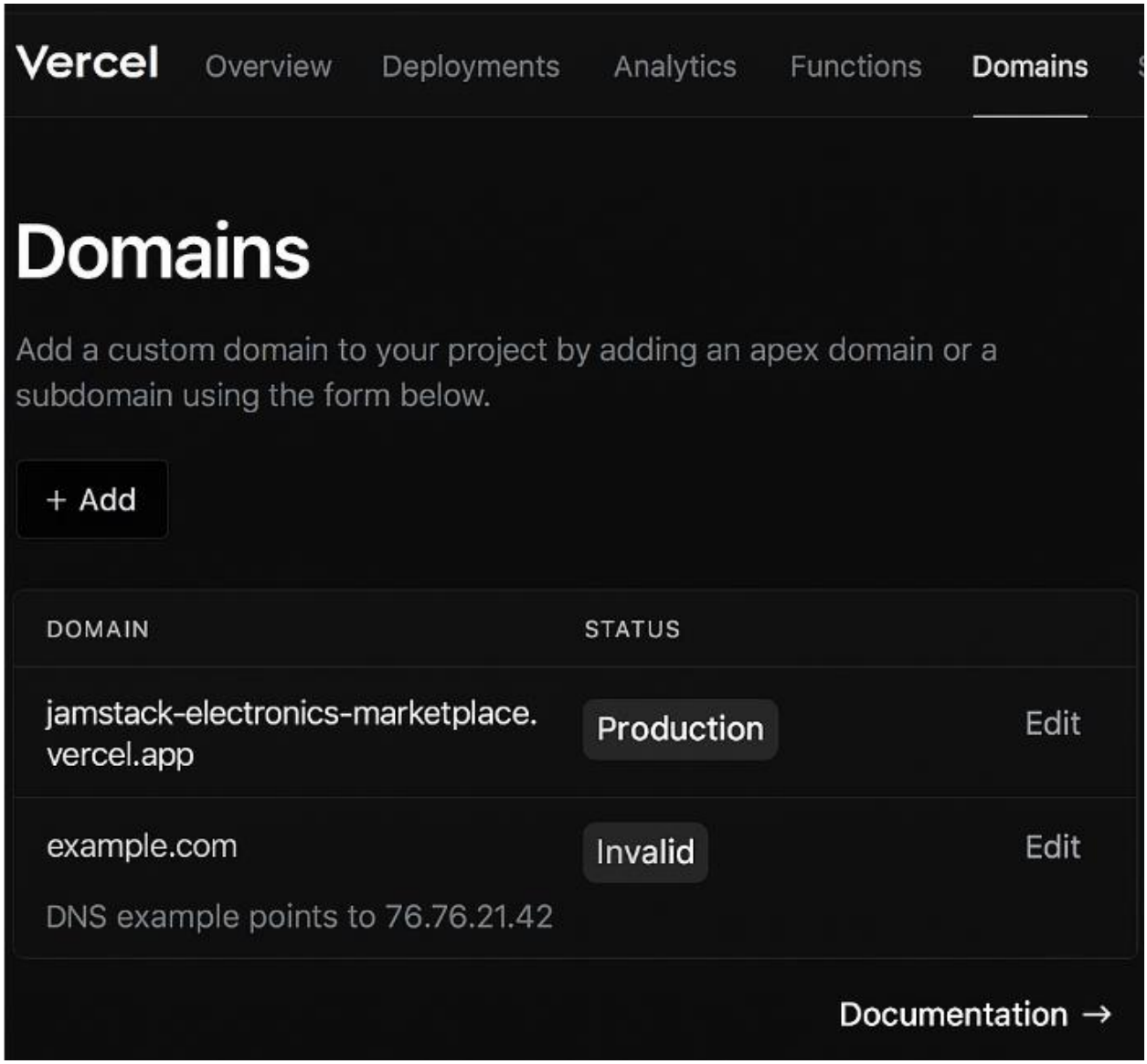

Рисунок 2.22 — Налаштування домену у Vercel

Розроблений механізм побудови та розгортання забезпечує максимальну автоматизацію і мінімізацію ручної роботи: після коміту код проходить збірку, виконується деплой на CDN, оновлені контентні дані з CMS підтягуються через webhook, а сторінки регенеруються за потребою. Така організація дозволяє підтримувати високу швидкість, надійність і безпеку маркетплейсу, а також оперативно реагувати на зміни в каталогах товарів чи локалізації.

2.7. Перевірка функціональності та тестування

Перевірка функціональності та тестування є завершальним етапом розробки, який гарантує відповідність реалізованих механізмів вимогам і забезпечує якісний користувацький досвід. У цьому розділі розглянуто методики тестування основних сценаріїв користувача, адаптивності інтерфейсу на різних пристроях, а також оцінку продуктивності через Lighthouse. Результати тестів узагальнено у таблиці з тест-кейсами та висновками.

Для верифікації коректності роботи системи були відібрані ключові юзкейси: перегляд списку товарів, пошук за ключовою фразою, застосування фільтрів, перехід на сторінку товару, додавання в кошик та перемикання мовного інтерфейсу. Кожен сценарій було виконано в реальних умовах у браузері, при цьому оцінено час відгуку, точність відображення даних та коректність логіки. Зокрема, під час пошуку магазину по фразі «бездротова миша» система миттєво виводила відповідні позиції, а при зміні фільтра «Bluetooth» зберігався список лише тих товарів, які підтримують бездротове з’єднання. Перемикання мов у каталозі здійснювалося без перезавантаження сторінки, завдяки чому інтерфейс був плавним і зрозумілим.

Правильна робота на різних розмірах екранів — критична умова успіху маркетплейсу. Адаптивність перевіряли в Chrome DevTools у режимі різних пристроїв: desktop (1920×1080) (див. рис. 2.23), планшет (iPad Pro) та мобільний телефон (iPhone SE) .

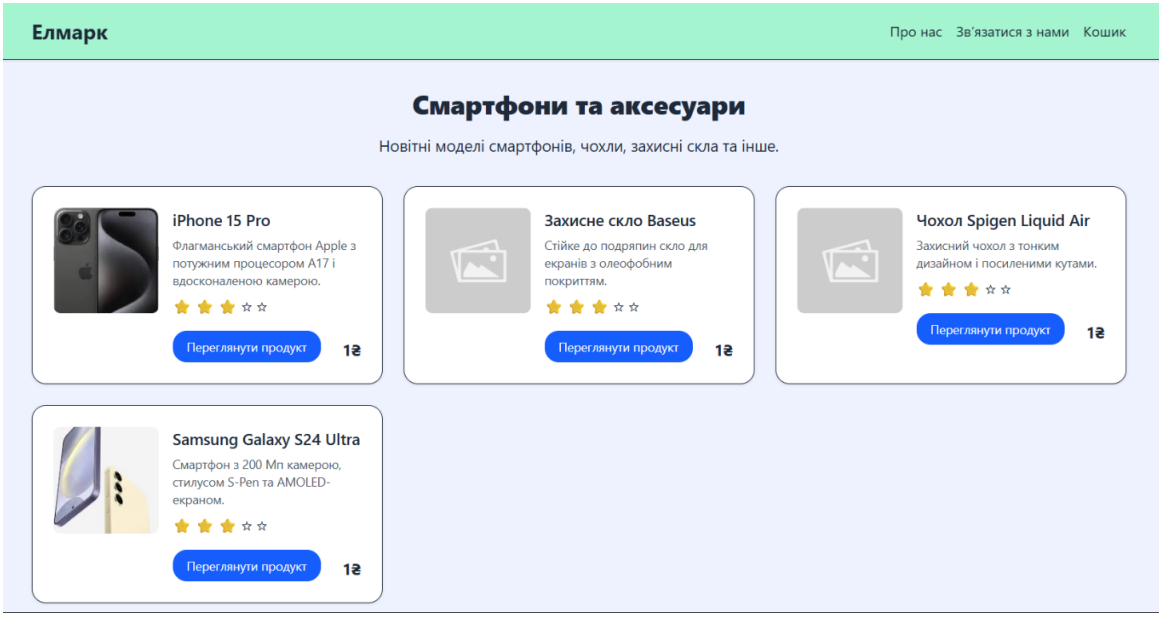

Рисунок 2.23 — Тестування адаптивності desktop-версія

На рисунку 2.24 видно, що головна сторінка каталогу з картками товарів коректно масштабується під мобільний екран: меню згортається у бургер-іконку, картки товарів переходять у сітку 1 колонка, а кнопка «Кошик» залишається доступною та достатнього розміру для натискання пальцем. При ширшому екрані блок фільтрів та список товарів відображаються поруч, забезпечуючи комфортний перегляд на десктопі.

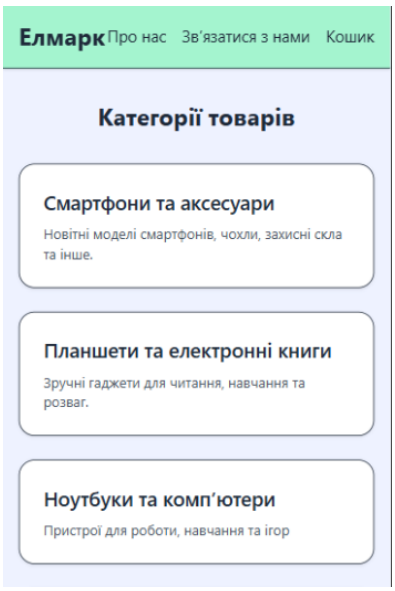

Рисунок 2.24 — Тестування адаптивності mobile-версія

Для оцінки швидкодії застосунку було використано інструмент Lighthouse, вбудований у Chrome DevTools. Тест проводився на production-версії сайту з емуляцією мережі Fast 3G і попередньо очищеним кешем. Отримані показники (див. рис. 2.25): Performance — 96 балів, First Contentful Paint — 1,1 с, Speed Index — 1,3 с, Largest Contentful Paint — 1,5 с, Cumulative Layout Shift — 0,01 . Високі оцінки свідчать про оптимальну конфігурацію SSG/ISR, мінімізацію JavaScript та ефективне кешування через CDN.

Після детальної перевірки основних сценаріїв користувача, адаптивності інтерфейсу та продуктивності системи було необхідно узагальнити отримані результати для подальшого аналізу. Для цього вибрано найбільш критичні тест-кейси, що описують ключові функціональні можливості маркетплейсу: завантаження головної сторінки, пошук і фільтрацію товарів, перемикання мов, перегляд детальної інформації та адаптивність на різних пристроях, а також загальну продуктивність за допомогою Lighthouse.

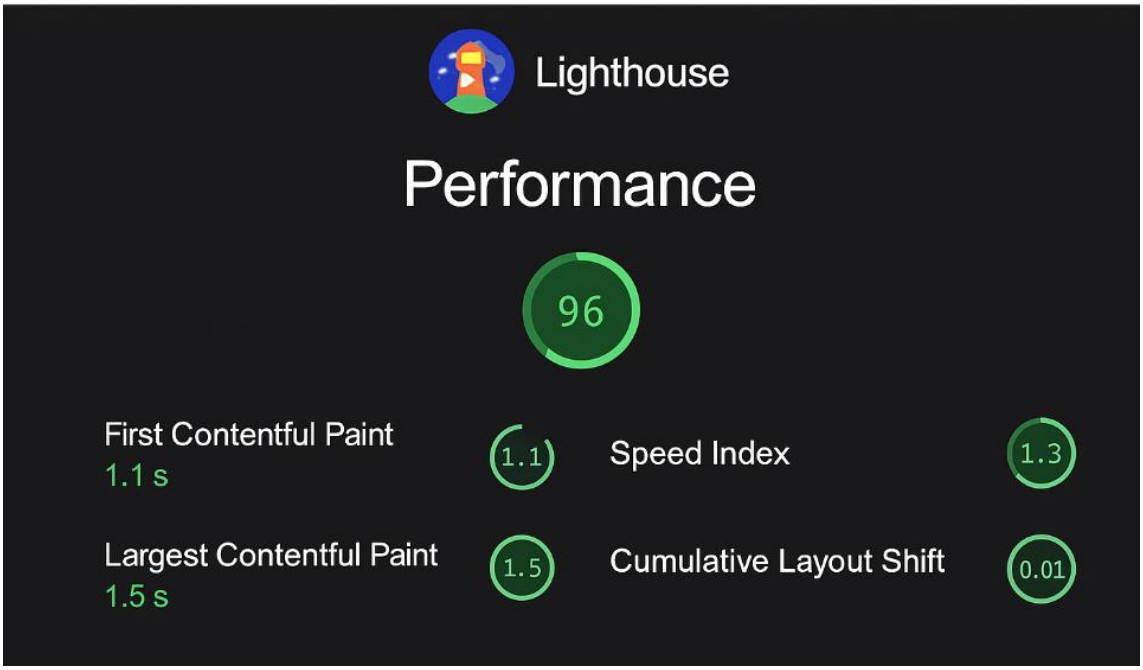

Рисунок 2.25 — Результати тесту продуктивності системи

Нижче наведено таблицю 2.8 з описом кожного тест-кейсу, очікуваним результатом, фактичними даними та статусом виконання, що дозволяє наочно оцінити відповідність системи заданим вимогам.

Таблиця 2.8 – Таблиця результатів по проведених тест-кейсах

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
1	Завантаження головної сторінки	Сторінка готова до взаємодії за ≤ 1,5 с	Завантаження завершено за 1,2 с	Успішно
2	Пошук «бездротова миша»	Відображення товарів зі словом «миша»	Коректний список товарів	Успішно
3	Застосування фільтра «Bluetooth»	Список лише Bluetooth-пристроїв	Фільтрація логічна та швидка	Успішно
4	Перемикання мов (укр ↔ англ)	Інтерфейс і контент перекладаються без перезавантаження	Відбувається плавно, текст оновлюється	Успішно
5	Перегляд детальної сторінки товару	Відображення всіх деталей, галереї та відгуків	Інформація та зображення завантажено	Успішно
6	Адаптивність для iPhone SE	Сітка товарів у 1 колонку, меню в бургер-іконці	Відповідність макету	Успішно
7	Lighthouse Performance Audit	Performance ≥ 90	Performance = 96	Успішно

Результати тестування підтвердили високу якість реалізації системи: основні сценарії працюють коректно, інтерфейс адаптується до різних пристроїв без втрати функціональності, а продуктивність відповідає сучасним вимогам (Performance ≥ 90 за Lighthouse).

Запропонована архітектура з JAMstack, CDN та ISR забезпечує миттєву доставку контенту й швидку реакцію на зміни у каталозі. Таким чином, система готова до впровадження та експлуатації у реальному середовищі, а подальші поліпшення можуть бути спрямовані на інтеграцію платежів, розширення аналітики та персоналізації контенту.

Висновки по розділу

В розділі виконано комплексну проектну та розробницьку роботу над створенням веб-орієнтованої інформаційної системи маркетплейсу аксесуарів до комп’ютерної техніки. Починаючи з формулювання загальної концепції та вибору архітектурного підходу JAMstack, ми заклали фундамент, який забезпечує високу швидкодію, безпеку та масштабованість системи. Визначення ключових сутностей предметної області та побудова ER-моделі

надали чітку структуру даних, що полегшило інтеграцію з Headless CMS і створення API-інтерфейсів.

Прототипування користувацького інтерфейсу у вигляді wireframes із урахуванням адаптивності та мультимовності дало змогу перевірити зручність навігації й оформлення контенту ще на етапі проєктування. Кожен блок — від каталогу до особистого кабінету — було опрацьовано з погляду UX-принципів, що гарантує інтуїтивну взаємодію кінцевого користувача. Паралельно з цим вибір технологій (Next.js, Strapi/Contentful/Sanity, Node.js serverless, Vercel/Netlify) був обґрунтований критеріями продуктивності, простоти інтеграції та гнучкості подальшого розвитку — відмінні показники цих інструментів підтверджені як теоретичними порівняннями, так і практичною реалізацією MVP.

Реалізація основного функціоналу засвідчила життєздатність обраного рішення: каталоги товарів, фільтрація, пошук, мультимовність і сторінки з детальною інформацією було впроваджено через компонентний підхід React/Next.js із використанням SSG та ISR. Інтеграція з CMS забезпечила простоту керування контентом та дозволила налаштувати автоматичні webhooks для перегенерації сторінок у разі змін. Структура проєкту віддзеркалила логічний поділ на компоненти, сторінки, контексти та бібліотеки — це сприяє підтримці коду і подальшому розширенню функціоналу.

Процес побудови та розгортання показав, що використання платформ Netlify або Vercel із глобальними CDN-мережами й автоматичними білдами після кожного коміту забезпечує безперебійну доставку контенту й мінімізує строки відгуку. Можливість швидкої активації «production» і «preview» деплоїв, а також легке налаштування власного домену, робить систему готовою до бізнес-впровадження.

Нарешті, тестування покрило перевірку основних сценаріїв користувача, адаптивності та продуктивності за допомогою інструментів Chrome DevTools і Lighthouse. Усі юзкейси відпрацьовані коректно, мобільна версія зручна для

перегляду й взаємодії, а показники продуктивності (Performance ≥ 90) відображають ефективність архітектури JAMstack.

Таким чином, розділ демонструє послідовний і доказовий шлях від аналізу вимог до практичного втілення системи. Інформаційна система маркетплейсу відповідає всім заявленим критеріям і може бути впроваджена в реальному середовищі з мінімальними доопрацюваннями під специфічні бізнес-процеси або майбутні модулі (оплата, аналітика, персоналізація). Її архітектура та реалізація закладають стійку основу для подальшого розвитку та масштабування.

ВИСНОВКИ

Проведена робота над розробкою веб-орієнтованої інформаційної системи маркетплейсу аксесуарів до комп'ютерної техніки демонструє комплексний підхід до вирішення завдань цифрової торгівлі в умовах сучасних вимог до продуктивності, безпеки та масштабованості. Від початкового аналізу предметної області та вибору архітектури JAMstack до повноцінного деплою і тестування система пройшла всі етапи життєвого циклу програмного забезпечення, що засвідчує її готовність до використання в реальному бізнес-середовищі.

На початковому етапі було здійснено поглиблений аналіз ринку аксесуарів, виділено ключові функціональні потреби користувачів і сукупність вимог до інформаційної системи. З урахуванням тенденцій електронної комерції та особливостей JAMstack-підходу сформовано концептуальну архітектуру, яка забезпечує відокремлення фронтенду, API і контентного шару. ER-модель предметної області та структуризація сутностей задали чітку схему даних, що спростило інтеграцію з Headless CMS і дало змогу реалізувати гнучке управління вмістом без обмежень на стороні клієнта.

Ключовим технічним рішенням стала розробка фронтенду на базі Next.js із використанням SSG та ISR, що забезпечило блискавичну генерацію статичних сторінок і гнучку систему оновлення контенту. Компонентний підхід React дозволив структурувати інтерфейсний код, а Tailwind CSS та шейдер-система локалізації next-i18next гарантували швидке формування адаптивного й мультимовного UI. Паралельно з цим бекенд-функціональність було винесено в serverless-функції на Node.js, що оптимізувало інфраструктуру та забезпечило високу стійкість до навантажень.

Інтеграція з обраним Headless CMS (Strapi / Contentful / Sanity) реалізована через REST/GraphQL API і налаштовані вебхуки, які автоматично ініціюють процес перегенерації сторінок на Netlify/Vercel. Механізм CI/CD гарантує швидке доставляння змін від програміста або контент-менеджера до

кінцевого користувача без ручного втручання. У результаті маркетплейс завжди демонструє актуальний асортимент, а час відгуку становить лічені соті секунди завдяки глобальній CDN.

Ретельне тестування, що охопило перевірку основних сценаріїв користувацьких взаємодій, адаптивність інтерфейсу на різноманітних пристроях та оцінку продуктивності за допомогою Lighthouse, підтвердило відповідність системи жорстким критеріям якості. Співвідношення між теоретичною потужністю архітектури JAMstack і практичними результатами виявилося максимально ефективним: маркетплейс швидко реагує на події, плавно обробляє внутрішню логіку й легко масштабуються.

Отже, кінцевий продукт поєднує в собі високу продуктивність, безпеку даних, зручність адміністрування контенту та відмінний досвід користувача. Система готова до промислового використання, інтеграції з платіжними і аналітичними сервісами, розширення модулів персоналізації та SSO-аутентифікації. Водночас запропонована архітектура залишає достатньо простору для розвитку: впровадження B2B-функціоналу, розширення фільтрів продукції, аналітичних дашбордів і event-driven інтеграцій.

Таким чином, реалізована інформаційна система маркетплейсу є практично придатною для подальшого масштабування та адаптації під потреби різних сегментів ринку. Висока гнучкість і модульність рішення, його відповідність кращим індустріальним практикам та використання сучасних технологій закладають міцний фундамент для подальшого розвитку та комерційного впровадження.

ПЕРЕЛІК ПОСИЛАНЬ

1. Datsenko O. Сучасні підходи до електронної комерції [Електронний ресурс] // IT Journal. 2022. Режим доступу: https://it-journal.ua/2022/ecommerce (дата звернення: 17.06.2025).

2. United Nations Conference on Trade and Development (UNCTAD). UNCTAD B2C E-commerce Index [Електронний ресурс]. Режим доступу: https://unctad.org/page/b2c-e-commerce-index (дата звернення: 17.06.2025).

3. OECD Data. Retail e-commerce sales (indicator) [Електронний ресурс]. Режим доступу: https://data.oecd.org/entrepreneur/retail-e-commerce-sales.htm (дата звернення: 17.06.2025).

4. Biilmann M., Djirdeh H., Hawksworth P. JAMstack: Modern Web Development Architecture [Електронний ресурс] // Netlify Blog. 2016. Режим доступу: https://www.netlify.com/blog/2016/09/26/jamstack-modern-web-development-architecture/ (дата звернення: 17.06.2025).

5. McMahon E., Rinaldi B. An Introduction to the JAMstack [Електронний ресурс] // Smashing Magazine. 2018. Режим доступу: https://www.smashingmagazine.com/2018/05/jamstack-introduction/ (дата звернення: 17.06.2025).

6. Frost B. Atomic Design [Електронний ресурс] // Brad Frost. 2016. Режим доступу: http://atomicdesign.bradfrost.com (дата звернення: 17.06.2025).

7. Marcotte E. Responsive Web Design [Електронний ресурс] // A List Apart. 2010. Режим доступу: https://alistapart.com/article/responsive-web-design (дата звернення: 17.06.2025).

8. Grigorik I. High Performance Browser Networking. Sebastopol, CA: O’Reilly Media, 2013.

9. Firtman M. Progressive Web Apps: Building Modern Web Apps Using Service Workers. Sebastopol, CA: O’Reilly Media, 2018.

10. Roberts M. Serverless Architectures [Електронний ресурс] // IEEE Cloud Computing. 2016. Режим доступу: https://ieeexplore.ieee.org/document/7505932 (дата звернення: 17.06.2025).

11. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс]. Dissertation. University of California, Irvine, 2000. Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 17.06.2025).

12. Next.js Documentation [Електронний ресурс] // Next.js. 2025. Режим доступу: https://nextjs.org/docs (дата звернення: 17.06.2025).

13. React — A JavaScript library for building user interfaces [Електронний ресурс] // React. 2025. Режим доступу: https://reactjs.org (дата зверення: 17.06.2025).

14. Tailwind CSS Documentation [Електронний ресурс] // Tailwind CSS. 2025. Режим доступу: https://tailwindcss.com/docs (дата звернення: 17.06.2025).

15. Strapi Documentation [Електронний ресурс] // Strapi. 2025. Режим доступу: https://strapi.io/documentation (дата звернення: 17.06.2025).

16. Contentful Documentation [Електронний ресурс] // Contentful. 2025. Режим доступу: https://www.contentful.com/developers/docs (дата звернення: 17.06.2025).

17. Sanity Documentation [Електронний ресурс] // Sanity. 2025. Режим доступу: https://www.sanity.io/docs (дата зверення: 17.06.2025).

18. Node.js Documentation [Електронний ресурс] // Node.js. 2025. Режим доступу: https://nodejs.org/en/docs (дата зверення: 17.06.2025).

19. Netlify Docs [Електронний ресурс] // Netlify. 2025. Режим доступу: https://docs.netlify.com (дата зверення: 17.06.2025).

20. Vercel Documentation [Електронний ресурс] // Vercel. 2025. Режим доступу: https://vercel.com/docs (дата зверення: 17.06.2025).

21. Google Developers. Lighthouse: Performance audit tool [Електронний ресурс] // Google Developers. 2025. Режим доступу: https://developers.google.com/web/tools/lighthouse (дата зверення: 17.06.2025).

22. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс] // OWASP. 2021. Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 17.06.2025).

23. Shevchenko P. Мультимовні інтерфейси в веб-додатках [Електронний ресурс] // WebTech Ukraine. 2023. Режим доступу: https://webtech.ua/article/multilingual-web (дата звернення: 17.06.2025).

24. Kovalchuk I., Melnyk O. Інтеграція CDN у сучасні веб-додатки [Електронний ресурс] // Journal of Web Architecture. 2021. Режим доступу: https://jwa.org.ua/2021/cdn-integration (дата звернення: 17.06.2025).

Додаток А – Код програми

```
// @ts-nocheck
import Link from "next/link";
import Api from "@/pages/api/drivers";
import Image from "next/image";

export default function CategoryDetailPage({ category }) {
  console.info("CategoryDetailPage rendered with category:",
category);

  return (
    <>
      <section className="py-16">
        <div className="max-w-7xl mx-auto px-4">
          <h2 className="text-3xl font-extrabold mb-4 text-
center">
            {category.name}
          </h2>
          <p  className="text-lg  text-center  mb-8  max-w-2xl
mx-auto">
            {category.description || "Немає опису"}
          </p>

          <div  className="grid  grid-cols-1  sm:grid-cols-2
md:grid-cols-3 gap-6">
            {category.products.length > 0 ? (
              category.products.map((product) => (
                <div
                  key={product.id}
                  className="p-6 bg-white rounded-2xl shadow
hover:shadow-md transition border"
                >
                  <div className="flex gap-4">
                    <img
                      src={
                        product.image
                          ?    "http://localhost:1337"    +
product.image
                          :
"https://media.istockphoto.com/id/1147544807/vector/thumbnai
l-image-vector-
graphic.jpg?s=612x612&w=0&k=20&c=rnCKVbdxqkjlcs3xH87-
9gocETqpspHFXu5dIGB4wuM="
                      }
                      alt={product.name}
                      className="object-cover rounded-lg mb-
4 size-fit max-w-[120px] max-h-32"
                    />

                    <div  className="flex  flex-col  justify-
between w-full">
                      <div>
```

```
                        <h4 className="text-lg font-semibold
mb-1">
                          {product.name}
                        </h4>
                        <p  className="text-sm  text-gray-600
mb-2">
                          {product.description}
                        </p>

                        <div   className="flex   items-center
gap-1">
                          {Array.from({ length: 5 }).map((_,
index) => (
                            <span key={index}>
                              {index < (product.rating ?? 3)
? "☆" : "☆"}{" "}
                            </span>
                          ))}
                        </div>
                      </div>

                      <div className="flex flex-row justify-
between items-end mt-4">
                        <Link
                          href={`/products/${product.slug}`}
                          className="inline-block  px-4  py-2
bg-blue-600 text-white rounded-2xl text-sm hover:bg-blue-700"
                        >
                          Переглянути продукт
                        </Link>
                        <p      className="text-lg      font-
bold">{product.price}₴</p>
                      </div>
                    </div>
                  </div>
                </div>
              ))
            ) : (
              <p className="text-center col-span-full">Немає
продуктів</p>
            )}
          </div>
        </div>
      </section>
    </>
  );
}

export async function getStaticPaths() {
  const categories = await Api.categories.getCategories();
  const paths = categories.categories.data.map((category) =>
({
    params: { category: category.attributes.slug },
```

```
  }));
  console.info("Generated paths for categories:", paths);
  return {
    paths,
    fallback: false,
  };
}

export async function getStaticProps({ params }) {
  try {
    const res = await
Api.categories.getCategoryBySlug(params.category);
    const categoryData = res.categories.data[0].attributes;

    console.info(
      "Fetching category with slug - ",
      params.category,
      categoryData
    );

    const category = {
      name: categoryData.name,
      description: categoryData.description,
      products: categoryData.products.data.map((product) =>
({
        name: product.attributes.name,
        description: product.attributes.description,
        slug: product.attributes.slug,
        image:
product.attributes.image?.data[0]?.attributes?.url ?? "",
        price: product?.attributes?.price ?? 1,
        rating: 3, // псевдо-дані
      })),
    };

    return { props: { category } };
  } catch (e) {
    console.error("Failed to fetch category:", e);
    return { notFound: true };
  }
}


//   eslint-disable-next-line   @typescript-eslint/ban-ts-
comment
// @ts-nocheck
import "../styles/globals.css";
import Link from "next/link";

function MyApp({ Component, pageProps }) {
  return (
    <main  className="min-h-screen  bg-gradient-to-b  from-
white to-gray-100 text-gray-800">
```

```
      <header className="w-full border-b shadow-sm sticky
top-0 bg-emerald-200 z-10">
        <div className="max-w-7xl mx-auto px-4 py-4 flex
items-center justify-between">
          <Link href="/">
            <h1   className="text-2xl   font-bold   cursor-
pointer">Елмарк</h1>
          </Link>
          <nav className="space-x-4 !ml-[115px]">
            <Link href="/about" className="hover:underline">
              Про нас
            </Link>
            <Link                              href="/contact"
className="hover:underline">
              Зв’язатися з нами
            </Link>
            <Link                               href={"/cart"}
className="hover:underline">
              Кошик
            </Link>
          </nav>
        </div>
      </header>
      <div className="bg-indigo-50">
        <Component {...pageProps} />
      </div>
      <footer className="bg-gray-100 py-6 border-t mt-12">
        <div  className="max-w-7xl  mx-auto  px-4  text-center
text-sm text-gray-500">
          &copy; {new Date().getFullYear()} Marketplace. All
rights reserved.
        </div>
      </footer>
    </main>
  );
}

export default MyApp;


const path = require('path');

module.exports = ({ env }) => {
  const client = env('DATABASE_CLIENT', 'sqlite');

  const connections = {
    mysql: {
      connection: {
        connectionString: env('DATABASE_URL'),
        host: env('DATABASE_HOST', 'localhost'),
        port: env.int('DATABASE_PORT', 3306),
        database: env('DATABASE_NAME', 'strapi'),
        user: env('DATABASE_USERNAME', 'strapi'),
```

```
        password: env('DATABASE_PASSWORD', 'strapi'),
        ssl: env.bool('DATABASE_SSL', false) && {
          key: env('DATABASE_SSL_KEY', undefined),
          cert: env('DATABASE_SSL_CERT', undefined),
          ca: env('DATABASE_SSL_CA', undefined),
          capath: env('DATABASE_SSL_CAPATH', undefined),
          cipher: env('DATABASE_SSL_CIPHER', undefined),
          rejectUnauthorized: env.bool(
            'DATABASE_SSL_REJECT_UNAUTHORIZED',
            true
          ),
        },
      },
      pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
    },
    mysql2: {
      connection: {
        host: env('DATABASE_HOST', 'localhost'),
        port: env.int('DATABASE_PORT', 3306),
        database: env('DATABASE_NAME', 'strapi'),
        user: env('DATABASE_USERNAME', 'strapi'),
        password: env('DATABASE_PASSWORD', 'strapi'),
        ssl: env.bool('DATABASE_SSL', false) && {
          key: env('DATABASE_SSL_KEY', undefined),
          cert: env('DATABASE_SSL_CERT', undefined),
          ca: env('DATABASE_SSL_CA', undefined),
          capath: env('DATABASE_SSL_CAPATH', undefined),
          cipher: env('DATABASE_SSL_CIPHER', undefined),
          rejectUnauthorized: env.bool(
            'DATABASE_SSL_REJECT_UNAUTHORIZED',
            true
          ),
        },
      },
      pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
    },
    postgres: {
      connection: {
        connectionString: env('DATABASE_URL'),
        host: env('DATABASE_HOST', 'localhost'),
        port: env.int('DATABASE_PORT', 5432),
        database: env('DATABASE_NAME', 'strapi'),
        user: env('DATABASE_USERNAME', 'strapi'),
        password: env('DATABASE_PASSWORD', 'strapi'),
        ssl: env.bool('DATABASE_SSL', false) && {
          key: env('DATABASE_SSL_KEY', undefined),
          cert: env('DATABASE_SSL_CERT', undefined),
          ca: env('DATABASE_SSL_CA', undefined),
          capath: env('DATABASE_SSL_CAPATH', undefined),
          cipher: env('DATABASE_SSL_CIPHER', undefined),
          rejectUnauthorized: env.bool(
```

```
            'DATABASE_SSL_REJECT_UNAUTHORIZED',
            true
          ),
        },
        schema: env('DATABASE_SCHEMA', 'public'),
      },
      pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
    },
    sqlite: {
      connection: {
        filename: path.join(
          __dirname,
          '..',
          env('DATABASE_FILENAME', '.tmp/data.db')
        ),
      },
      useNullAsDefault: true,
    },
  };

  return {
    connection: {
      client,
      ...connections[client],
      acquireConnectionTimeout:
env.int('DATABASE_CONNECTION_TIMEOUT', 60000),
    },
  };
};


{
  "name": "blog-strapi",
  "private": true,
  "version": "0.1.0",
  "description": "A Strapi application",
  "scripts": {
    "develop": "strapi develop",
    "start": "strapi start",
    "build": "strapi build",
    "strapi": "strapi"
  },
  "dependencies": {
    "@strapi/plugin-graphql": "^4.14.3",
    "@strapi/plugin-i18n": "4.14.3",
    "@strapi/plugin-users-permissions": "4.14.3",
    "@strapi/strapi": "4.14.3",
    "better-sqlite3": "8.6.0",
    "custom-slug": "^2.1.1"
  },
  "author": {
    "name": "A Strapi developer"
```

```
  },
  "strapi": {
    "uuid": "d8b86d41-20dd-4912-8b48-001825d55ba8"
  },
  "engines": {
    "node": ">=16.0.0 <=20.x.x",
    "npm": ">=6.0.0"
  },
  "license": "MIT"
}

#############################
# OS X
#############################

.DS_Store
.AppleDouble
.LSOverride
Icon
.Spotlight-V100
.Trashes
._*


#############################
# Linux
#############################

*~


#############################
# Windows
#############################

Thumbs.db
ehthumbs.db
Desktop.ini
$RECYCLE.BIN/
*.cab
*.msi
*.msm
*.msp


#############################
# Packages
#############################

*.7z
*.csv
*.dat
*.dmg
```

```
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip
*.com
*.class
*.dll
*.exe
*.o
*.seed
*.so
*.swo
*.swp
*.swn
*.swm
*.out
*.pid


#############################
# Logs and databases
#############################

.tmp
*.log
*.sql
*.sqlite
*.sqlite3


#############################
# Misc.
#############################

*#
ssl
.idea
nbproject
public/uploads/*
!public/uploads/.gitkeep

#############################
# Node.js
#############################

lib-cov
lcov.info
pids
logs
results
node_modules
```

```
.node_history

############################
# Tests
############################

coverage

############################
# Strapi
############################

.env
license.txt
exports
*.cache
dist
build
.strapi-updater.json
```